Parte 1 — Machine Learning Clásico

50 clases · Parte 1 del programa

Parte 1 — Machine Learning Clásico

> *50 clases · bundle consolidado del currículo v3.*

Índice de clases

- Clase 090 — Clase 090 — Stacking (stacked generalization)
- Clase 091 — Clase 091 — La maldición de la dimensionalidad
- Clase 092 — Clase 092 — PCA: proyección, varianza explicada, incremental, randomized, kernel
- Clase 093 — Clase 093 — LLE (Locally Linear Embedding)
- Clase 094 — Clase 094 — MDS, Isomap, t-SNE, UMAP, LDA
- Clase 095 — Clase 095 — Clustering K-Means: selección de K, MiniBatch
- Clase 096 — Clase 096 — DBSCAN
- Clase 097 — Clase 097 — Agglomerative, BIRCH, Mean Shift, Affinity Propagation, Spectral
- Clase 098 — Clase 098 — Gaussian Mixture Models
- Clase 099 — Clase 099 — Detección de anomalías: Isolation Forest, LOF, One-Class SVM

Clase 050 — Clase 050 — Panorama del ML: tipos, batch vs online, instance vs model-based

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 1. Duración estimada: 60 min.*

Objetivo

Que el alumno arme un mapa mental claro del campo de ML antes de entrar en algoritmos concretos: qué tipos de aprendizaje existen, en qué se diferencia entrenar de una sola vez (batch) vs en streaming (online), y por qué algunos modelos "memorizan ejemplos" (instance-based) y otros "abstraen una regla" (model-based). Sirve de andamio para ubicar cada algoritmo de las próximas clases dentro de esta taxonomía.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Clasificar un problema dado como supervisado, no supervisado, semi-supervisado o reforzado, justificando con la pinta de los datos (¿hay etiquetas?, ¿hay recompensa?).
2. Decidir batch vs online según el volumen de datos, la velocidad a la que cambia la distribución y el costo de reentrenar.
3. Distinguir instance-based de model-based y reconocer cuál usa scikit-learn por debajo en un algoritmo dado (KNN vs regresión lineal, por ejemplo).
4. Detectar señales de mala generalización (overfitting / data drift) en términos del marco del cap. 1 — anticipo de la clase 048.
5. Ubicar cualquier algoritmo del curso dentro de la grilla (supervisión × batch/online × instance/model) sin googlear.

Temas

#	Tema	Por qué importa
1	Qué es ML (Samuel 1959, Mitchell 1997)	Definición operativa: aprender = mejorar e
2	Aprendizaje supervisado	El 80% de lo que vas a tocar en la industr
3	Aprendizaje no supervisado	Clustering, reducción de dim, detección de
4	Semi-supervisado y auto-supervisado	Etiquetar es caro; mezclás un poco de labe

5	Aprendizaje por refuerzo	Otro paradigma (agente + recompensa); útil
6	Batch vs online learning	Reentrenar de cero vs aprender incremental
7	Instance-based vs model-based	Dos formas de generalizar: por similitud c

Definiciones y características

Aprendizaje supervisado

: El dataset trae cada instancia con su etiqueta (y). El algoritmo aprende un mapeo f: X → y. Sub-divisiones: clasificación (y categórica) y regresión (y continua). Ejemplos: spam filter, predicción de precio de casa.

Aprendizaje no supervisado

: Solo hay X, sin y. El algoritmo busca estructura: clusters (k-means), componentes (PCA), densidad (DBSCAN), anomalías (isolation forest). Métricas de éxito son menos directas que en supervisado.

Aprendizaje semi-supervisado

: Mezcla — pocas instancias etiquetadas + muchas sin etiqueta. Típico cuando etiquetar es caro (imagenes médicas, audio transcripto). Google Photos lo usa para clusterear caras y pedirte el nombre solo del cluster.

Aprendizaje por refuerzo (RL)

: Un agente observa el entorno, toma acciones, recibe recompensa (positiva o negativa), aprende una política que maximiza recompensa acumulada. Distinto de supervisado: no hay "respuesta correcta", solo señal de recompensa. Ej: AlphaGo, robots, trading bots.

Batch learning (offline)

: Se entrena con todo el dataset de una sola pasada. El modelo queda congelado en producción. Para incorporar datos nuevos hay que reentrenar desde cero. Simple, reproducible, pero pesado si el dataset crece o la distribución cambia.

Online learning (incremental)

: El modelo se actualiza con mini-batches o instancias una a la vez (partial_fit en sklearn). Ideal para streaming (precios, clicks) o datasets que no caben en RAM (out-of-core). El parámetro clave es el learning rate: alto = se adapta rápido pero olvida; bajo = estable pero lento.

Instance-based learning

: El sistema "memoriza" las instancias de entrenamiento y generaliza comparando una nueva instancia con las vistas, vía una medida de similitud. Ejemplo canónico: KNN. No hay parámetros aprendidos — el "modelo" es el propio dataset.

Model-based learning

: Se asume una forma funcional (lineal, árbol, red neuronal) con parámetros θ, y se ajustan los θ minimizando una función de costo sobre el train set. Una vez entrenado, podés tirar el dataset: el modelo es autocontenido. Ejemplos: regresión lineal/logística, random forest, redes neuronales.

Generalización

: Capacidad del modelo de funcionar bien en datos nuevos (no vistos en entrenamiento). Es el objetivo real de ML — no la performance en train. Se mide con un test set separado. Falla por dos motivos: overfitting (memorizó ruido) o underfitting (modelo demasiado simple). Tema central de la clase 048.

Dataset / recursos

Para los ejercicios prácticos: California Housing (sklearn.datasets.fetch_california_housing) — regresión, supervisado, ~20k filas. Y Iris (sklearn.datasets.load_iris) — clasificación clásica, 150 filas, perfecto para

comparar KNN (instance-based) vs LogisticRegression (model-based) en pocos segundos.

Lectura de fondo: Géron, cap. 1 entero (~25 páginas). Mirá especialmente las figuras 1-13 a 1-17, que son el resumen visual de toda la taxonomía.

Ejercicios

1. Clasificá 6 problemas reales. Para cada uno indicá supervisado/no-supervisado/semi/RL y por qué: (a) detectar fraude en transacciones con tarjeta; (b) segmentar clientes de un e-commerce; (c) traducir inglés→español; (d) jugar al ajedrez; (e) detectar caras duplicadas en una galería de 10k fotos donde solo 50 están taggeadas; (f) predecir el precio del dólar a 7 días.

2. Batch o online. Decidí qué corresponde y justificá en una línea: (a) modelo de scoring crediticio que se reentrena trimestralmente; (b) recomendador de noticias que reacciona a clicks en tiempo real; (c) clasificador de imágenes médicas en un hospital; (d) detector de spam en Gmail.

3. KNN vs LogReg en Iris. Entrená un KNeighborsClassifier(n_neighbors=5) y un LogisticRegression(max_iter=1000) sobre iris. Compará: accuracy en test, tamaño del modelo serializado (pickle.dumps), tiempo de inferencia sobre 1000 predicciones. ¿Cuál es instance-based? Verificalo mirando el tamaño.

4. Out-of-core con SGDRegressor. Cargá California Housing y entrená un SGDRegressor en mini-batches de 500 filas usando partial_fit en un loop. Plotteá el MSE en train a medida que pasan los batches. Es el patrón canónico de online learning.

5. Mapa mental. En papel (o draw.io), dibujá una grilla 2×2×2 con los ejes supervisión / batch-online / instance-model. Ubicá: KNN, regresión lineal, k-means, random forest, SGDClassifier, AlphaGo, autoencoder. Algunos van a quedar en celdas raras — anotá por qué.

Homework verificable

Notebook que: (a) carga Iris y California Housing; (b) entrena 4 modelos — KNeighborsClassifier, LogisticRegression, KNeighborsRegressor, SGDRegressor con partial_fit en 10 batches; (c) reporta para cada uno: tipo de aprendizaje (sup/no-sup), batch o online, instance o model-based, métrica en test (accuracy o RMSE) y tamaño en bytes del modelo serializado; (d) produce una tabla resumen en markdown al final del notebook.

Criterio de aceptación: La tabla final clasifica correctamente los 4 modelos en los 3 ejes y muestra que el KNN ocupa significativamente más bytes que LogReg para el mismo problema (evidencia empírica de que es instance-based).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
"Mi modelo da 99% en train y 60% en test"	Overfitting clásico. El modelo memorizó el
partial_fit da resultados muy distintos a	Online learning depende del orden de los b
KNeighborsClassifier tarda una eternidad e	Es instance-based — comparar contra todo e
"El test accuracy fluctúa cada vez que cor	No fijaste random_state en el train_test_s
Confundir "no supervisado" con "sin entren	k-means igual entrena (ajusta centroides).

Preguntas frecuentes

¿Semi-supervisado es lo mismo que self-supervised?

No. Semi-supervisado = pocas etiquetas + muchos datos sin etiquetar, en un problema supervisado clásico. Self-supervised (auto-supervisado) = el modelo se inventa la etiqueta a partir del propio dato (ej: BERT enmascara palabras y predice la enmascarada). Self-supervised es el motor de los LLMs modernos.

¿Cuándo elijo online learning en serio, no como juguete?

Tres casos: (1) streaming real donde los datos llegan continuamente y la distribución cambia (clicks, precios, sensores IoT); (2) out-of-core — el dataset no entra en RAM y entrenar offline en disco es prohibitivo; (3) adaptación rápida a concept drift sin reentrenar de cero. Para el 90% de problemas tabulares estáticos, batch es más simple y suficiente.

¿KNN realmente no entrena nada?

Técnicamente fit solo guarda el train set (con un índice como KD-tree opcional). Todo el trabajo se hace en predict. Por eso se le dice "lazy learner". Contraste total con regresión lineal, donde fit calcula los coeficientes y predict es una multiplicación.

¿Random forest es instance o model-based?

Model-based. Aunque "guarda árboles" — esos árboles son la forma funcional aprendida, no las instancias originales. Una vez fiteado, podés tirar el train set y el modelo sigue funcionando solo con los árboles. KNN no — sin el train set, KNN no existe.

¿Por qué arrancamos con este panorama antes de meternos en código?

Porque sin el mapa, cada algoritmo nuevo se siente desconectado del anterior. Con el marco del cap. 1, cuando lleguemos a SVM (clase ~055) ya sabés que es supervisado, batch, model-based — y eso te dice automáticamente qué API de sklearn esperar, cómo evaluarlo y cuáles son sus debilidades genéricas. El marco ahorra horas más adelante.

Referencias

- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (3rd ed., O'Reilly, 2022), cap. 1 — The Machine Learning Landscape.
- scikit-learn User Guide — Supervised learning
- scikit-learn User Guide — Unsupervised learning
- scikit-learn — Scaling strategies (out-of-core) — base teórica de partial_fit.
- Mitchell, T. Machine Learning (McGraw-Hill, 1997) — definición clásica de aprendizaje (T, P, E).
- Distill — A visual exploration of Gaussian Processes — bonus para entender model-based no paramétrico.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 051 — Clase 051 — Desafíos del ML: overfitting, underfitting, datos insuficientes

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 1. Duración estimada: 60 min.*

Objetivo

Que el alumno identifique los seis problemas que hacen fracasar un proyecto de ML — datos insuficientes, no representativos, de mala calidad, features irrelevantes, overfitting y underfitting — y sepa qué herramienta aplicar a cada uno (más datos, mejor muestreo, limpieza, feature engineering, regularización, o un modelo más expresivo). El eje conceptual es el bias-variance tradeoff y la intuición de que "el modelo memorizó vs. generalizó".

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Distinguir overfitting de underfitting mirando la brecha entre error de entrenamiento y error de validación.
2. Diagnosticar cuál de los seis desafíos de Géron está rompiendo un pipeline concreto.
3. Aplicar regularización (L1/L2, reducir capacidad del modelo, más datos) como contramedida al overfitting.
4. Detectar sampling bias y data snooping bias antes de medir performance.
5. Justificar por qué "más datos" suele ganarle a "modelo más complejo" (Banko & Brill 2001 / "The Unreasonable Effectiveness of Data").

Temas

#	Tema	Por qué importa
1	Datos insuficientes	Hasta el algoritmo más simple necesita vol
2	Datos no representativos (sampling bias)	Si el train no se parece al deploy, el mod
3	Datos de mala calidad (outliers, NaN, ruid	Garbage in, garbage out. La limpieza es 60
4	Features irrelevantes / feature engineerin	Modelos buenos con features malos pierden
5	Overfitting	El modelo aprendió el ruido del train. Baj
6	Underfitting	El modelo es demasiado rígido para captar
7	Regularización (L1/L2, early stopping, dro	Solución estándar al overfitting cuando no

Definiciones y características

Overfitting

: El modelo performa muy bien en train y mal en test. Memorizó ejemplos en vez de extraer patrones. Síntoma típico: train_score test_score. Causas: modelo demasiado complejo, pocas observaciones, ruido en los labels. Fixes: más datos, regularización, reducir capacidad, early stopping.

Underfitting

: El modelo es demasiado simple para la estructura del dato. Performa mal tanto en train como en test. Síntoma: ambos scores bajos y parecidos. Fix: modelo más expresivo, más features, menos regularización, o aceptar que la señal no está.

Regularización

: Restricción explícita sobre los parámetros del modelo para que no se ajuste al ruido. En lineales: penalizar la norma de los coeficientes (Ridge = L2, Lasso = L1). En árboles: limitar profundidad. En redes: dropout, weight decay. Controla la variance a costa de aceptar algo más de bias.

Bias-variance tradeoff

: Descomposición del error de generalización en tres términos: error = bias² + variance + irreducible_noise. Bias = error por supuestos del modelo (un lineal sobre una curva). Variance = sensibilidad a la muestra de

entrenamiento (cambias el train un poco y el modelo cambia mucho). Reducir uno suele subir el otro — el arte del ML es encontrar el sweet spot.

Sampling bias

: La muestra de entrenamiento no es representativa de la población. Caso clásico: encuesta Literary Digest 1936, predijo Landon ganador porque encuestaron a suscriptores con teléfono (sesgo socioeconómico). En ML moderno: entrenar reconocimiento facial con caras mayormente blancas y desplegarlo global.

Data snooping bias

: Mirar el test set para tomar decisiones de modelado (qué features incluir, qué hiperparámetros probar). Inflás artificialmente la performance reportada porque el "test" ya contaminó tus elecciones. Regla: separá test al inicio y no lo toques hasta el final.

Feature engineering

: Proceso de construir features informativos a partir del raw data. Incluye selección (descartar irrelevantes), extracción (combinar features existentes), y creación (fecha → día_de_semana, es_finde, mes). Históricamente, más palanca que cambiar de modelo.

Dataset / recursos

Demos sintéticas con sklearn.datasets.make_regression y make_classification para visualizar curvas de aprendizaje (train vs. validation a medida que crece N) y curvas de validación (score vs. hiperparámetro de capacidad). No hace falta dataset externo — la clase es conceptual + diagnóstica.

Ejercicios

1. Diagnóstico visual. Generá un dataset con make_regression(n_samples=50, noise=20). Ajustá PolynomialFeatures(degree=15) + LinearRegression. Compará train_score y test_score. ¿Es overfitting o underfitting? Repetí con degree=1 sobre datos no lineales.

2. Learning curve. Usá sklearn.model_selection.learning_curve sobre un dataset. Plotea train_score y val_score vs. tamaño del train. Identificá: (a) si la brecha se cierra con más datos → vale la pena conseguir más, (b) si las dos convergen bajo → modelo demasiado simple.

3. Ridge vs. Lasso. Sobre el mismo dataset polinomial del ej. 1, ajustá Ridge(alpha=...) para alpha [0.001, 0.01, 0.1, 1, 10, 100]. Plotea train_score y test_score vs. alpha (curva de validación). Encontrá el sweet spot.

4. Sampling bias simulado. Generá un dataset clasificación binaria balanceado (n_samples=10000). Entrená un modelo con un train sesgado (90% clase 0, 10% clase 1) y testealo en un test balanceado. Reportá accuracy global y por clase. ¿Qué te dice el accuracy global solo?

5. Feature engineering manual. Para un dataset (fecha_timestamp, monto), predecí monto (a) usando solo timestamp como número, (b) extrayendo día_semana, mes, es_finde. Compará R^2 — la diferencia es el valor de feature engineering.

Homework verificable

Notebook que tome sklearn.datasets.fetch_california_housing, entrene tres modelos: (a) LinearRegression baseline, (b) LinearRegression sobre PolynomialFeatures(degree=3) sin regularizar, (c) Ridge(alpha=10) sobre las mismas polynomial features. Para cada uno reportá train_R^2 y test_R^2 con train_test_split(random_state=42). Plotea las tres barras lado a lado.

Criterio de aceptación: El modelo (b) debe mostrar overfitting evidente (train alto, test bajo o negativo). El modelo (c) debe tener test_R^2 mejor que (b) y train_R^2 menor que (b) — evidencia de que la regularización

trabajó. El test set se separa al principio y no se usa para tunear alpha (eso va por CV en el train).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
train_score = 0.99, test_score = 0.40	Overfitting clásico. Fix: regularizá (Ridg
Ambos scores bajos (0.30 train, 0.28 test)	Underfitting. Fix: modelo más expresivo, m
Reportás accuracy=0.95 en clases desbalanc	El modelo predice siempre la mayoritaria.
Tuneás alpha mirando el test set	Data snooping. El test ya no es test. Fix:
Ridge no cambia nada vs. LinearRegression	No escalaste los features. Ridge penaliza

Preguntas frecuentes

¿Más datos o modelo más complejo?

Más datos, casi siempre. Banko & Brill (2001) mostraron que algoritmos mediocres con mucho dato superan a algoritmos sofisticados con poco. Solo cuando estás bloqueado para conseguir más datos vale la pena pelear con la arquitectura.

¿Ridge o Lasso?

Ridge (L2) si querés conservar todos los features y solo bajarles la magnitud. Lasso (L1) si querés selección automática — fuerza coeficientes a 0 exactos. ElasticNet mezcla las dos.

¿Cómo sé si tengo "datos insuficientes"?

Plotea la learning curve. Si val_score sigue subiendo cuando agregás más train → te faltan datos. Si ya hizo plateau → el cuello de botella es otro (modelo, features, ruido).

¿El test set se puede usar más de una vez?

No para tomar decisiones. Sí al final, para reportar performance. Si tunear hiperparámetros y elegir features mirando el test, ese número deja de ser un estimador honesto. Para tuning usá CV sobre el train o un validation set aparte.

¿Overfitting siempre es malo?

Casi siempre. La excepción rara: cuando deploy y train son el mismo universo cerrado (overfitting a una tabla fija que nunca crece). En ML real, donde llegan datos nuevos, overfitting = modelo frágil que falla en producción.

Referencias

- Géron, cap. 1 § "Main challenges of machine learning" y § "Testing and validating".
- sklearn — Validation curves
- sklearn — Underfitting vs. overfitting
- Banko & Brill (2001), Scaling to very very large corpora for natural language disambiguation.
- Halevy, Norvig & Pereira (2009), The Unreasonable Effectiveness of Data.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 052 — Clase 052 — Testing, validación, hyperparameter tuning, no free lunch theorem

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 2 + sklearn user guide. Duración estimada: 70 min.*

Objetivo

Que el alumno entienda cómo medir generalización sin engañarse — separar train/val/test correctamente, usar cross-validation para estimar performance con poca varianza, tunear hiperparámetros con GridSearchCV / RandomizedSearchCV, y aceptar el no free lunch theorem: ningún modelo gana en todos los datasets.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Separar un dataset en train/validation/test y justificar por qué el test no se toca hasta el final.
2. Aplicar KFold y StratifiedKFold con cross_val_score para estimar generalización.
3. Tunear hiperparámetros con GridSearchCV y RandomizedSearchCV, leyendo cv_results_.
4. Reconocer cuándo KFold rompe (datos temporales, grupos) y elegir el splitter correcto.
5. Explicar el no free lunch theorem y por qué siempre conviene comparar varios modelos.

Temas

#	Tema	Por qué importa
1	Train / validation / test split	Test es sagrado: una sola medición al fina
2	KFold y StratifiedKFold	Estima generalización con menos varianza q
3	cross_val_score y cross_validate	Una línea, K métricas, intervalo de confia
4	GridSearchCV vs RandomizedSearchCV	Grid es exhaustivo, random escala mejor en
5	Pipeline + CV (evitar leakage del scaler)	Fitear el scaler dentro del fold, nunca af
6	No free lunch theorem	Ningún algoritmo domina en todo dataset —

Versión profundizada — 2026

El tema moderno que vivía como complemento dentro de esta clase ahora tiene clase propia dedicada con patrón completo, ejercicios y homework:

- Clase 049b — Validación temporal: TimeSeriesSplit, walk-forward, blocking

Definiciones y características

Train / validation / test split

: Train entrena, validation tunea hiperparámetros, test se mira una sola vez al final. Proporciones típicas: 60/20/20 o 70/15/15. En CV el val se reemplaza por los K folds, pero el test queda igualmente intocable.

Cross-validation (CV)

: Estimar generalización promediando K mediciones rotando qué fold es val. Reduce varianza vs un hold-out único. Costo: K veces más entrenamientos.

Hold-out

: Una sola partición train/val. Rápido pero ruidoso — con datasets chicos la métrica depende mucho de qué

muestras cayeron en val.

Hyperparameter

: Configuración del modelo que no se aprende del data (ej. max_depth, C, learning_rate). Se tunea en validation, nunca en test.

Generalization gap

: Diferencia entre score en train y score en val/test. Gap grande = overfitting. Gap chico pero score bajo = underfitting.

No free lunch theorem (Wolpert, 1996)

: Promediado sobre todos los problemas posibles, ningún algoritmo de ML es mejor que otro. En la práctica: no hay un "mejor modelo" universal; siempre hay que comparar (linear, RF, GBM, etc.) sobre tu dataset específico.

Leak temporal

: Cuando el train contiene información que en producción no estaría disponible al momento de predecir (típicamente, datos del futuro respecto al target). Mata silenciosamente proyectos de forecasting.

Dataset / recursos

- sklearn.datasets.load_breast_cancer para CV estratificado.
- sklearn.datasets.fetch_california_housing para tuning con GridSearchCV.
- Serie sintética con np.cumsum(np.random.randn(500)) para TimeSeriesSplit.

Ejercicios

1. Hold-out vs CV. Sobre breast_cancer, comparar el score de un único train_test_split (varios random_state) vs cross_val_score(cv=5). Mostrar que el hold-out varía ±0.03 entre seeds, CV mucho menos.

2. StratifiedKFold. Con un target desbalanceado 90/10, comparar KFold vs StratifiedKFold mirando la proporción de la clase minoritaria en cada fold.

3. GridSearchCV en pipeline. Pipeline([('scaler', StandardScaler()), ('svc', SVC())]) + GridSearchCV sobre C y gamma. Verificar que el scaler se fitea dentro de cada fold (no leakage).

4. RandomizedSearchCV. Mismo problema que (3) pero con RandomizedSearchCV(n_iter=30) y distribuciones (scipy.stats.loguniform). Comparar tiempo y best_score_.

5. TimeSeriesSplit. Generar serie con tendencia + ruido. Aplicar TimeSeriesSplit(n_splits=5) y entrenar Ridge con features lag-1, lag-7. Reportar score por fold. Repetir con KFold(shuffle=True) y mostrar que el score se infla artificialmente (leakage temporal).

Homework verificable

Notebook que sobre california_housing: (a) split 80/20 train/test con random_state=42; (b) GridSearchCV(cv=5) con RandomForestRegressor sobre n_estimators {100, 300} y max_depth {None, 10, 20}; (c) reportar best_params_ y best_score_; (d) evaluar el mejor modelo una sola vez sobre test; (e) comparar contra un baseline DummyRegressor(strategy='mean').

Criterio de aceptación: El test se evalúa una única vez al final del notebook. El RF supera al baseline en R^2 al menos por 0.4. cv_results_ queda exportado a CSV.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Score de CV altísimo pero en producción se	Leakage. Fix: scaler/encoder dentro del Pi
Usar KFold con datos temporales y obtener	Estás "viendo el futuro". Fix: TimeSeriesS
GridSearchCV tarda horas	Espacio de búsqueda demasiado grande. Fix:
Mirar el test set varias veces para "ajust	Estás haciendo overfitting del test → la m
Folds con clase minoritaria ausente (Value	KFold random en target desbalanceado. Fix:

Preguntas frecuentes

¿Cuándo TimeSeriesSplit en lugar de KFold?

Siempre que haya orden temporal con sentido predictivo: forecasting, logs, sensores, transacciones, métricas de negocio. Si las observaciones son intercambiables (tabular cross-sectional, ej. clasificar imágenes independientes), KFold/StratifiedKFold está bien. Regla mecánica: si tu target depende del tiempo y predecís hacia adelante, TimeSeriesSplit.

¿GridSearchCV o RandomizedSearchCV?

Grid si tenés ≤ ~50 combinaciones y querés exhaustividad. Random a partir de espacios grandes (>100 combos) o cuando hay hiperparámetros continuos (C, alpha, learning_rate): con 30-60 iteraciones random llegás cerca del óptimo con una fracción del costo (resultado teórico de Bergstra & Bengio 2012).

¿K=5 o K=10 en KFold?

K=5 es el default razonable (buen compromiso varianza/costo). K=10 cuando el dataset es chico y necesitás más muestras por fold. K=N (LOOCV) casi nunca: costo absurdo y varianza alta.

¿Por qué importa el no free lunch theorem en la práctica?

Porque te recuerda no quedarte con un solo modelo "favorito". Para cualquier dataset nuevo, comparar al menos: un baseline tonto (Dummy), un lineal (Ridge/LogReg), un árbol ensamble (RandomForest/GradientBoosting). El que gane depende del problema, no de tu intuición previa.

¿Puedo usar cross_val_score y después GridSearch sobre el mismo set?

Sí, pero entendé que estás haciendo CV anidada implícita. Lo correcto cuando querés estimar la performance del proceso de tuning: cross_val_score(GridSearchCV(...), X_train, y_train, cv=outer_cv). CV externa evalúa, CV interna tunea. Caro pero honesto.

Referencias

- Géron, Hands-On ML, cap. 2 — End-to-End ML Project (sección "Better Evaluation Using Cross-Validation").
- sklearn — Cross-validation user guide
- sklearn — TimeSeriesSplit
- sklearn — GridSearchCV / RandomizedSearchCV
- Bergstra & Bengio (2012), Random Search for Hyper-Parameter Optimization, JMLR.
- Wolpert (1996), The Lack of A Priori Distinctions Between Learning Algorithms.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 053 — Clase 053 — Validación temporal: TimeSeriesSplit, walk-forward, blocking

> *Parte: 1 — Machine Learning Clásico · Fuente: Bergmeir & Benítez (2012) + sklearn TimeSeriesSplit docs. Duración estimada: 70 min.*

Objetivo

Aplicar validación correcta para series temporales — donde KFold y train_test_split aleatorio causan leakage del futuro al pasado y métricas infladas. Cubrir TimeSeriesSplit, walk-forward validation (rolling y expanding), blocking para datos con dependencias intra-cluster, purged + embargoed CV (López de Prado, finanzas).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar sklearn.model_selection.TimeSeriesSplit(n_splits, max_train_size, test_size, gap).
- Implementar walk-forward rolling (ventana fija) y expanding (acumulativo).
- Detectar leakage cuando KFold se usa sobre datos temporales.
- Aplicar purged + embargoed K-Fold para evitar leakage por feature engineering con lags.
- Reportar métricas multi-fold con dispersión (no solo promedio).

Temas

- ¿Por qué KFold falla en series? Aleatorización mezcla pasado y futuro.
- TimeSeriesSplit: split secuencial, train siempre antes que test.
- Expanding vs rolling window.
- gap (embargo) para target leakage con lags.
- Purged CV (López de Prado): elimina overlap entre train y test.
- CombinatorialPurgedKFold para backtesting.

Definiciones y características

- TimeSeriesSplit(n_splits=5): divide la serie en N+1 chunks secuenciales; itera (train=primer N, test=último).
- Walk-forward expanding: train crece, test va avanzando.
- Walk-forward rolling: train tiene tamaño fijo, ventana se mueve.
- Embargo / gap: número de samples ignoradas entre train y test — evita leakage por features con lags.
- Purged CV: elimina del train cualquier sample cuyo target overlapped con el test.

Dataset / recursos

- Serie temporal sintética o seaborn.load_dataset('flights').
- Librerías: scikit-learn, pandas, mlxtend (alternativa con más options).

Ejercicios

1. TSSplit vs KFold leak: con serie sintética con tendencia, comparar score CV con KFold aleatorio vs TimeSeriesSplit. KFold infla.
2. Walk-forward expanding: tscv = TimeSeriesSplit(n_splits=5). Iterar y reportar score por fold.
3. Rolling window: con max_train_size=100, simular walk-forward de window fijo.
4. gap: con feature y_t-1 (target lag), aplicar gap=1 para evitar que test "vea" su propio target.
5. Score con dispersión: reportar mean ± std de RMSE por fold, no solo mean.

Homework verificable

Forecasting con XGBoost en serie de retail:

1. Feature engineering: lags 1, 7, 30 + rolling mean.
2. CV con TimeSeriesSplit(5, gap=30).
3. Reportar RMSE por fold + global.
4. Comparar contra KFold ingenuo — mostrar inflación.

Criterio de aceptación: KFold subestima RMSE en ≥ 30 %; TimeSeriesSplit da estimación realista que se sostiene en test final.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Métrica CV mucho mejor que producción	KFold sobre temporal. Fix: TimeSeriesSplit
Feature y_t-1 y CV sin gap	Target leak. Fix: gap = max_lag.
n_splits muy alto con serie corta	Folds muy chicos. Fix: 3-5 folds.
Reportar solo mean	Variabilidad oculta. Fix: mean ± std.
Rolling con max_train_size mal calibrado	Train muy chico → ruido. Fix: ≥ 1 ciclo es

Preguntas frecuentes

Expanding o rolling?

Expanding usa toda la historia (más data); rolling refleja "olvido" si crees que la dinámica cambia. Probá ambos.

TimeSeriesSplit con feature engineering sobre toda la serie?

Leak. Fix: features con rolling/lag calculadas con min_periods=window para no usar futuro.

Para crypto / trading?

Purged + embargoed (López de Prado Advances in Financial Machine Learning).

Hyperparameter tuning con TimeSeriesSplit?

GridSearchCV con cv=TimeSeriesSplit(5). Funciona idéntico.

Nested CV?

Recomendado para tuning + evaluation honesta. Outer TSSplit para reportar, inner para tunear.

Referencias

- Bergmeir & Benítez (2012), On the use of cross-validation for time series predictor evaluation.
- López de Prado (2018), Advances in Financial Machine Learning, cap. 7.
- sklearn.model_selection.TimeSeriesSplit.
- Hyndman & Athanasopoulos, Forecasting: Principles and Practice.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 054 — Clase 054 — Proyecto end-to-end: visión, datos, exploración, preparación

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 2 (California Housing). Duración estimada: 90 min.*

Objetivo

Que el alumno recorra la primera mitad de un proyecto de ML real de punta a punta: framear el problema en términos de negocio, conseguir los datos, hacer un EDA honesto, separar train/test sin contaminarse, y dejar el pipeline de preparación (limpieza, encoding, scaling) listo para entrenar — todo sobre el dataset California Housing del capítulo 2 de Géron.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Framear el problema en términos de negocio: tipo de tarea (regresión/clasificación), métrica, baseline.
2. Hacer un EDA reproducible: describe, info, hist, corr, scatter matrix, mapas geográficos.
3. Separar train/test correctamente con train_test_split estratificado por una variable clave (income bucket).
4. Construir un pipeline con Pipeline + ColumnTransformer que limpie, encode (OneHotEncoder) y escale (StandardScaler) en un solo objeto.
5. Evitar data leakage: todo cálculo (medias, encodings, scalers) se ajusta solo en train y se aplica en test.

Temas

#	Tema	Por qué importa
1	Framing del problema	Sin objetivo claro y métrica, cualquier mo
2	EDA: describe, hist, corr, geo plot	Conocer los datos antes de modelar.
3	Stratified split por income bucket	Test representativo, no muestreo aleatorio
4	Limpieza: NaN, outliers, tipos	El 60% del trabajo real.
5	Encoding categórico (OneHotEncoder, Ordina	Los modelos no comen strings.
6	Scaling (StandardScaler, MinMaxScaler)	Imprescindible para modelos sensibles a es
7	Pipelines + ColumnTransformer	Reproducible, sin leakage, deployable.

Versión profundizada — 2026

El tema moderno que vivía como complemento dentro de esta clase ahora tiene clase propia dedicada con patrón completo, ejercicios y homework:

- Clase 050b — Feature Engineering avanzado + MICE imputation

Definiciones y características

Pipeline (sklearn)

: Secuencia de transformaciones (nombre, estimator) que termina opcionalmente en un modelo. fit ajusta cada paso con la salida del anterior; transform/predict aplica en orden. Encadena todo en un objeto y previene leakage cuando se usa con CV.

ColumnTransformer

: Aplica transformadores distintos a subconjuntos de columnas (numéricas → scaler, categóricas → encoder) y concatena el resultado. Es el pegamento entre EDA y modelo.

Stratified split

: Separación train/test que preserva la distribución de una variable clave (target en clasificación, o un bucket de una numérica como income_cat). Evita que el test caiga sesgado por azar — crítico en N chico.

OneHotEncoder

: Convierte cada valor único de una categórica en una columna binaria. Default sklearn: sparse matrix. Parámetros clave: handle_unknown='ignore' (para categorías nuevas en test), drop='first' (para evitar multicolinealidad en modelos lineales).

StandardScaler

: Resta media y divide por desvío estándar (z-score). Necesario para modelos sensibles a escala: SVM, KNN, redes neuronales, regresión regularizada (Ridge/Lasso). Árboles no lo necesitan.

Target leakage

: Cualquier feature, encoding o estadístico que contenga información del target o del test al momento de entrenar. Inflará métricas en validación y colapsará en producción. Síntoma típico: AUC > 0.99 sospechoso.

SimpleImputer

: Imputación univariada (media/mediana/moda/constante). Default razonable para empezar; reemplazable por KNNImputer o IterativeImputer sin cambiar el resto del pipeline.

Métrica vs función de costo

: La métrica la elige el negocio (RMSE en dólares, recall en fraude); la función de costo la usa el optimizador internamente (puede ser distinta — MSE para entrenar, RMSE para reportar).

Dataset / recursos

California Housing (Géron cap. 2). 20 640 filas, 10 columnas, target = median_house_value. Disponible vía sklearn.datasets.fetch_california_housing() o el CSV del repo de Géron. Contiene una categórica (ocean_proximity) y una columna con NaN (total_bedrooms) — perfecta para practicar pipeline completo.

Ejercicios

1. EDA mínimo. Cargá el dataset y producí: df.info(), df.describe(), df.hist(bins=50, figsize=(12,8)). Identificá al menos 2 anomalías (cap visual de median_house_value, distribución skewed de population).

2. Stratified split por income bucket. Creá income_cat = pd.cut(df['median_income'], bins=[0, 1.5, 3, 4.5, 6, np.inf]) y usá StratifiedShuffleSplit para train/test 80/20. Verificá que la distribución de income_cat sea casi idéntica en train y test.

3. Target encoding sin leakage. Tomá una variable categórica (creá zipcode_fake a partir de buckets de lat/long si querés). Implementá target encoding con category_encoders.TargetEncoder ajustado solo en train. Compará RMSE de un RandomForestRegressor con (a) one-hot vs (b) target encoded.

4. KNNImputer vs SimpleImputer. Sobre total_bedrooms (que tiene NaN reales), comparen RMSE final del pipeline usando SimpleImputer(strategy='median') vs KNNImputer(n_neighbors=5). Reportá cuál ganó y en cuánto.

5. Pipeline completo. Armá un ColumnTransformer con: numéricas → SimpleImputer(median) + StandardScaler; categóricas → OneHotEncoder(handle_unknown='ignore'). Envolvelo en un Pipeline con

LinearRegression al final. fit en train, RMSE en test.

Homework verificable

Notebook con: (a) carga del dataset + EDA con al menos 4 gráficos; (b) stratified split por income bucket con verificación de proporciones; (c) pipeline ColumnTransformer (numéricas con KNNImputer+StandardScaler, categóricas con OneHotEncoder); (d) feature engineering manual con al menos 2 features derivadas (rooms_per_household, bedrooms_per_room); (e) baseline LinearRegression con RMSE reportado en train y test.

Criterio de aceptación: El pipeline se entrena con pipeline.fit(X_train, y_train) sin tocar X_test antes del scoring final. RMSE de test reportado. Sin warnings de sklearn por leakage o categorías nuevas.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Target encoding da AUC casi perfecto en tr	Encoding calculado sobre todo el dataset →
ValueError: Found unknown categories ['X']	El test trae una categoría nueva no vista
RMSE de test ridículamente bajo, idéntico	Imputaste/escalaste sobre pd.concat([train
fillna(df.mean()) baja drásticamente la va	Imputación univariada con media aplasta la
Modelo lineal con coeficientes raros (uno	Olvidaste el StandardScaler — features est

Preguntas frecuentes

¿Cuándo target encoding > one-hot?

Cuando la cardinalidad es alta (>~15 valores únicos) y existe relación monotónica entre categoría y target. One-hot con 1000 zipcodes te da 1000 columnas sparse y árboles que no convergen; target encoding te da 1 sola columna numérica informativa. Siempre con CV out-of-fold para no filtrar el target.

¿Necesito escalar si uso Random Forest o XGBoost?

No. Los árboles particionan por umbrales, son invariantes a transformaciones monotónicas. Sí lo necesitás para SVM, KNN, redes, regresión lineal/logística con regularización.

¿Stratified split en regresión?

Sí — pero estratificás por una versión bucketizada del target o de una feature clave (como income_cat en California Housing). StratifiedShuffleSplit no acepta target continuo directamente.

¿StandardScaler o MinMaxScaler?

StandardScaler por default (z-score, asume distribución aprox. normal). MinMaxScaler cuando necesitás bounds fijos [0,1] (redes con sigmoide, ciertos algoritmos de visión). RobustScaler si hay outliers fuertes.

¿Hago feature engineering antes o después del split?

Features que dependen solo de la fila (ratios, log, sin/cos): antes o después, da igual. Features que dependen de estadísticos agregados (target encoding, frequency encoding, z-scores globales): siempre después del split y fit solo en train.

Referencias

- Géron, cap. 2 — End-to-End Machine Learning Project (California Housing).
- sklearn — Pipeline y ColumnTransformer
- sklearn — impute (SimpleImputer, KNNImputer, IterativeImputer)

- sklearn — TargetEncoder
- category_encoders docs
- Rubin, D. B. (1976). Inference and missing data. Biometrika.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 055 — Clase 055 — Feature Engineering avanzado: target encoding + MICE imputation

> *Parte: 1 — Machine Learning Clásico · Fuente: Micci-Barreca (2001) target encoding + Van Buuren (2018) Flexible Imputation of Missing Data. Duración estimada: 85 min.*

Objetivo

Dominar feature engineering moderno más allá de one-hot y SimpleImputer: target encoding con regularización + cross-validation (evita leakage), KNNImputer y IterativeImputer (MICE) para imputación multivariada inteligente, y category_encoders library para codificaciones modernas (CatBoost, James-Stein, hashing).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Aplicar target encoding con CV (no leak): category_encoders.TargetEncoder(cv=5, smoothing=10.0).
- Aplicar KNNImputer: imputa basado en vecinos.
- Aplicar IterativeImputer (MICE) de sklearn — predice cada feature con modelo de las demás.
- Decidir entre métodos: SimpleImputer (baseline), KNN (correlaciones locales), MICE (multivariada).
- Reconocer leakage en target encoding y evitarlo con CV interno.

Temas

- Target encoding clásico + smoothing bayesiano.
- Leak en target encoding sin CV: features ven sus propios targets.
- KNNImputer (sklearn): nearest neighbors por filas.
- IterativeImputer / MICE: estima cada feature con regresión.
- category_encoders: CatBoost, James-Stein, target, hashing.
- Pipeline-safe imputation.

Definiciones y características

- Target encoding: reemplazar categoría con su mean target. Con smoothing: combina con global mean.
- Smoothing: enc = (n·mean_cat + k·global) / (n + k) — protege categorías raras.
- CV target encoding: cada fold usa encoding fitted en otros folds → sin leakage.
- KNNImputer: para cada NaN, promedia k vecinos en feature space.
- MICE: iterativo — predice cada feature con regresión usando las demás.
- CatBoostEncoder: variant de target encoding sin necesitar CV (ordering trick).

Dataset / recursos

- fetch_openml('credit-g') o California Housing con NaN inyectados.
- Librerías: category_encoders (pip install category_encoders), scikit-learn, pandas.

Ejercicios

1. Target encoding leak: encoding sobre train+test → métrica inflada. Mostrar.
2. Target encoding con CV: TargetEncoder(cv=5, smoothing=10) dentro de pipeline. Sin leak.
3. CatBoost encoder: alternativa sin CV. Comparar performance.
4. KNNImputer: con dataset con NaN, imputar con k=5; comparar contra SimpleImputer (mean).
5. MICE: IterativeImputer(estimator=BayesianRidge(), max_iter=10). Comparar.

Homework verificable

Sobre credit-g con NaN sintéticos (10 % de missing en 3 columnas):

1. Pipeline 1: SimpleImputer + OneHot + LogReg.
2. Pipeline 2: KNNImputer + TargetEncoder(CV) + LogReg.
3. Pipeline 3: MICE + CatBoostEncoder + LogReg.
4. Comparar accuracy + tiempo.

Criterio de aceptación: pipelines modernos (2, 3) superan SimpleImputer baseline en AUC por ≥ 0.5 pp; sin target leakage en target encoding.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Target encoding fitted sobre X completo →	Leak. Fix: TargetEncoder dentro de pipelin
MICE con max_iter=2 poco convergente	Fix: 10-20 iterations.
KNNImputer lento con N grande	O(N²) en distancia. Fix: KNN con n_neighbo
Smoothing=0 sobre categoría con n=1	Encoding = ese 1 target → overfit. Fix: sm
OneHot para 10k+ categorías	Curse of dimensionality. Fix: target encod

Preguntas frecuentes

Target encoding mejor que one-hot?

Para alta cardinalidad (> 50 categorías), sí — mucho. Para baja, comparable o peor.

MICE o KNN?

MICE mejor con features correlated (predice con modelo). KNN mejor con clusters locales. Probar.

Smoothing value?

Empírico — 5-30 típicamente. Más smoothing = más conservador hacia global mean.

category_encoders integrado en sklearn?

No, lib externa. sklearn tiene TargetEncoder desde 1.3, pero category_encoders tiene más options.

Para tree-based (XGBoost)?

Mucho menos crítico — los árboles manejan categóricas razonable. Pero target encoding ayuda con cardinalidad alta.

Referencias

- Micci-Barreca (2001), A Preprocessing Scheme for High-Cardinality Categorical Attributes.

- Van Buuren (2018), Flexible Imputation of Missing Data (libro gratuito).
- category_encoders docs.
- sklearn IterativeImputer.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 056 — Clase 056 — Selección y entrenamiento de modelo

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 2 § Select and Train a Model.*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno entrene varios modelos baseline sobre el pipeline ya preparado (California Housing), los compare con cross-validation en vez de un único split, identifique sub/overfitting con learning curves, y elija el candidato más prometedor para pasar a fine-tuning — sin malgastar tiempo afinando un modelo que no tiene techo.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Entrenar baselines (LinearRegression, DecisionTreeRegressor, RandomForestRegressor) sobre el X_prepared del pipeline de la clase anterior.
2. Evaluar con cross_val_score usando K-Fold y scoring='neg_root_mean_squared_error' en vez de un solo train/test.
3. Leer learning curves para diagnosticar bias vs varianza (underfitting vs overfitting).
4. Comparar modelos con media ± desvío de los folds y decidir cuál merece HPO.
5. Persistir el modelo elegido con joblib.dump(...) para retomarlo en la próxima clase.

Temas

#	Tema	Por qué importa
1	Entrenar baseline LinearRegression y medir	Punto de comparación honesto. Si el lineal
2	DecisionTreeRegressor con RMSE = 0 en tra	Caso canónico de overfitting; te enseña a
3	cross_val_score(..., cv=10, scoring='neg_r	Estimación robusta de error de generalizac
4	RandomForestRegressor y comparación con	Baseline fuerte en tabular; suele ganar an
5	learning_curve — train vs validation score	Diagnóstico visual de bias/varianza.
6	validation_curve — score vs un hiperparáme	Antesala del grid search de la clase sigui
7	Decidir cuándo pasar a HPO vs cuándo segui	Criterio de corte; evita el agujero del tu

Definiciones y características

Baseline model

: Modelo simple (lineal, árbol corto, dummy) contra el cual se compara cualquier modelo más complejo. Si un

Random Forest no le gana al LinearRegression por margen claro, el feature engineering está fallando — no el modelo.

cross_val_score(estimator, X, y, cv, scoring)

: Entrena cv veces sobre folds disjuntos y devuelve un array de scores. En sklearn, scoring siempre es "más alto es mejor" — por eso para RMSE se usa neg_root_mean_squared_error (negado) y después se invierte el signo.

scoring

: String que selecciona la métrica. Para regresión: 'neg_root_mean_squared_error', 'neg_mean_absolute_error', 'r2'. Para clasificación: 'accuracy', 'f1', 'roc_auc'. La lista completa está en Scoring parameter.

Learning curve

: Gráfico de score (train y validation) en función del tamaño del training set. Diagnostica: brecha grande train ↔ val = overfitting (varianza alta); ambas curvas bajas y juntas = underfitting (bias alto); ambas convergen alto = modelo OK.

Validation curve

: Gráfico de score (train y validation) en función de un hiperparámetro (max_depth, n_estimators, etc.). Te muestra a ojo el rango interesante antes de tirar un grid search.

Model card

: Documento corto (markdown, una página) que registra: dataset, features, modelo, métricas en CV, hiperparámetros, fecha, supuestos y limitaciones. Práctica de Google/Hugging Face. Te salva cuando volvés al proyecto 3 meses después.

joblib.dump / joblib.load

: Serialización optimizada para objetos sklearn (matrices NumPy grandes). Preferido sobre pickle puro. Convención: modelo.pkl o modelo.joblib.

Dataset / recursos

Seguimos con California Housing y el X_prepared que sale del pipeline construido en las clases 049-050. Si arrancás esta clase fresca, el notebook regenera el pipeline desde fetch_california_housing() para que sea autocontenido.

Ejercicios

1. Baseline lineal. Entrená LinearRegression() sobre X_prepared, predecí sobre los primeros 5 ejemplos, compará con los y reales. Calculá RMSE sobre todo el train. Esperá algo en el orden de ~68k USD.

2. Árbol que memoriza. Entrená DecisionTreeRegressor(random_state=42) sin restringir profundidad. Calculá RMSE sobre train. Vas a obtener 0 (o casi). Discutí en una celda markdown por qué eso no significa que el modelo sea bueno.

3. Cross-validation honesto. Corré cross_val_score(tree, X_prepared, y, scoring='neg_root_mean_squared_error', cv=10). Reportá media y desvío del RMSE (recordá negar el signo). Compará con el lineal evaluado con el mismo cv=10.

4. Random Forest. Entrená RandomForestRegressor(n_estimators=100, random_state=42) y evaluá con CV de 10 folds. Esperá que la media baje a ~50k USD. Hacé una tabla markdown con los 3 modelos: media ± desvío.

5. Learning curve. Usá sklearn.model_selection.learning_curve sobre el Random Forest con train_sizes=np.linspace(0.1, 1.0, 5). Plotteá train_score y val_score vs tamaño. Diagnosticá: ¿alta varianza, alto bias, o convergencia?

Homework verificable

Notebook que: (a) entrene los 3 baselines sobre X_prepared; (b) corra CV de 10 folds para cada uno y guarde los arrays de scores; (c) genere una tabla comparativa con RMSE media, RMSE desvío, tiempo de fit; (d) elija el modelo más prometedor con justificación de 3 líneas en markdown; (e) plottee la learning curve del elegido; (f) serialice el modelo entrenado en models/modelo_baseline.pkl con joblib.dump.

Criterio de aceptación: El Random Forest aparece con RMSE-CV menor al lineal y al árbol pelado. El archivo .pkl se puede recargar con joblib.load y predice sin errores sobre los primeros 5 ejemplos de X_prepared.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Reporto el RMSE como un número enorme posi	Olvidaste que sklearn devuelve scores nega
Decision tree con RMSE = 0 en train y lo d	Estás midiendo en el mismo set en que entr
cross_val_score tarda eternidad con Random	Cada fold reentrena el bosque entero. Fix:
Learning curve plana en train y val ambas	Underfitting. Fix: no es problema de datos
Hago cross_val_score sobre el X crudo (sin	Te salteaste el preprocessing. Fix: pasale

Preguntas frecuentes

¿Por qué CV de 10 y no un train/test split?

Un solo split te da un solo número con varianza alta — podés tener mala/buena suerte con esa partición. CV te da 10 números: media + desvío. Si el desvío entre folds es grande, el modelo es inestable y un score puntual miente.

¿neg_root_mean_squared_error o neg_mean_squared_error?

neg_root_mean_squared_error está disponible desde sklearn 0.22 y te devuelve directo el RMSE negado. Si tu versión es vieja, usá neg_mean_squared_error y aplicá np.sqrt(-scores) a mano.

¿Cuándo dejo de probar baselines y paso a HPO?

Cuando el mejor baseline ya le saca margen claro al segundo, y la learning curve muestra que más datos no van a mover la aguja. Si seguís en bias alto, antes de HPO sumá features o subí la familia de modelo.

¿Sirve mirar el RMSE de train?

Sí, pero solo como diagnóstico, no como métrica de selección. Train bajo + val alto = overfitting. Train alto + val alto = underfitting. La métrica que ranquea modelos es siempre la de validation (o CV).

¿Cuál es la diferencia entre learning_curve y validation_curve?

learning_curve varía el tamaño del training set (¿necesito más datos?). validation_curve varía un hiperparámetro con N fijo (¿qué max_depth conviene?). Las dos comparten el formato train-vs-val pero responden preguntas distintas.

Referencias

- Géron, cap. 2 § Select and Train a Model + § Better Evaluation Using Cross-Validation.
- sklearn cross_val_score

- sklearn learning_curve
- sklearn Scoring parameter
- Model Cards for Model Reporting (Mitchell et al., 2019)

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 057 — Clase 057 — Fine-tuning: grid search y randomized search

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 2. Duración estimada: 70 min.*

Objetivo

Que el alumno deje de tunear hiperparámetros "a ojo" y use búsquedas sistemáticas con validación cruzada — GridSearchCV cuando el espacio es chico y discreto, RandomizedSearchCV cuando es grande o continuo — integradas dentro de un Pipeline para evitar data leakage del preprocesamiento.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Distinguir parámetros entrenados (pesos del modelo) de hiperparámetros (los que vos fijás antes del fit).
2. Configurar GridSearchCV con param_grid, cv, scoring, n_jobs=-1 y leer best_params_ / best_estimator_ / cv_results_.
3. Usar RandomizedSearchCV con param_distributions (scipy.stats: randint, uniform, loguniform) y n_iter para presupuestar trials.
4. Integrar HPO dentro de un Pipeline usando claves del tipo 'step__hparam' para tunear preprocesamiento + modelo juntos.
5. Decidir grid vs random vs bayesiano según tamaño del espacio, costo por fit y continuidad de los hiperparámetros.

Temas

#	Tema	Por qué importa
1	Hiperparámetros vs parámetros	El error conceptual #1 al arrancar con skl
2	param_grid y scoring	Definís el espacio y la métrica que querés
3	cv y n_jobs=-1	CV honesto + paralelismo gratis sobre todo
4	GridSearchCV	Producto cartesiano completo. Garantiza en
5	RandomizedSearchCV con distribuciones (rar	Muestrea n_iter puntos; gana cuando pocos
6	Pipelines + HPO con sintaxis step__hparam	Evita leakage del scaler/imputer al hacer
7	Inspeccionar cv_results_	DataFrame con todos los trials; sirve para

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s)

con patrón completo, ejercicios y homework:

- Clase 052a — Optuna y HPO bayesiano dedicado

Definiciones y características

Hiperparámetro

: Parámetro que vos fijás antes del entrenamiento y que el fit no aprende. Ej.: max_depth, C, n_estimators. Distinto de los parámetros aprendidos (pesos, splits del árbol).

GridSearchCV(estimator, param_grid, cv, scoring, n_jobs)

: Evalúa el producto cartesiano de param_grid con CV. Total de fits = ∏ |valores| × cv. Determinista y exhaustivo dentro del grid.

RandomizedSearchCV(estimator, param_distributions, n_iter, cv, scoring)

: Muestrea n_iter combinaciones de las distribuciones. Acepta tanto listas (como grid) como distribuciones scipy (randint, uniform, loguniform). Gana cuando algunos hparams son irrelevantes.

param_distributions

: Dict {'hparam': distribución_o_lista}. Para escalas tipo learning rate o C de SVM, usá loguniform(1e-4, 1e0) — random uniforme en log-space, no lineal.

refit=True (default)

: Tras encontrar la mejor combinación, re-entrena el estimador sobre todo el train set con esos hparams. Por eso best_estimator_ viene listo para predecir.

best_params_

: Dict con la combinación ganadora.

best_estimator_

: El modelo ya re-entrenado en train completo. Es el que usás para .predict() en test.

scoring

: Métrica a maximizar. Strings como 'accuracy', 'f1', 'neg_root_mean_squared_error' (negados porque sklearn maximiza siempre). También callables custom con make_scorer.

cv_results_

: Dict-of-arrays con todos los trials: parámetros, mean/std de score por fold, tiempos. Lo cargás en un DataFrame para decidir si refinar.

Dataset / recursos

fetch_california_housing() de sklearn (mismo que viene usando Géron en cap. 2). Para los ejercicios con clasificación, load_breast_cancer().

Ejercicios

1. GridSearchCV sobre RandomForestRegressor. California Housing. param_grid con n_estimators {50, 100, 200} y max_features {4, 6, 8}. cv=5, scoring='neg_root_mean_squared_error', n_jobs=-1. Reportá best_params_ y RMSE en test.

2. RandomizedSearchCV con distribuciones. Mismo dataset, ahora con n_estimators=randint(50, 500), max_features=randint(2, 8), min_samples_leaf=randint(1, 20). n_iter=30. Compará tiempo y score vs el grid del ej. 1.

3. Pipeline + HPO. Construí Pipeline([('scaler', StandardScaler()), ('svr', SVR())]). Tuneá svr__C con loguniform(1e-1, 1e3) y svr__gamma con loguniform(1e-4, 1e-1). Mostrá por qué meter el StandardScaler afuera del CV sería data leakage.

4. Inspección de cv_results_. Cargá search.cv_results_ en un DataFrame, ordenalo por mean_test_score y plotteá score vs n_estimators. ¿Hay meseta? ¿Vale subir más?

5. Optuna sobre el mismo problema. Repetí el ej. 2 pero con Optuna (n_trials=30, TPESampler, MedianPruner). Compará score final y tiempo con RandomizedSearchCV. Mostrá study.best_params y un plot de optuna.visualization.plot_optimization_history.

Homework verificable

Notebook que: (a) baseline con RandomForestRegressor sin tunear; (b) GridSearchCV con grid chico (≤27 combos); (c) RandomizedSearchCV con n_iter=30 y distribuciones log; (d) Optuna con n_trials=30 y MedianPruner; (e) tabla comparativa con RMSE en test, tiempo de búsqueda y mejor combinación de cada método.

Criterio de aceptación: los tres métodos (grid/random/Optuna) mejoran sobre el baseline. La tabla final tiene RMSE_test, segundos y best_params. El Pipeline se usa en todos para evitar leakage del scaler.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ValueError: Invalid parameter 'C' for esti	Estás pasando 'C' pero dentro de un Pipeli
GridSearchCV tarda eternidades	Producto cartesiano explotó. Fix: pasá a R
RMSE de CV mucho mejor que en test	Data leakage: aplicaste StandardScaler.fit
ImportError: cannot import name 'HalvingGr	Sigue siendo experimental en sklearn. Fix:
scoring='rmse' tira ValueError: 'rmse' is	sklearn no tiene 'rmse' directo; maximiza

Preguntas frecuentes

¿GridSearch, RandomSearch u Optuna?

Regla de bolsillo: espacio chico y discreto (≤50 combos) → Grid, te asegura el óptimo del grid. Espacio mediano o con hparams continuos → Random con loguniform para los que viven en escala log. Cada fit cuesta caro (XGBoost grande, red neuronal) o el espacio es enorme → Optuna con TPE + MedianPruner. Para producción seria hoy, Optuna es default.

¿Cuántos n_iter en RandomizedSearchCV?

Bergstra & Bengio (2012) mostraron que 60 trials random alcanzan el top-5% del espacio con probabilidad >95% si pocos hparams dominan. Empezá con 30-60 y subí si cv_results_ muestra que el top-N todavía mejora.

¿n_jobs=-1 en el search o en el modelo?

Si lo ponés en los dos, se pelean por los cores. Recomendación: n_jobs=-1 en el search (paraleliza folds × candidatos) y n_jobs=1 dentro del estimador. Para modelos que ya paralelizan internamente y son rápidos (RandomForest chico), a veces es al revés — medilo.

¿cv=5 o cv=10?

cv=5 es el sweet spot por costo/varianza. cv=10 baja un poco la varianza pero duplica tiempo. Para datasets chicos (≤1k filas), cv=10 o incluso LOO. Para datasets grandes, cv=3 ya alcanza.

¿Tuneo sobre train y evalúo sobre test, o necesito un set de validación aparte?

Con CV adentro de GridSearchCV ya tenés validación honesta sobre folds del train. Test set se toca una sola vez al final, con el best_estimator_ ya elegido. Si vas a hacer muchas iteraciones de exploración manual encima, dejá un set de validación separado para no contaminar test.

Referencias

- Géron, cap. 2 § Fine-Tune Your Model.
- sklearn — GridSearchCV
- sklearn — RandomizedSearchCV
- sklearn — Successive Halving (experimental)
- Optuna docs — TPE, pruners, dashboard.
- Bergstra & Bengio (2012), Random Search for Hyper-Parameter Optimization, JMLR.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 058 — Clase 058 — Optuna y HPO bayesiano dedicado

> *Parte: 1 — Machine Learning Clásico · Fuente: Akiba et al. (2019) + Optuna docs. Duración estimada: 80 min.*

Objetivo

Profundizar en Optuna —el framework de hyperparameter optimization estándar industrial 2026— aplicado a ML clásico (sklearn, XGBoost, LightGBM, CatBoost). Pasar de Grid/Random Search (clase 052) a TPE (Tree-structured Parzen Estimator) + Hyperband Pruner + persistencia con SQLite. Aprender a interpretar plot_optimization_history, plot_param_importances y plot_slice para entender qué hiperparámetros mueven la aguja.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Definir un objective(trial) con suggest_int, suggest_float, suggest_categorical, suggest_float('lr', 1e-5, 1e-1, log=True).
- Aplicar TPE (default) vs CmaEs vs NSGAIISampler (multi-objective).
- Aplicar pruners (MedianPruner, HyperbandPruner) para cortar trials malos temprano.
- Persistir el study con storage='sqlite:///study.db' y resumir trials.
- Visualizar e interpretar los 5 plots de Optuna.
- Comparar costo total: Grid Search (1000 trials) vs Optuna (100 trials) llegan a mismo accuracy.

Temas

- TPE: modela $P(x \mid y < \gamma)$ y $P(x \mid y \geq \gamma)$ con KDE; samplea de la primera.
- Pruning: callback que reporta progreso intermedio; si va mal vs históricos, kill.
- Multi-objective: optimizar accuracy AND latencia.
- Distributed: varios workers contra el mismo SQLite/PostgreSQL.

- Integration con sklearn, XGBoost, LightGBM, CatBoost.

Definiciones y características

- Trial: una corrida del objective.
- Study: contenedor de trials; se persiste opcional.
- TPE: Tree-structured Parzen Estimator. Modelo bayesiano con KDE bi-modal.
- HyperbandPruner: combina successive halving con varios brackets.
- storage: backend de persistencia. SQLite default; Postgres para distributed.
- plot_param_importances: importancia fANOVA — cuánto contribuye cada hiperparámetro a la varianza del objetivo.

Dataset / recursos

- sklearn.datasets.fetch_california_housing (regresión) o fetch_openml('credit-g') (clasificación).
- Librerías: optuna, optuna-integration, scikit-learn, xgboost, lightgbm.

Ejercicios

1. Objective básico: tunear LogisticRegression(C, penalty) y RandomForest(n_estimators, max_depth) con TPE. 50 trials.
2. Search space compuesto: hiperparámetros condicionales (e.g., solver=liblinear solo permite ciertos penalty). Optuna lo maneja con if.
3. Pruning en XGBoost: usar XGBoostPruningCallback que reporta validación por boosting round → mata trials malos.
4. Persistencia: optuna.create_study(study_name='exp1', storage='sqlite:///opt.db', load_if_exists=True). Re-correr y agregar trials.
5. Multi-objective: maximizar accuracy AND minimizar inference time; obtener Pareto front con optuna.create_study(directions=['maximize', 'minimize']).

Homework verificable

HPO con Optuna sobre XGBoost en credit-g:

1. Espacio: n_estimators, max_depth, lr, subsample, colsample_bytree, reg_alpha, reg_lambda.
2. 100 trials con TPE + Hyperband pruning.
3. Reportar best_params, best_value, plot history + importances.
4. Comparar contra RandomizedSearchCV(50 trials).

Criterio de aceptación: Optuna ≥ RandomizedSearch en AUC; plot_param_importances identifica lr y/o max_depth como las más importantes.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
LR muestreado uniforme	Fix: suggest_float('lr', 1e-5, 1e-1, log=T
TPE no parece "inteligente" en < 20 trials	El modelo interno necesita warmup. Fix: ≥
study.optimize(n_jobs=4) se cuelga	Race conditions con SQLite. Fix: usar Post
Pruner muy agresivo mata trials buenos	Fix: ajustar MedianPruner(n_startup_trials
Olvido reportar valor intermedio para prun	El pruner no funciona sin trial.report(val

Preguntas frecuentes

¿TPE o CmaEs?

TPE para search spaces mezclados (cat + cont). CmaEs para puramente continuo, con poca cardinalidad — converge más rápido.

¿Cuántos trials?

Para ML clásico, 50-200 alcanza. Para DL caro (cada trial 1 h), 20-50 con pruning agresivo.

¿Distributed HPO?

Postgres storage + varios workers (Python processes en distintas máquinas). Optuna lo soporta nativo.

¿Pruning siempre?

Solo si el objective expone progreso intermedio (cada época en NN, cada boosting round en XGBoost). Para fit().score() directo, no aplica.

¿Visualizaciones para reportar al cliente?

plot_param_importances y plot_slice son las más comunicables. Muestran "qué cambió y cuánto".

Referencias

- Akiba et al. (2019), Optuna, KDD.
- Optuna docs — tutorials y examples.
- Bergstra, Bardenet, Bengio & Kégl (2011), Algorithms for Hyper-Parameter Optimization, NeurIPS — paper original TPE.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 059 — Clase 059 — Launch, monitoreo y mantenimiento de modelos

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 2 § Launch, Monitor, and Maintain Your System.*

> *Duración estimada: 60 min.*

Objetivo

Que el alumno entienda que entrenar el modelo es la mitad del trabajo: el resto es ponerlo en producción de forma segura, monitorearlo para detectar degradación (data drift, model drift) y mantenerlo vivo con un ciclo de retraining. Además, documentar el modelo con una Model Card para que terceros (compliance, negocio, usuarios) sepan qué hace, dónde falla y qué no hay que hacerle.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Diseñar un pipeline de deploy mínimo (serialización con joblib, servicio detrás de una API, versionado del artefacto).
2. Distinguir data drift de model drift y elegir métricas para cada uno (PSI, KS, accuracy en holdout

móvil).

3. Definir un retraining trigger (calendario fijo vs. trigger por drift vs. trigger por caída de KPI de negocio).
4. Comparar estrategias de release (canary, shadow deploy, A/B test) y elegir según riesgo.
5. Redactar una Model Card con secciones mínimas (uso previsto, métricas por subgrupo, limitaciones).

Temas

#	Tema	Por qué importa
1	Pipeline de deploy: joblib.dump, contenedo	El modelo serializado es el artefacto prod
2	Data drift (inputs cambian) vs. model drif	Se monitorean distinto; confundirlos te ll
3	Métricas de drift: PSI, KS-test, distancia	Cuantifican el cambio en distribución ante
4	Retraining: calendario, trigger por drift,	Cuándo reentrenar sin gastar de más ni que
5	Estrategias de release: shadow, canary, A/	Bajan el riesgo de un modelo malo en produ
6	Alertas y observabilidad	Logueo de inputs/outputs, dashboards, on-c
7	Model Cards y Datasheets	Documentación responsable del modelo y de
8	Governance y rollback	Versionar modelos como código; poder volve

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 053a — Model Cards y Responsible ML

Definiciones y características

Deployment

: Proceso de exponer un modelo entrenado como servicio consumible (REST, batch, edge). Incluye serialización del artefacto (joblib, pickle, ONNX), versionado y contenedorización.

Model drift (concept drift)

: La relación entre X e y cambia con el tiempo. Ejemplo: hábitos de compra post-pandemia. Se detecta midiendo accuracy/AUC sobre una ventana móvil con labels reales (cuando llegan).

Data drift (covariate shift)

: La distribución de los inputs X cambia, aunque la relación X→y siga estable. Se detecta sin necesidad de labels — comparás la distribución de cada feature en producción vs. la del set de entrenamiento (PSI, KS).

Retraining trigger

: Regla que decide cuándo reentrenar. Tres familias: (a) calendario fijo (cada N días), (b) drift detectado (PSI > 0.25), (c) caída de KPI de negocio (conversion baja > X%).

A/B test

: Dos versiones del modelo sirven tráfico en paralelo (e.g. 50/50). Se compara KPI de negocio con significancia estadística. Requiere volumen.

Shadow deploy

: El modelo nuevo recibe los mismos requests que el viejo pero sus predicciones no se devuelven al usuario — se loguean para comparar. Cero riesgo, ideal para validar en datos reales antes del switch.

Model card

: Documento estandarizado que describe modelo, uso previsto, métricas (overall + por subgrupo), datos,

limitaciones y consideraciones éticas. Formalizado por Mitchell et al. (2019).

Governance

: Conjunto de políticas: quién aprueba un deploy, cómo se versiona el modelo, cómo se hace rollback, qué se loguea para auditoría. Equivalente del "code review" para modelos.

Dataset / recursos

Sintético: dos snapshots de un dataset tabular (mes 1 y mes 6) para simular drift. Plantilla Markdown de Model Card en model_card_template.md.

Ejercicios

1. Serializar y cargar. Entrená un RandomForestClassifier sobre Titanic. Guardalo con joblib.dump. Cargalo en otro notebook y verificá que predice idéntico.

2. Detectar data drift con PSI. Calculá Population Stability Index entre dos snapshots de la feature edad. Interpretá: PSI < 0.1 estable, 0.1-0.25 leve, > 0.25 drift significativo.

3. KS-test para drift. Aplicá scipy.stats.ks_2samp a la feature monto entre snapshot1 y snapshot2. ¿p-valor < 0.05?

4. Simular shadow deploy. Tenés modelo A (viejo) y B (nuevo). Pasá 1000 requests por ambos, logueá las predicciones y reportá tasa de desacuerdo.

5. Redactar una Model Card. Tomá tu mejor modelo de la Parte 1 hasta ahora y completá una model card con las 7 secciones del complemento. Incluí al menos una métrica desagregada por subgrupo.

Homework verificable

Notebook que: (a) entrene un clasificador, (b) lo serialice con joblib, (c) simule un mes de tráfico productivo con una feature drifteada, (d) calcule PSI por feature, (e) dispare una alerta si PSI > 0.25, (f) entregue un archivo MODEL_CARD.md completo en la carpeta del homework.

Criterio de aceptación: El notebook detecta el drift inyectado artificialmente. La Model Card tiene las 7 secciones mínimas con valores reales (no placeholders), incluye al menos una métrica por subgrupo y declara explícitamente un out-of-scope use.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
El modelo cargado con joblib.load predice	Cambió la versión de sklearn entre dump y
Reentreno cada semana "por las dudas" y el	Retraining sin trigger real introduce ruid
PSI da siempre 0 o inf	División por cero cuando un bin queda vací
El A/B test "no da significativo" después	Volumen insuficiente. Fix: hacé power anal
Model card con métrica única "accuracy = 0	Oculta sesgos por subgrupo y no dice para

Preguntas frecuentes

¿Cada cuánto hay que reentrenar?

No hay número mágico. Si el dominio es estable (físico, industrial), meses o años. Si es comportamiento humano online (e-commerce, fraude), semanas o incluso días. Lo correcto es dejar que el trigger lo decida (PSI, caída de KPI), no el calendario.

¿Diferencia práctica entre shadow deploy y canary?

Shadow: el modelo nuevo predice pero sus predicciones no se devuelven al usuario — cero riesgo, solo comparás. Canary: el modelo nuevo sí sirve tráfico real, pero a un % chico (1-5%) — bajo riesgo pero no nulo. Shadow para validar, canary para liberar gradualmente.

¿Model card o documentación técnica clásica?

Las dos. La doc técnica (README, API reference) es para desarrolladores. La model card es para stakeholders no técnicos: producto, legal, compliance, auditoría. Si tu modelo cae bajo EU AI Act, la model card no es opcional.

¿pickle o joblib o ONNX?

joblib para sklearn (más eficiente con arrays NumPy grandes que pickle puro). pickle para objetos Python genéricos. ONNX cuando necesitás portabilidad cross-language (servir un modelo Python desde un backend en C# o Java).

¿PSI o KS-test?

PSI es más interpretable para negocio (umbrales 0.1 / 0.25 son estándar de la industria de scoring crediticio) y opera sobre features categóricas/binneadas. KS-test es estadísticamente más riguroso para features continuas y devuelve p-valor. En la práctica, equipos serios reportan ambas.

Referencias

- Géron, cap. 2 § Launch, Monitor, and Maintain Your System.
- Mitchell et al. (2019). Model Cards for Model Reporting. FAT* 2019.
- Gebru et al. (2018). Datasheets for Datasets.
- HuggingFace Model Cards guide.
- Google model-card-toolkit.
- EU AI Act — documentación técnica (Anexo IV).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 060 — Clase 060 — Model Cards y Responsible ML

> *Parte: 1 — Machine Learning Clásico · Fuente: Mitchell et al. (2018) Model Cards for Model Reporting + EU AI Act + NIST AI RMF. Duración estimada: 70 min.*

Objetivo

Aprender a documentar modelos para producción y auditoría: el Model Card (Mitchell et al. 2018, adoptado por Google y luego por la industria) — ficha estandarizada con: propósito, métricas, limitaciones, distribución de datos, riesgos. Conocer el EU AI Act (en vigor 2025-2026), NIST AI RMF, y las plantillas modernas (HuggingFace model cards, Datasheets for Datasets).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Escribir un Model Card completo con las 9 secciones de Mitchell et al.

- Distinguir un Model Card (sobre el modelo) de un Datasheet (sobre el dataset, Gebru et al. 2018).
- Reportar métricas por subgrupo (no solo global) — clave en fairness.
- Reconocer los 4 tiers de riesgo del EU AI Act (prohibido, alto, transparencia, mínimo).
- Aplicar el NIST AI RMF (Map, Measure, Manage, Govern) en un proyecto real.

Temas

- Secciones de un Model Card: Model Details, Intended Use, Factors, Metrics, Evaluation Data, Training Data, Quant Analyses, Ethical Considerations, Caveats.
- Métricas por subgrupo (sexo, edad, raza, geografía).
- HuggingFace Model Card auto-generation.
- EU AI Act: clasificación de riesgo, obligaciones por tier.
- NIST AI RMF: framework de gestión.
- ISO/IEC 42001 — sistema de gestión de IA.

Definiciones y características

- Model Card: ficha estructurada que documenta un modelo para terceros.
- Datasheet for Datasets: equivalente para datasets — origen, sesgos, demográficos.
- EU AI Act: regulación europea (vigor 2024-2026). Multas hasta 35M€ o 7 % revenue.
- High-risk AI: sistemas en empleo, crédito, educación, justicia, infraestructura. Requieren conformity assessment.
- GPAI (General-Purpose AI): LLMs y similares — obligaciones de transparencia adicionales.
- NIST AI RMF: framework voluntario US, ampliamente adoptado.

Dataset / recursos

- Modelo del proyecto end-to-end (clase 050).
- Plantilla HuggingFace: <https://huggingface.co/docs/hub/model-cards>.
- Librerías: model-card-toolkit (Google).

Ejercicios

1. Model Card básico: para un Random Forest entrenado en California Housing, llenar las 9 secciones. Salvar como MODEL_CARD.md junto al modelo.
2. Subgroup metrics: para un clasificador de credit-g, reportar accuracy y FPR por sex y age_group. Identificar disparidades.
3. Risk classification (EU AI Act): para 5 use cases (recomendación de películas, score crediticio, recurso humano selection, marketing email, detector de spam), clasificar el tier.
4. HuggingFace Card: usar el template de HF; subirla a un repo público si tenés modelo en Hub.
5. NIST RMF: para un proyecto propio, llenar las 4 categorías (Map: contexto, Measure: métricas, Manage: mitigaciones, Govern: ownership).

Homework verificable

Model Card completo para el proyecto end-to-end (clase 050):

1. Las 9 secciones de Mitchell et al. llenadas con datos reales.
2. Métricas por al menos 2 subgrupos demográficos.
3. Sección "Ethical Considerations" con ≥ 3 riesgos identificados y mitigaciones.
4. Sección "Caveats and Recommendations" con limitaciones de validez.

Criterio de aceptación: un revisor externo puede entender propósito, performance, riesgos y cómo usarlo apropiadamente sin acceso al código.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Model Card que solo reporta accuracy globa	Esconde disparidades de subgrupo. Fix: tab
"Intended use" vago ("for predictions")	Inútil. Fix: especificar exactamente qué d
Omitir "Out-of-scope use cases"	No avisás al usuario. Fix: explícito ("NO
Métricas en test, no en producción real	Distribution shift no documentado. Fix: mo
Asumir EU AI Act no aplica	Aplica si el modelo se usa en UE, independ

Preguntas frecuentes

¿Model Card obligatorio?

Por ley: depende de jurisdicción y caso de uso (EU AI Act lo requiere para high-risk). Como buena práctica: siempre.

¿Qué pasa si mi modelo es high-risk EU AI Act?

Obligaciones: conformity assessment, registro en base UE, documentación técnica, supervisión humana, robustez. Multas hasta 7 % revenue.

¿Model Cards en producción industrial?

Sí. Google, Meta, Microsoft, OpenAI, Anthropic — todas publican Model Cards. HuggingFace lo requiere para modelos en Hub.

¿Datasheet for Datasets equivalente?

Sí, Gebru et al. (2018). Documenta origen, demográficos, sesgos del dataset. HuggingFace tiene Dataset Cards.

¿Y ISO 42001?

Sistema de gestión de IA (publicado dic 2023). Certificación auditable. Análogo a ISO 27001 para seguridad.

Referencias

- Mitchell, M., et al. (2018), Model Cards for Model Reporting, FAT* 2019.
- Gebru, T., et al. (2018), Datasheets for Datasets, CACM 2021.
- EU AI Act texto completo: <https://artificialintelligenceact.eu/>.
- NIST AI Risk Management Framework: <https://www.nist.gov/itl/ai-risk-management-framework>.
- HuggingFace Model Cards: <https://huggingface.co/docs/hub/model-cards>.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 061 — Clase 061 — CRISP-DM como framework metodológico

> *Parte: 1 — Machine Learning Clásico · Fuente: CRISP-DM 1.0 spec + Géron cap. 2. Duración estimada: 50 min.*

Objetivo

Que el alumno entienda CRISP-DM (Cross-Industry Standard Process for Data Mining) como esqueleto metodológico para todo proyecto de ML/DS — sus 6 fases, su naturaleza iterativa (no en cascada), y cuándo conviene complementarlo o reemplazarlo por TDSP de Microsoft o por el "ML lifecycle moderno" con MLOps. La idea es dejar de improvisar el orden del trabajo y tener un mapa al que volver cuando un proyecto se trabe.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Enumerar y explicar las 6 fases de CRISP-DM y los entregables típicos de cada una.
2. Identificar la fase actual de un proyecto real y la próxima transición esperada.
3. Definir business success criteria antes de tocar datos, distinguiéndolos de métricas técnicas (accuracy, RMSE).
4. Reconocer iteraciones legítimas (volver de Evaluation a Business Understanding) vs. retrabajo por mala planificación.
5. Comparar CRISP-DM con TDSP y el ML lifecycle moderno y elegir según contexto (PoC, equipo chico, producción seria, MLOps).

Temas

#	Tema	Por qué importa
1	Business Understanding	Sin objetivo de negocio claro, el modelo n
2	Data Understanding	EDA, calidad, volumen, disponibilidad — de
3	Data Preparation	El 60–80% del tiempo real; limpieza, featu
4	Modeling	Selección de algoritmos, tuning, validació
5	Evaluation	Compara con business success criteria, no
6	Deployment	Puesta en producción, monitoreo, retrainin
7	Iteración + comparación TDSP / ML lifecycl	CRISP-DM no es cascada; existen alternativ

Definiciones y características

Business Understanding

: Fase 1. Traduce el problema de negocio a un problema de DS. Entregables: objetivos de negocio, criterios de éxito del negocio (ej. "reducir churn 5pp en 6 meses"), inventario de recursos, plan de proyecto. Sin esto, todo lo demás es ejercicio académico.

Data Understanding

: Fase 2. Recolectar datos, describirlos, explorarlos (EDA), verificar calidad. Entregable: reporte de calidad y primeras hipótesis. Si los datos no alcanzan, se vuelve a Business Understanding a renegociar scope.

Data Preparation

: Fase 3. Limpieza, integración, formateo, feature engineering, splits (train/val/test). Suele consumir el 60–80% del proyecto. Entregable: dataset final modelable + script reproducible.

Modeling

: Fase 4. Selección de técnicas, diseño de test, entrenamiento, tuning. Entregable: modelos candidatos con métricas técnicas. Si el modelo requiere otro formato de datos, se vuelve a Data Preparation.

Evaluation

: Fase 5. ¿El modelo cumple los business success criteria definidos en fase 1? No solo accuracy — costo del falso positivo, latencia, fairness, interpretabilidad. Entregable: decisión go/no-go.

Deployment

: Fase 6. Producción, monitoreo de drift, plan de retraining, documentación, handoff. Termina con un sistema funcionando, no con un notebook.

Iteración

: CRISP-DM no es cascada — las flechas van en ambos sentidos entre fases adyacentes, y la flecha externa cierra el ciclo (Deployment → nueva ronda de Business Understanding). Iterar es la norma, no la excepción.

Business success criteria

: Métricas en términos del negocio (dinero, conversión, tiempo, NPS) acordadas con stakeholders antes de modelar. Distintas de métricas técnicas (accuracy, F1, RMSE). Sin ellas, no hay forma objetiva de cerrar la fase de Evaluation.

Dataset / recursos

No hay dataset. La clase es conceptual + un caso práctico para aplicar las 6 fases (sugerido: predicción de churn telco, o un caso del alumno).

Ejercicios

1. Identificar la fase. Dadas 6 situaciones de proyecto (ej. "estoy probando XGBoost vs Random Forest", "estoy graficando histogramas para ver outliers"), asigná cada una a su fase CRISP-DM.

2. Business success criteria. Para un caso de detección de fraude bancario, escribí 3 criterios de éxito de negocio (no técnicos) y 3 técnicos. Distinguilos claramente.

3. Aplicar las 6 fases a un caso real. Elegí un problema (churn de Netflix, recomendación de productos, predicción de demanda en una panadería). Redactá un párrafo por cada fase con entregables concretos. Mínimo 1 iteración explícita (ej. "vuelvo de Modeling a Data Preparation porque…").

4. Comparar frameworks. Tabla de 3 columnas (CRISP-DM, TDSP, ML lifecycle moderno con MLOps) y 5 filas (origen, fases, foco, herramientas, cuándo usarlo).

5. Detectar iteración legítima vs retrabajo. Dados 4 escenarios de "estoy volviendo a una fase anterior", clasificalos como iteración esperada o como síntoma de mala planificación en la fase anterior.

Homework verificable

Documento Markdown (1–2 páginas) tomando un proyecto propio (real o sintético) y desarrollando las 6 fases de CRISP-DM con: (a) objetivo de negocio en una frase; (b) 2 business success criteria cuantificables; (c) descripción mínima de datos disponibles; (d) plan de data prep en bullets; (e) 2 algoritmos candidatos justificados; (f) cómo se evaluará contra los criterios de (b); (g) cómo se desplegaría y monitorearía.

Criterio de aceptación: Las 6 fases están presentes, los success criteria son cuantificables (con número y unidad), y la fase de Evaluation mapea explícitamente a los criterios de (b) — no solo a métricas técnicas.

Errores comunes

Síntoma	Causa y cómo arreglar
El modelo tiene gran accuracy pero el nego	Saltaste Business Understanding. Fix: volv
Tratar CRISP-DM como cascada (una fase des	Malentendido del framework. Fix: el spec o
Confundir métricas técnicas con success cr	Reportás F1=0.87 al gerente; el gerente no

Saltar Deployment ("ya tengo el notebook")	El proyecto muere en el laptop del DS. Fix
Hacer EDA infinito sin avanzar a Modeling	Parálisis en Data Understanding. Fix: time

Preguntas frecuentes

¿CRISP-DM o TDSP o ML lifecycle moderno?

Depende. CRISP-DM (1999, IBM/SPSS) es agnóstico de herramientas y sigue siendo el más enseñado — bueno como esqueleto mental y para proyectos chicos/PoC. TDSP (Microsoft, 2016) es más prescriptivo, define roles (data scientist, engineer, PM), templates de carpetas y artefactos Git — bueno para equipos. ML lifecycle moderno (con MLOps: feature stores, model registry, CI/CD de modelos, monitoring de drift) es lo que se usa cuando el modelo está en producción seria. No son excluyentes: CRISP-DM como mapa conceptual + TDSP como estructura de equipo + MLOps como tooling de producción.

¿Tengo que pasar por las 6 fases en orden estricto?

No. Las flechas entre fases adyacentes son bidireccionales y hay un loop externo. Lo que sí es no negociable: no podés saltarte Business Understanding ni Evaluation.

¿Cuánto tiempo lleva cada fase?

Regla aproximada: 10% Business, 10% Data Understanding, 50–70% Data Preparation, 10% Modeling, 10% Evaluation, variable Deployment. Si Modeling te consume el 50%, algo está mal — probablemente saltaste prep.

¿CRISP-DM aplica a deep learning?

Sí, conceptualmente. Las fases son las mismas. Cambia el detalle en Data Preparation (tensores, augmentation), Modeling (arquitecturas, GPU) y Deployment (serving de modelos pesados, ONNX). Pero el esqueleto se sostiene.

¿Y si el negocio no sabe qué quiere?

Ese es exactamente el trabajo de Business Understanding — ayudar al negocio a articularlo. Si después de varias sesiones no hay objetivo claro, el proyecto no está listo para empezar. Documentalo y volvé cuando lo esté.

Referencias

- CRISP-DM 1.0 step-by-step data mining guide (Wirth & Hipp, 2000)
- Géron, cap. 2 End-to-End Machine Learning Project.
- Microsoft Team Data Science Process (TDSP)
- Google ML Lifecycle / MLOps levels

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 062 — Clase 062 — Clasificación binaria con MNIST

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 3. Duración estimada: 60 min.*

Objetivo

Que el alumno arme su primer clasificador binario "de verdad" sobre MNIST (¿este dígito es un 5 o no?), entrenando un SGDClassifier, evaluándolo con cross_val_score sobre StratifiedKFold, y entendiendo por qué la accuracy sola miente cuando las clases están desbalanceadas.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Cargar MNIST con fetch_openml('mnist_784', as_frame=False) y separar train/test respetando el split original (60k/10k).
2. Construir un target binario y_train_5 = (y_train == 5) y entrenar un SGDClassifier sobre él.
3. Predecir y obtener scores con predict() y decision_function(), entendiendo la diferencia entre clase y score continuo.
4. Validar con cross_val_score sobre StratifiedKFold y leer los 3 valores que devuelve.
5. Detectar el accuracy paradox: comparar contra un clasificador trivial "nunca-5" y ver que también saca ~90%.

Temas

#	Tema	Por qué importa
1	fetch_openml('mnist_784')	Dataset estándar de entrada a CV/ML; 70k i
2	Target binario 5 vs no-5	Forma canónica de empezar antes de meterse
3	SGDClassifier(random_state=42)	Lineal, escalable, online; el "hola mundo"
4	cross_val_score + StratifiedKFold	Mantiene la proporción de clases en cada f
5	Accuracy paradox	90% suena bien hasta que ves que Never5Cla
6	predict vs decision_function vs predict_pr	Score continuo es lo que después te deja m

Definiciones y características

MNIST

: Dataset de 70.000 imágenes de dígitos manuscritos (0-9), 28×28 píxeles en escala de grises, aplanadas a vectores de 784 features. Split convencional: 60k train + 10k test, ya barajado. Es el "hello world" de ML — chico, limpio, sin sorpresas.

SGDClassifier

: Clasificador lineal entrenado con Stochastic Gradient Descent. Procesa instancias de a una (o mini-batch), lo que lo hace ideal para datasets grandes y aprendizaje online. Sensible al random_state (fijarlo siempre para reproducibilidad).

decision_function(X)

: Devuelve el score crudo (distancia firmada al hiperplano de decisión), no la clase. Score > 0 → clase positiva, score < 0 → negativa. Es lo que vas a necesitar después para mover el umbral y construir curvas PR/ROC.

Accuracy

: (TP + TN) / total. Métrica obvia pero traicionera: si el 90% de tus instancias son de la clase negativa, predecir "siempre negativo" te da 90% sin haber aprendido nada.

Baseline trivial (dummy classifier)

: Modelo que predice siempre la clase mayoritaria (o aleatorio). Sirve de piso: si tu modelo no le gana al

dummy, no aprendió. En 5-vs-no-5, el dummy "nunca-5" saca ~90.9% (porque solo ~9.1% de los dígitos son 5).

StratifiedKFold

: Variante de K-Fold que conserva la proporción de clases en cada fold. Imprescindible con desbalanceo: con K-Fold común podrías terminar con un fold sin un solo 5.

cross_val_score(clf, X, y, cv=3, scoring='accuracy')

: Entrena y evalúa clf en cv folds y devuelve un array de cv scores. Por default usa StratifiedKFold cuando la tarea es clasificación.

Dataset / recursos

MNIST vía fetch_openml (se cachea localmente en ~/scikit_learn_data/ después de la primera descarga):

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', as_frame=False, parser='auto')
X, y = mnist.data, mnist.target.astype(int)
X_train, X_test = X[:60000], X[60000:]
y_train, y_test = y[:60000], y[60000:]
y_train_5 = (y_train == 5)
y_test_5  = (y_test == 5)
```

Ejercicios

1. Cargar y explorar. Cargá MNIST, imprimí X.shape y y.shape, mostrá un dígito con matplotlib.imshow(X[0].reshape(28, 28), cmap='binary') y verificá que y[0] == 5.

2. Target binario + SGD. Construí y_train_5, entrená SGDClassifier(random_state=42) y predecí sobre X[0]. ¿Devuelve True?

3. decision_function. Sobre la misma instancia, llamá a decision_function([X[0]]) y compará el signo con el resultado de predict.

4. Cross-validation. Corré cross_val_score(sgd, X_train, y_train_5, cv=3, scoring='accuracy'). ¿Qué tres valores te da? ¿Cuál es el promedio?

5. Baseline trampa. Implementá un Never5Classifier (clase con fit que no hace nada y predict que devuelve np.zeros(len(X), dtype=bool)) y corré el mismo cross_val_score. Comparalo con el SGD: ¿la diferencia es la que esperabas?

Homework verificable

Notebook que: (a) carga MNIST y arma el target 5-vs-no-5; (b) entrena SGDClassifier(random_state=42) sobre el train completo; (c) reporta accuracy 3-fold con cross_val_score; (d) reporta accuracy 3-fold del Never5Classifier; (e) en una celda markdown, explica en 2-3 líneas por qué ambos modelos están "cerca" en accuracy y por qué eso no significa que sean equivalentes.

Criterio de aceptación: Accuracy del SGD ≥ 0.95 en los 3 folds, accuracy del dummy ≈ 0.909 en los 3 folds, y la celda markdown menciona explícitamente el desbalanceo (~9% de 5s) como causa del paradox.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
"¡Mi modelo saca 90% de accuracy, está gen	Trampa clásica con clases desbalanceadas.
fetch_openml tarda eternidades o falla	La primera vez descarga ~55MB y a veces el

y viene como strings ('5', '0', ...) y la	Por default OpenML devuelve labels como st
Resultados distintos cada vez que entrenás	SGDClassifier es estocástico. Fix: SGDClas
decision_function no existe / AttributeErr	Algunos clasificadores (como RandomForest)

Preguntas frecuentes

¿Por qué arrancar con clasificación binaria si MNIST es multiclase?

Pedagógico: binario aísla los conceptos de score, umbral, precision/recall sin el ruido de "¿es 3 o 8?". Multiclase viene en la clase 059 (One-vs-Rest, One-vs-One).

¿SGDClassifier con loss='hinge' (default) es lo mismo que SVM?

Es un SVM lineal entrenado con SGD en vez de con el optimizador cuadrático de SVC. Misma frontera teórica, distinta forma de llegar. Para datasets grandes, SGD le gana en tiempo.

¿Por qué cv=3 y no cv=10?

Géron usa 3 porque MNIST es grande (60k×784) y 10 folds tarda. En datasets chicos, 5-10 es lo habitual. La regla: más folds = menos varianza en la estimación pero más cómputo.

¿predict_proba está disponible en SGDClassifier?

Solo si entrenás con loss='log_loss' (regresión logística) o loss='modified_huber'. Con el default 'hinge' no — usás decision_function.

¿Hace falta escalar las features de MNIST?

Para SGDClassifier sí, conviene (los píxeles 0-255 son grandes y SGD es sensible a escala). Géron en el cap. 3 lo omite para simplificar; en la práctica StandardScaler o dividir por 255 ayuda a la convergencia.

Referencias

- Géron, cap. 3 § "Training a Binary Classifier" y "Measuring Accuracy Using Cross-Validation".
- sklearn SGDClassifier
- sklearn fetch_openml
- sklearn StratifiedKFold

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 063 — Clase 063 — Métricas: confusion matrix, precision, recall, F1

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 3. Duración estimada: 70 min.*

Objetivo

Que el alumno deje de mirar accuracy como métrica única y aprenda a leer una confusion matrix, a elegir entre precision, recall, F1 o F-beta según el costo de los errores, y a interpretar classification_report clase por clase. En particular, entender por qué en problemas desbalanceados (fraude, churn, diagnóstico) accuracy

miente y qué hacer al respecto.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Construir e interpretar una confusion matrix con sklearn.metrics.confusion_matrix identificando TP, FP, TN, FN.
2. Calcular y comparar precision, recall, F1 y F-beta a mano y con precision_score, recall_score, f1_score, fbeta_score.
3. Elegir la métrica adecuada según el costo asimétrico de los errores (FP vs FN) del problema.
4. Leer classification_report y diferenciar macro avg vs weighted avg en multiclase.
5. Diagnosticar class imbalance y aplicar class_weight='balanced', SMOTE o threshold tuning según corresponda.

Temas

#	Tema	Por qué importa
1	Confusion matrix (TP/FP/TN/FN)	Base de toda métrica de clasificación.
2	Precision = TP / (TP+FP)	Cuán "puras" son las predicciones positiva
3	Recall = TP / (TP+FN)	Qué fracción de positivos reales capturo.
4	F1 y F-beta	Media armónica; F-beta pondera recall (β>1
5	classification_report y macro/weighted avg	Métricas por clase en multiclase.
6	Accuracy y por qué falla con clases desbal	Paradoja del 99% inútil.
7	Class imbalance: class_weight, SMOTE, thre	Cuándo aplicar cada uno.

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 056a — Class imbalance: SMOTE, ADASYN, class_weight, threshold tuning

Definiciones y características

TP / FP / TN / FN

: True Positive: positivo real predicho positivo. False Positive: negativo real predicho positivo (error tipo I). True Negative: negativo real predicho negativo. False Negative: positivo real predicho negativo (error tipo II). Toda métrica binaria se deriva de estos cuatro.

Confusion matrix

: Tabla 2×2 (o k×k en multiclase) con counts de cada combinación (real, predicho). En sklearn las filas son la clase real y las columnas la predicha: [[TN, FP], [FN, TP]].

Precision = TP / (TP + FP)

: De todo lo que predije positivo, ¿qué fracción acerté? Importa cuando un FP es caro (ej.: bloquear una transacción legítima, marcar email legítimo como spam).

Recall (sensitivity, TPR) = TP / (TP + FN)

: De todos los positivos reales, ¿qué fracción detecté? Importa cuando un FN es caro (ej.: no detectar un tumor, dejar pasar un fraude).

F1 = 2 · (P · R) / (P + R)

: Media armónica de precision y recall. Penaliza fuerte cuando una de las dos es baja. Default razonable cuando no hay preferencia clara entre FP y FN.

F-beta = (1 + β²) · (P · R) / (β² · P + R)

: F1 generalizada. β > 1 pondera más recall (β=2 → recall pesa 4× más que precision). β < 1 pondera más precision (β=0.5). Útil cuando el costo de FP y FN es asimétrico pero ambos importan.

Accuracy = (TP + TN) / (TP + TN + FP + FN)

: Fracción total de aciertos. Engañosa con clases desbalanceadas: con 99% negativos, predecir "todo negativo" da 99% accuracy con cero recall. Reservala para problemas balanceados.

class_weight

: Hiperparámetro de la mayoría de estimadores sklearn. 'balanced' calcula pesos inversamente proporcionales a la frecuencia. También acepta dict {0: 1, 1: 10} para control manual. Modifica la función de pérdida, no el dataset.

Dataset / recursos

- MNIST 5-vs-not-5 (binarizado): clásico de Géron cap. 3. Imbalance ~10% positivo.
- Credit card fraud (Kaggle, opcional): ~0.17% positivo, ideal para ver SMOTE en acción.
- Sintético con make_classification(weights=[0.99, 0.01]) para experimentar sin descargar.

Ejercicios

1. Confusion matrix a mano. Dado y_true = [0,1,1,0,1,1,0,0,1,0] y y_pred = [0,1,0,0,1,1,1,0,1,0]: calculá TP/FP/TN/FN, precision, recall y F1 con lápiz y papel. Verificá con sklearn.metrics.

2. MNIST 5-detector. Entrená SGDClassifier sobre MNIST binarizado (5 vs no-5). Mostrá confusion matrix con ConfusionMatrixDisplay y reportá precision, recall y F1.

3. classification_report multiclase. Sobre MNIST completo (10 clases) con LogisticRegression, imprimí classification_report. Identificá qué dígito tiene peor recall y por qué (mirá la confusion matrix).

4. Class imbalance con class_weight. Generá un dataset con make_classification(n_samples=10000, weights=[0.99, 0.01], random_state=42). Entrená dos LogisticRegression: una sin class_weight y otra con class_weight='balanced'. Compará recall de la clase minoritaria.

5. SMOTE dentro de Pipeline. Mismo dataset que el ejercicio 4. Armá un imblearn.pipeline.Pipeline con SMOTE + LogisticRegression, evaluá con cross_val_score(scoring='f1') y compará contra class_weight='balanced'. Verificá que SMOTE solo se aplica al train fold (leelo en docs).

Homework verificable

Notebook sobre el dataset sintético desbalanceado (~1% positivo) que: (a) entrene baseline LogisticRegression y reporte confusion matrix + classification_report; (b) repita con class_weight='balanced'; (c) arme Pipeline con SMOTE; (d) haga threshold tuning con precision_recall_curve sobre val set maximizando F1; (e) tabla comparativa de F1 y recall de la clase minoritaria para las 4 estrategias (baseline, class_weight, SMOTE, threshold tuning).

Criterio de aceptación: las 4 estrategias documentadas, F1 de la clase minoritaria del mejor enfoque ≥ 2× el del baseline, y SMOTE aplicado solo dentro de Pipeline (chequear en código que no aparece SMOTE().fit_resample(X, y) sobre el dataset completo antes del split).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Accuracy 99% pero el modelo "no detecta na	Clases desbalanceadas + métrica equivocada
Aplicar SMOTE().fit_resample(X, y) antes d	Data leakage: vecinos sintéticos se constr
precision_score lanza UndefinedMetricWarni	No hubo predicciones positivas (TP+FP=0).
Confunden ejes de la confusion matrix	En sklearn las filas son la clase real y l
Usar F1 cuando el costo de FP y FN es muy	F1 trata ambos errores como equivalentes.

Preguntas frecuentes

¿SMOTE, class_weight o threshold tuning?

Empezá siempre por class_weight='balanced' — es gratis, no toca el dataset, no genera leakage. Si no alcanza, sumá threshold tuning (también gratis, solo cambia el corte sobre predict_proba). Recurrí a SMOTE/ADASYN cuando los dos anteriores no rinden y sospechás que el modelo no separa bien la frontera de decisión. Ojo: SMOTE no es magia — en datasets tabulares ruidosos a veces empeora.

¿Precision o recall?

Depende del costo asimétrico: si un FN es caro (no detectar un cáncer, dejar pasar un fraude) → priorizá recall. Si un FP es caro (bloquear cliente legítimo, marcar email serio como spam) → priorizá precision. Cuando no hay preferencia clara, usá F1.

¿macro avg o weighted avg en multiclase?

macro avg = promedio simple por clase (todas pesan igual). Útil cuando te importan todas las clases por igual, incluyendo las raras. weighted avg = promedio ponderado por soporte (clases mayoritarias pesan más). Útil cuando refleja el costo real del negocio. En problemas desbalanceados macro avg es más honesto.

¿Por qué F1 es media armónica y no aritmética?

Porque la armónica penaliza fuerte los valores bajos. Si precision=1.0 y recall=0.0, la media aritmética da 0.5 (engañosa); la armónica da 0. F1 te obliga a que ambas sean razonables.

¿El umbral 0.5 es óptimo?

Casi nunca — es el default de predict() pero no tiene base teórica. Con precision_recall_curve sobre un set de validación encontrás el umbral que maximiza F1 (o F-beta, o la métrica de negocio). En problemas desbalanceados moverlo suele dar más mejora que cambiar de modelo.

Referencias

- Géron, cap. 3 — Performance Measures, Confusion Matrix, Precision and Recall, The Precision/Recall Trade-off.
- scikit-learn — Classification metrics
- imbalanced-learn — User guide
- imbalanced-learn — SMOTE
- Chawla et al. (2002), SMOTE: Synthetic Minority Over-sampling Technique, JAIR 16.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 064 — Clase 064 — Class imbalance: SMOTE, ADASYN, class_weight, threshold tuning

> *Parte: 1 — Machine Learning Clásico · Fuente: Chawla et al. (2002) SMOTE + He et al. (2008) ADASYN + imbalanced-learn docs. Duración estimada: 80 min.*

Objetivo

Tratar datasets desbalanceados —fraude (1 % positivo), churn (5 %), enfermedades raras—. Las trampas son sutiles: accuracy puede ser 99 % con un clasificador trivial. Cubrir las 4 estrategias estándar: class_weight, threshold tuning, oversampling (SMOTE, ADASYN), undersampling (Tomek, ENN). Y la decisión clave: ¿qué métrica reportar? (F1, PR-AUC, MCC, no accuracy).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Detectar imbalance: value_counts(normalize=True). Decidir si > 10:1 amerita tratamiento.
- Aplicar class_weight='balanced' o pesos custom en sklearn.
- Aplicar threshold tuning: optimizar el umbral de decisión sobre la curva PR según la métrica del negocio.
- Usar SMOTE (synthetic minority over-sampling) de imbalanced-learn: SMOTE(k_neighbors=5).fit_resample(X, y).
- Combinar oversampling + undersampling (SMOTETomek, SMOTEENN).
- Reportar PR-AUC y MCC (Matthews Correlation Coefficient) en lugar de accuracy.

Temas

- Imbalance ratio: >10:1 problemático.
- Métricas: precision, recall, F1, F-beta, PR-AUC, MCC.
- class_weight: penalizar más errores en minoría durante training.
- Threshold tuning: mover el umbral fuera del 0.5 default.
- SMOTE: interpola entre vecinos de la minoría.
- ADASIN: como SMOTE pero con más densidad en zonas "difíciles".
- Tomek links / ENN: remueve borderline de la mayoría.
- imbalanced-learn pipelines.

Definiciones y características

- Class imbalance: una clase mucho más frecuente que otra(s).
- class_weight='balanced': pesos automáticos inversamente proporcionales a frecuencia.
- SMOTE: para cada sample minoritario, crear sintéticos en línea recta a sus k vecinos.
- ADASIN: SMOTE adaptativo — genera más cerca de la frontera de decisión.
- PR-AUC: área bajo Precision-Recall curve. Más informativa que ROC-AUC cuando hay imbalance fuerte.
- MCC: (TPTN - FPFN) / sqrt(...). Único valor en [-1, 1]. Robusto a imbalance.
- Threshold tuning: elegir el cutoff p > τ → predict 1 que maximiza la métrica de negocio.

Dataset / recursos

- fetch_openml('creditcardfraud') (Kaggle, 0.17 % positivo).
- Librerías: imbalanced-learn (pip install imbalanced-learn), scikit-learn.

Ejercicios

1. Baseline sin tratamiento: LogisticRegression en creditcardfraud. Accuracy alto, recall pésimo.
2. class_weight: LogisticRegression(class_weight='balanced'). Recall sube, precision baja.
3. Threshold tuning: con probabilidades de predict_proba, barrer thresholds y plotear F1 vs threshold. Elegir el óptimo.
4. SMOTE: from imblearn.over_sampling import SMOTE; X_res, y_res = SMOTE().fit_resample(X_train, y_train). Entrenar y evaluar.
5. Pipeline imblearn: Pipeline([('smote', SMOTE()), ('clf', LogisticRegression())]). Importante: SMOTE solo se aplica en train (imblearn pipeline lo maneja).

Homework verificable

Sobre creditcardfraud:

1. 4 modelos: baseline, class_weight, SMOTE, SMOTETomek.
2. Reportar precision, recall, F1, PR-AUC y MCC para cada uno.
3. Curva PR de los 4 lado a lado.
4. Threshold tuning sobre el mejor para maximizar F2 (favorece recall).

Criterio de aceptación: al menos uno de los tratamientos supera al baseline en F1 por ≥ 0.1; PR-AUC ≥ 0.8.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Accuracy 99.5 % en fraude con clasificador	Reporta accuracy en imbalance. Fix: usar P
SMOTE aplicado antes de split	Leakage: samples sintéticos derivados de t
Threshold default 0.5 con probabilidades c	Subóptimo. Fix: tunear sobre val.
SMOTE con features categóricas codificadas	Interpola entre 0/1, sin sentido. Fix: SMO
Oversampling para clase de 0.1 % a 50/50	Excesivo. Fix: ratio 1:3 o 1:5 suele ser s

Preguntas frecuentes

class_weight o SMOTE?

class_weight es más simple y barato. SMOTE puede ayudar en casos extremos o con árboles/ensembles. Probá ambos.

¿Cuándo PR-AUC vs ROC-AUC?

PR-AUC cuando imbalance fuerte (>10:1) — más sensible. ROC-AUC para casos balanceados o solo para comparar relativamente.

¿MCC vs F1?

MCC es simétrico (trata clases igual). F1 favorece la minoría. Para fraude/medical, MCC es más conservador.

¿SMOTE con DL?

Menos común — DL prefiere ajustar la loss (focal loss, weighted CE).

¿Undersampling pierde información?

Sí. Por eso es un last resort. SMOTE/oversampling sintético suele ser mejor.

Referencias

- Chawla et al. (2002), SMOTE, JAIR.

- He et al. (2008), ADASYN, IJCNN.
- Saito & Rehmsmeier (2015), The Precision-Recall Plot Is More Informative than the ROC Plot, PLOS ONE.
- imbalanced-learn docs.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 065 — Clase 065 — Precision/Recall tradeoff

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 3. Duración estimada: 50 min.*

Objetivo

Que el alumno entienda que no se puede maximizar precision y recall al mismo tiempo: mover el threshold de decisión sube uno y baja el otro. La clase enseña a usar decision_function + precision_recall_curve para elegir el threshold según el costo del negocio, no según el default de 0.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Explicar el tradeoff entre precision y recall en términos del threshold del clasificador.
2. Obtener scores crudos con decision_function(X) (o predict_proba) en vez de quedarse con predict.
3. Calcular la curva con precision_recall_curve(y_true, scores) y graficarla.
4. Elegir un threshold que cumpla una restricción del negocio (ej: precision ≥ 90%).
5. Reportar average_precision_score como métrica resumen única de la curva.

Temas

#	Tema	Por qué importa
1	Threshold de decisión: el default 0 no es	Define cuántos FP y FN tolerás.
2	decision_function vs predict_proba vs pred	El primero devuelve score crudo, el último
3	precision_recall_curve	Te da los 3 arrays: precision, recall, thr
4	Elegir threshold según restricción de nego	"Precision ≥ 90%" o "Recall ≥ 95%" — depen
5	average_precision_score (AP)	Resumen escalar del área bajo la curva PR.
6	Cuándo PR > ROC	Datasets muy desbalanceados (la próxima cl

Definiciones y características

decision_function(X)

: Devuelve el score crudo del clasificador (distancia al hiperplano en SGD/SVM). Valores > threshold → clase positiva. El default de predict es threshold = 0.

Threshold de decisión

: Punto de corte sobre el score. Subirlo → menos positivos predichos → más precision, menos recall. Bajarlo → más positivos predichos → más recall, menos precision.

precision_recall_curve(y_true, scores)

: Devuelve (precisions, recalls, thresholds) para todos los thresholds posibles. Notar que precisions y recalls tienen un elemento más que thresholds (el extremo donde recall=0).

precision_score(y_true, y_pred)

: TP / (TP + FP). "De los que predije positivos, cuántos lo eran". Costo de FP alto → maximizar precision.

recall_score(y_true, y_pred)

: TP / (TP + FN). "De los positivos reales, cuántos encontré". Costo de FN alto (cáncer, fraude) → maximizar recall.

average_precision_score(y_true, scores) (AP)

: Área bajo la curva precision-recall, calculada como promedio ponderado de precisions en cada threshold. Métrica resumen — mejor que F1 cuando hay desbalance fuerte.

cross_val_predict(..., method='decision_function')

: Variante de cross_val_predict que devuelve scores crudos en vez de labels. Imprescindible para construir la curva PR sin leakage.

Dataset / recursos

MNIST (clasificador binario "es el 5 vs no") — mismo dataset que la clase 056. Permite reusar el SGDClassifier ya entrenado.

Ejercicios

1. Score crudo. Entrená SGDClassifier sobre MNIST binario "es 5". Para una imagen concreta, llamá sgd.decision_function([X[0]]) y compará con sgd.predict([X[0]]). Mostrá que predict es decision_function > 0.

2. Curva PR. Con cross_val_predict(sgd, X_train, y_train_5, cv=3, method='decision_function') obtené y_scores. Pasalos a precision_recall_curve y graficá precision y recall vs threshold en el mismo eje.

3. Threshold para precision ≥ 90%. Encontrá el threshold mínimo que garantice precision >= 0.90. Pista: thresholds[np.argmax(precisions >= 0.90)]. Aplicalo: y_pred_90 = (y_scores >= threshold_90) y verificá precision y recall resultantes.

4. Curva precision vs recall. Graficá precision (eje Y) contra recall (eje X) — la forma canónica de la curva PR. Marcá el punto correspondiente al threshold por default (0).

5. Average precision. Calculá average_precision_score(y_train_5, y_scores). Compará con el F1 que sacaste en la clase 056. ¿Cuál te parece más informativo?

Homework verificable

Notebook con MNIST binario (es 5 vs no): (a) entrenar SGDClassifier; (b) obtener y_scores con cross_val_predict(method='decision_function'); (c) graficar las dos curvas (precision/recall vs threshold y precision vs recall); (d) encontrar el threshold que da precision ≥ 90% y reportar el recall en ese punto; (e) reportar average_precision_score.

Criterio de aceptación: El threshold elegido para precision ≥ 90% verifica esa cota cuando se aplica sobre y_scores. Las dos curvas están graficadas con ejes etiquetados.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar

precisions y thresholds tienen distinto la	Es por diseño: precision_recall_curve devu
Usás predict y querés mover el threshold	predict ya colapsó el score a label. Fix:
Sacaste la curva con y_pred en vez de y_sc	La curva PR necesita scores continuos, no
Threshold elegido sobre train, no sobre va	Overfitting del threshold. Fix: elegí thre
Modelos sin decision_function (ej: RandomF	No todos los estimadores la exponen. Fix:

Preguntas frecuentes

¿Precision o recall — cuál priorizo?

Depende del costo asimétrico. Filtro de spam: un FP (mail bueno marcado como spam) cuesta más que un FN → priorizá precision. Detección de cáncer: un FN (cáncer no detectado) cuesta vidas → priorizá recall. No hay respuesta universal, hay análisis de costos.

¿decision_function o predict_proba?

Son intercambiables a los fines de la curva PR (ambas son scores monótonos). predict_proba devuelve probabilidades calibradas en [0,1] (más interpretable); decision_function devuelve scores sin acotar. Para SGDClassifier usá decision_function; para RandomForestClassifier usá predict_proba(X)[:, 1].

¿Por qué la curva PR tiene esa forma escalonada?

Cada salto corresponde a un sample que cambia de lado del threshold. Con N muestras hay hasta N thresholds distintos. En datasets chicos se nota más; con 10k+ samples se ve suave.

¿F1 o average precision?

F1 es a un threshold fijo (típicamente 0.5) — útil cuando el threshold ya está definido. Average precision integra sobre todos los thresholds — útil para comparar modelos sin fijar threshold. Para selección de modelo, AP es más informativo.

¿Y si la clase positiva es la minoría extrema (<1%)?

Ahí la curva PR brilla y la ROC engaña. Lo cierra la clase 058 (ROC y AUC).

Referencias

- Géron, cap. 3 § Precision/Recall Tradeoff y § The ROC Curve.
- scikit-learn — precision_recall_curve
- scikit-learn — average_precision_score
- scikit-learn — Precision-Recall example

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 066 — Clase 066 — Curva ROC y AUC

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 3 § The ROC Curve.*

> *Duración estimada: 50 min.*

Objetivo

Que el alumno entienda qué mide la curva ROC, calcule el AUC con scikit-learn y sepa decidir cuándo ROC es la métrica adecuada y cuándo conviene usar Precision-Recall — sobre todo en datasets desbalanceados, donde ROC tiende a mentir.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Construir la curva ROC con roc_curve graficando TPR vs FPR a distintos umbrales.
2. Calcular el AUC con roc_auc_score e interpretar el valor (0.5 = azar, 1.0 = perfecto).
3. Comparar dos clasificadores superponiendo sus curvas ROC y eligiendo por AUC.
4. Decidir ROC vs PR según la prevalencia de la clase positiva.
5. Evitar la trampa del desbalance: reconocer cuándo un AUC alto esconde mala precisión.

Temas

#	Tema	Por qué importa
1	TPR (recall) y FPR	Los dos ejes de ROC; entender qué se gana
2	Curva ROC con roc_curve	Visualiza el trade-off completo, no un pun
3	AUC con roc_auc_score	Resumen escalar e independiente del umbral
4	Diagonal de azar y modelo perfecto	Referencias visuales obligatorias.
5	ROC vs Precision-Recall	En clases desbalanceadas, ROC pinta lindo
6	Comparación de modelos	Cómo elegir entre clasificadores con AUC +

Definiciones y características

TPR (True Positive Rate / Recall / Sensibilidad)

: Proporción de positivos reales correctamente detectados. TPR = TP / (TP + FN). Es el eje Y de ROC.

FPR (False Positive Rate)

: Proporción de negativos reales clasificados erróneamente como positivos. FPR = FP / (FP + TN) = 1 - especificidad. Es el eje X de ROC.

Curva ROC (Receiver Operating Characteristic)

: Gráfico de TPR vs FPR barriendo todos los umbrales de decisión. Cada punto es un umbral. Cuanto más cerca del vértice superior izquierdo, mejor.

AUC (Area Under the Curve)

: Área bajo la curva ROC. Probabilidad de que el modelo asigne mayor score a un positivo aleatorio que a un negativo aleatorio. 0.5 = azar, 1.0 = perfecto, <0.5 = peor que tirar moneda (invertí las etiquetas).

Diagonal de azar

: Línea y = x. Representa un clasificador que adivina al azar. Toda curva ROC útil va por encima de esta diagonal.

Curva Precision-Recall (PR)

: Alternativa a ROC para clases desbalanceadas. Grafica precision vs recall. No incluye TN, por lo que es insensible al inflado por la clase mayoritaria.

ROC vs PR — regla práctica

: Usá ROC cuando las clases están balanceadas o te importan ambas por igual. Usá PR cuando la clase

positiva es rara (fraude, enfermedad, clicks) y los falsos positivos en una clase abundante distorsionan el FPR.

Dataset / recursos

- sklearn.datasets.fetch_openml('mnist_784') con el clasificador binario "es un 5" (Géron cap. 3), naturalmente desbalanceado (~10% positivos).
- Opcional: dataset sintético desbalanceado vía make_classification(weights=[0.99, 0.01]) para ver el contraste ROC vs PR en extremo.

Ejercicios

1. Scores y curva ROC. Entrená un SGDClassifier sobre "es un 5", obtené scores con cross_val_predict(..., method='decision_function') y graficá la curva ROC con roc_curve.

2. AUC. Calculá roc_auc_score(y_true, y_scores). Interpretá el valor en una línea.

3. Comparar dos modelos. Entrená un RandomForestClassifier (usá predict_proba, columna 1) y superponé ambas curvas ROC en un mismo plot. Elegí el mejor por AUC.

4. ROC vs PR en desbalance. Generá make_classification(weights=[0.99, 0.01], n_samples=10_000), entrená un modelo decente y graficá lado a lado curva ROC y curva PR (precision_recall_curve). Mostrá cómo ROC sigue "linda" mientras PR revela la pobreza real.

5. Punto operativo. Sobre la curva ROC del ejercicio 1, encontrá el umbral que maximiza TPR - FPR (índice de Youden) y reportá precision y recall en ese punto.

Homework verificable

Notebook con dataset "es un 5" de MNIST: (a) entrenar SGD y RandomForest; (b) graficar ambas ROC superpuestas con AUC en la leyenda; (c) graficar las dos curvas PR con average_precision_score en la leyenda; (d) tabla final con AUC y AP por modelo; (e) párrafo de 3-4 líneas eligiendo el ganador y justificando con qué métrica decidiste y por qué (pista: prevalencia ≈ 10%).

Criterio de aceptación: El notebook corre end-to-end. Ambas curvas en un mismo eje. La conclusión menciona explícitamente el desbalance y por qué PR/AP puede ser preferible a ROC/AUC.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
AUC = 0.99 pero la precision real es 0.10	Clase muy desbalanceada — el FPR sigue baj
roc_auc_score lanza ValueError: y should b	Le pasaste predict_proba entero (matriz N×
Curva ROC "escalonada" rara con predict	Estás pasando etiquetas duras (0/1) en lug
AUC < 0.5	El modelo está invertido o pasaste la colu
Comparo modelos solo por AUC y elijo mal	AUC promedia sobre todos los umbrales, inc

Preguntas frecuentes

¿ROC o Precision-Recall?

Si la clase positiva es rara (<10-20%) o te importa específicamente cómo te va con los positivos, usá PR. ROC incluye TN en el denominador del FPR y con muchos TN el FPR queda artificialmente bajo aunque la precision sea malísima. En clases balanceadas o cuando importan ambas clases por igual, ROC está bien.

¿Qué AUC se considera "bueno"?

Depende del dominio. Regla gruesa: 0.5 azar, 0.7 aceptable, 0.8 bueno, 0.9+ excelente, 1.0 sospechá data leakage. En medicina o fraude a veces 0.95 es lo mínimo aceptable.

¿Puedo usar AUC con multiclase?

Sí: roc_auc_score(..., multi_class='ovr') (one-vs-rest) o 'ovo' (one-vs-one). Pero para multiclase suele ser más informativo mirar la matriz de confusión y métricas por clase.

¿decision_function o predict_proba?

Da igual para ROC/AUC — ROC depende del orden de los scores, no de su escala. decision_function devuelve scores no calibrados (puede ser negativo), predict_proba devuelve probabilidades en [0, 1]. Si tu modelo no tiene predict_proba (como SGDClassifier default), usá decision_function.

¿Cómo elijo el umbral final si AUC es independiente del umbral?

AUC sirve para comparar modelos. Para deployar tenés que elegir umbral según el costo de FP vs FN en tu dominio. Métodos: índice de Youden (max(TPR - FPR)), restricción tipo "recall ≥ 0.95 y maximizá precision", o análisis de costo explícito.

Referencias

- Géron, cap. 3 § The ROC Curve y § Precision/Recall Trade-off.
- scikit-learn — roc_curve
- scikit-learn — roc_auc_score
- scikit-learn — Precision-Recall vs ROC
- Saito & Rehmsmeier (2015), The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 067 — Clase 067 — Clasificación multiclase, multilabel, multioutput

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 3. Duración estimada: 60 min.*

Objetivo

Que el alumno distinga los tres escenarios de clasificación más allá del binario — multiclase (una salida con K>2 clases), multilabel (varias etiquetas por muestra) y multioutput (varias salidas, cada una con su propio rango de valores) — y sepa qué estrategia de sklearn (OneVsRest, OneVsOne, MultiOutputClassifier) usar en cada caso, eligiendo además la métrica correcta (accuracy global, hamming loss, macro/micro F1).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Diferenciar multiclase, multilabel y multioutput con un ejemplo concreto de cada uno.
2. Elegir entre OvR y OvO según el costo computacional del clasificador base y el tamaño del dataset.

3. Entrenar un clasificador multilabel con KNeighborsClassifier y evaluarlo con f1_score(average='macro') y hamming_loss.
4. Envolver un clasificador binario en MultiOutputClassifier para resolver un problema multioutput.
5. Interpretar las salidas de predict_proba en cada escenario (lista de arrays vs array 2D).

Temas

#	Tema	Por qué importa
1	Multiclase: definición y clasificadores na	Decision Tree, Random Forest, Naive Bayes,
2	OneVsRestClassifier (OvR)	K modelos binarios — escala lineal en K, d
3	OneVsOneClassifier (OvO)	K·(K-1)/2 modelos — caro en clases, barato
4	Multilabel: y es matriz binaria K-dim	Tags de noticias, géneros de película, mul
5	Métricas multilabel: hamming loss, macro/m	Accuracy global castiga demasiado; necesit
6	Multioutput: MultiOutputClassifier / Multi	Cada salida es independiente, con sus prop

Definiciones y características

Clasificación multiclase

: Una sola etiqueta por muestra pero con K>2 valores posibles (ej.: dígitos 0–9). y es vector 1D de enteros. Algunos clasificadores la soportan nativamente (RandomForest, GaussianNB, SGDClassifier); otros (SVM, Logistic puro) son binarios y sklearn los envuelve automáticamente con OvR u OvO.

One-vs-Rest (OvR / OvA)

: Estrategia que entrena K clasificadores binarios, cada uno "esta clase vs todas las demás". Predice la clase del clasificador con mayor score. Default para la mayoría — lineal en K, ideal cuando entrenar es barato pero el clasificador base no escala con N.

One-vs-One (OvO)

: Entrena K·(K-1)/2 clasificadores binarios, uno por cada par de clases. Cada uno ve solo las muestras de sus dos clases (subsets chicos). Conviene cuando el clasificador base escala mal con N (ej.: SVM kernelizado, $O(N^2)$); sklearn lo usa por default para SVC.

Clasificación multilabel

: Cada muestra puede pertenecer a varias clases simultáneamente. y es matriz binaria (n_samples, n_labels) con 0/1 por etiqueta. Ejemplo: una foto puede tener ['perro', 'aire libre', 'soleado'] a la vez. KNeighborsClassifier, RandomForestClassifier y DecisionTreeClassifier lo soportan nativamente.

Clasificación multioutput

: Generalización: cada muestra tiene varias salidas y cada salida es a su vez multiclase (no binaria como en multilabel). Ejemplo típico de Géron: denoising de imagen — cada píxel es una "salida" con 256 valores posibles. Se resuelve con MultiOutputClassifier(base_estimator).

Hamming loss

: Fracción promedio de etiquetas mal predichas sobre el total (n_samples × n_labels). Métrica natural para multilabel — un error en una de 5 etiquetas pesa 0.2, no 1.0 como subset accuracy. Cuanto más bajo, mejor.

Macro vs micro averaging

: Macro = promedio simple de la métrica calculada por clase (cada clase pesa igual; bueno con desbalance si querés tratar todas las clases por igual). Micro = suma global de TP/FP/FN y luego cálculo (cada muestra pesa igual; bueno cuando importa el volumen total).

MultiOutputClassifier

: Wrapper de sklearn que entrena un clasificador independiente por columna de salida. No modela correlaciones entre salidas (para eso existe ClassifierChain). Trivial: MultiOutputClassifier(RandomForestClassifier()).fit(X, Y).

Dataset / recursos

MNIST (multiclase 10 clases) vía fetch_openml('mnist_784', as_frame=False). Para multilabel/multioutput se construye Y sintético sobre MNIST: etiquetas [es_grande (>=7), es_impar] → multilabel; o X_noisy → X_clean → multioutput.

Ejercicios

1. Multiclase nativo vs OvR. Entrená SGDClassifier en MNIST (10 clases) y comparalo con OneVsRestClassifier(SGDClassifier()). ¿Cuántos clasificadores entrena cada uno? Mirá .estimators_.

2. OvO con SVM. Entrená SVC() sobre un subset de 5k muestras de MNIST. Verificá que clf.decision_function(X[:1]) devuelve 45 scores = 10·9/2. Forzá luego OneVsRestClassifier(SVC()) y compará tiempos.

3. Multilabel con KNN. Construí Y_multilabel = np.c_[y >= 7, y % 2 == 1]. Entrená KNeighborsClassifier() con ese Y. Reportá f1_score(Y_test, Y_pred, average='macro') y hamming_loss(Y_test, Y_pred).

4. Macro vs micro F1. Con el modelo del ejercicio 3, calculá F1 con average='macro' y average='micro'. Si una de las etiquetas estuviera muy desbalanceada (ej.: solo 5% de positivos), ¿cuál cambiaría más y por qué?

5. Multioutput denoising. Generá X_train_noisy = X_train + np.random.randint(0, 100, X_train.shape). Entrená KNeighborsClassifier() con X_noisy como features y X_clean como target (cada píxel es una salida multiclase 0–255). Predecí y visualizá una imagen denoised.

Homework verificable

Notebook sobre MNIST con: (a) clasificador multiclase con RandomForestClassifier y matriz de confusión 10×10; (b) versión multilabel con etiquetas [es_grande, es_impar] usando KNeighborsClassifier y reporte de hamming_loss + f1_score(average='macro'); (c) ejemplo multioutput de denoising sobre 100 imágenes con ruido uniforme, mostrando 3 pares (ruidosa, denoised, original).

Criterio de aceptación: F1 macro multilabel > 0.95 sobre test. Hamming loss < 0.05. Denoising visualmente reconocible (no hace falta métrica numérica, sí imagen comparativa lado a lado).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ValueError: y should be a 1d array, got ..	El clasificador no soporta multilabel nati
Accuracy multilabel sale carísima (1.0 es	accuracy_score en multilabel exige que tod
SVC tarda eternidades en MNIST completo	SVC con kernel es O(N²)–O(N³). Sklearn ya
predict_proba de un multilabel devuelve li	Es lo esperado: una matriz (n_samples, n_c
Macro F1 da muy distinto al micro F1	Indica desbalance fuerte entre clases/etiq

Preguntas frecuentes

¿Cuándo OvR y cuándo OvO?

Default: OvR (lineal en K, simple). Usá OvO solo si el clasificador base escala mal con N (SVM kernelizado

es el caso típico) y K es chico. Sklearn ya elige bien por default — rara vez hace falta override manual.

Multilabel vs multioutput, ¿en qué se diferencian exactamente?

Multilabel = caso particular de multioutput donde cada salida es binaria (0/1 = "tiene/no tiene esa etiqueta"). Multioutput general = cada salida puede ser multiclase (ej.: 256 niveles de gris por píxel). Sklearn los trata casi igual; la diferencia es solo el rango de valores que puede tomar cada columna de Y.

¿MultiOutputClassifier modela correlaciones entre las salidas?

No — entrena un modelo independiente por columna. Si las etiquetas están correlacionadas (ej.: "es perro" y "tiene cola"), ClassifierChain aprende esa estructura: pasa la predicción de la primera etiqueta como feature extra al modelo de la siguiente.

¿Por qué cross_val_score con SGDClassifier en MNIST da accuracy alta pero la matriz de confusión muestra confusión entre 3 y 5?

Accuracy global enmascara errores localizados entre pocas clases. La matriz de confusión normalizada por filas es indispensable en multiclase — la clase 060 (la próxima) se dedica exactamente a eso.

¿Puedo combinar class_weight / SMOTE con multilabel?

class_weight='balanced' sí, etiqueta por etiqueta. SMOTE multilabel es más delicado (existe MLSMOTE en imbalanced-learn, no en sklearn core). Para el curso: usá class_weight y métricas macro.

Referencias

- Géron, cap. 3 § Multiclass / Multilabel / Multioutput Classification.
- sklearn — Multiclass and multioutput algorithms
- sklearn — MultiOutputClassifier
- sklearn — Métricas de clasificación (hamming_loss, f1_score)

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 068 — Clase 068 — Análisis de errores

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 3 § Error Analysis. Duración estimada: 60 min.*

Objetivo

Que el alumno deje de mirar el accuracy global y empiece a auditar dónde se equivoca un clasificador: confusion matrix normalizada por fila, pares de clases confundidas, inspección visual de ejemplos mal clasificados, y el loop de error analysis como puerta de entrada al data-centric AI (mejorar datos, no solo modelos).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Construir y normalizar una confusion matrix por fila (recall por clase) y leerla sin confundir filas con columnas.
2. Identificar pares de clases confundidas ordenando los off-diagonals normalizados de mayor a menor.
3. Inspeccionar visualmente ejemplos mal clasificados (hard examples) para formular hipótesis de causa raíz.
4. Decidir si la próxima iteración mejora el modelo (features, regularización, capacidad) o mejora los datos (relabel, augmentación, balancear).
5. Ejecutar el error analysis loop: entrenar → matriz → slices → hipótesis → fix → re-entrenar.

Temas

#	Tema	Por qué importa
1	Confusion matrix cruda vs normalizada por	Sin normalizar, las clases mayoritarias ta
2	Off-diagonals: qué clase se confunde con c	El error no es uniforme; suele haber 2-3 p
3	Inspección visual de hard examples	Te dice si es ruido de label, ambigüedad r
4	Slice analysis (error por subgrupo)	Accuracy global oculta sesgos por subpobla
5	Data-centric AI: cuándo arreglar datos	Más barato y efectivo que tunear hiperpará
6	Error analysis loop	Workflow iterativo y reproducible.

Definiciones y características

Confusion matrix normalizada por fila

: Matriz donde C[i,j] = P(predicho=j | real=i). La diagonal es el recall por clase; los off-diagonals son la distribución de errores condicional al label real. Siempre dividí por la suma de fila, no por el total: lo que querés ver es "del total de los i verdaderos, qué fracción cayó en j".

Error rate por clase

: 1 - recall_i. Útil para rankear clases por dificultad. Una clase con 60% recall en un modelo de 95% accuracy global es un problema invisible al accuracy.

Hard examples

: Instancias mal clasificadas con alta confianza, o cerca del borde de decisión. Inspeccionarlas a mano (50-100 alcanza) revela el 80% de las causas raíz: labels equivocados, imágenes borrosas, ambigüedad ontológica, dominio fuera de distribución.

Slice analysis

: Computar métricas no en el set completo sino en subconjuntos definidos por una variable (edad, región, tipo de cámara, longitud del texto). Detecta sesgos que el promedio entierra.

Data-centric AI

: Paradigma (Andrew Ng) que pone el foco en mejorar la calidad y consistencia de los datos en vez de iterar modelos. Mucho del error analysis es la herramienta operacional del data-centric.

Error analysis loop

: Ciclo train → confusion matrix → top-k confusiones → inspeccionar ejemplos → hipótesis (modelo/dato/feature) → intervenir → repetir. Convierte el debugging de ML de arte a proceso.

Top-k confusiones

: Los k pares (i, j) con i != j ordenados por C_norm[i,j] descendente. Son la lista priorizada de qué atacar primero.

Dataset / recursos

MNIST (sklearn.datasets.fetch_openml('mnist_784') o keras.datasets.mnist). Es el ejemplo canónico de Géron cap. 3: 10 clases, errores no uniformes (los 4↔9, 3↔5, 7↔9 son los pares clásicos que aparecen en cualquier modelo decente). Alternativa más densa: Fashion-MNIST (shirt vs t-shirt vs pullover es un caos pedagógicamente útil).

Ejercicios

1. Matriz cruda y normalizada. Entrená un SGDClassifier sobre MNIST. Calculá confusion_matrix(y_true, y_pred) y normalizá por fila (cm / cm.sum(axis=1, keepdims=True)). Plotealá con plt.matshow y poné ceros en la diagonal para que los errores se vean.

2. Top-5 confusiones. Del array normalizado, extraé los 5 pares (i, j) con mayor valor off-diagonal. Imprimí "real=i → pred=j: XX%".

3. Galería de errores. Para el par (real, pred) peor del ejercicio 2, mostrá una grilla 5×5 de imágenes mal clasificadas. Anotá si te parecen ambiguas, mal labeladas o claramente del label real.

4. Slice por grosor de trazo. Calculá la suma de píxeles por imagen como proxy de "grosor". Dividí el test set en terciles y reportá accuracy por tercil. ¿El modelo es peor con dígitos finos o gruesos?

5. Intervención data-centric. Tomá el par confundido del ejercicio 2. Augmentá el set de entrenamiento solo con esa clase (shifts de 1px) y re-entrená. Reportá el cambio en el recall de esa clase y el accuracy global.

Homework verificable

Notebook con MNIST que entregue: (a) confusion matrix normalizada por fila como heatmap; (b) tabla con los 3 pares de clases más confundidos y su porcentaje; (c) grilla de 16 ejemplos mal clasificados del par peor, con título real=X pred=Y; (d) tabla de accuracy por slice según una variable derivada (intensidad media, posición del centro de masa, lo que elijas); (e) un párrafo de hipótesis: ¿el error es de modelo o de datos? ¿qué probarías en la siguiente iteración?

Criterio de aceptación: Las filas de la matriz normalizada suman 1. El top-3 de confusiones está justificado numéricamente. La galería muestra ejemplos reales del par identificado. La hipótesis distingue explícitamente "mejoro modelo" vs "mejoro datos".

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
La matriz "se ve toda igual" / no se disti	No normalizaste, o no pusiste ceros en la
Normalicé por columna y los números no cie	Normalización por columna te da precision
Hago error analysis sobre el train set	Te miente: el modelo memorizó. Fix: siempr
"Tuneo hiperparámetros 3 días y no mejora"	Probablemente el techo es la calidad del l
Reporto solo accuracy global	Esconde clases minoritarias y subgrupos. F

Preguntas frecuentes

¿Cuándo mejoro el modelo y cuándo mejoro los datos?

Heurística operativa: si los ejemplos mal clasificados te confunden a vos también (ambiguos, mal labelados, fuera de dominio) → datos (relabel, limpiar, augmentar, recolectar más de esa clase). Si los errores son obvios para un humano pero el modelo los falla sistemáticamente → modelo (más capacidad, mejores features, menos regularización). En la práctica, el 70% de las veces son datos; por eso Ng popularizó el

data-centric.

¿Por fila o por columna se normaliza?

Por fila (axis=1) para análisis de errores estándar: te da P(pred | real) = distribución de errores condicional a la verdad. Por columna sirve para diagnosticar precision (cuando el modelo dice j, ¿qué tan seguido es realmente j?). Géron usa por fila en cap. 3.

¿Cuántos ejemplos mal clasificados hay que mirar?

50 a 100 suele alcanzar para ver patrones. Más allá tenés rendimientos decrecientes. Lo importante es que estén estratificados por el par de confusión que te interesa, no muestreados al azar.

¿cross_val_predict o un split fijo?

cross_val_predict te da predicciones out-of-fold para todo el train set sin leakage, ideal para análisis de errores con N moderado. Para producción usá un test set holdout fijo.

¿Esto reemplaza el ROC/PR curve?

No. Las curvas ROC/PR son para threshold tuning y comparación de modelos a nivel agregado. El error analysis es para debugging cualitativo y priorización. Son complementarias.

Referencias

- Géron, Hands-On ML, cap. 3 § "Error Analysis".
- Andrew Ng — A Chat with Andrew on MLOps: From Model-centric to Data-centric AI.
- scikit-learn — confusion_matrix y ConfusionMatrixDisplay.
- scikit-learn — cross_val_predict.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 069 — Clase 069 — Regresión lineal: ecuación normal vs gradient descent

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4 § Linear Regression. Duración estimada: 60 min.*

Objetivo

Que el alumno entienda regresión lineal desde adentro: la hipótesis $\hat{y} = \theta^T x$, por qué se usa MSE como costo, las dos formas de resolverla (ecuación normal cerrada vs gradient descent iterativo), y cuándo conviene cada una según el tamaño del dataset y la cantidad de features.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Escribir la hipótesis lineal $\hat{y} = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$ en forma matricial $X\theta$.

2. Derivar la ecuación normal $\theta = (X^T X)^{-1} X^T y$ y resolverla con NumPy.
3. Usar LinearRegression de sklearn y entender que internamente usa pseudoinversa SVD (más estable que la ecuación normal).
4. Comparar complejidad: ecuación normal $O(n^3)$ en features vs gradient descent $O(n)$ por iteración.
5. Justificar la elección entre forma cerrada y GD según n features y m muestras.

Temas

#	Tema	Por qué importa
1	Hipótesis lineal y notación vectorial	Base de todo modelo lineal (incluso logíst
2	MSE como función de costo	Convexa, derivable, mínimo global garantiz
3	Ecuación normal (forma cerrada)	Solución exacta en un solo paso.
4	Pseudoinversa SVD	Lo que sklearn usa por debajo — funciona a
5	Gradient descent (intuición)	Cuando n es grande, la ecuación normal n
6	Complejidad computacional	$O(n^3)$ vs $O(n)$ por iteración — define

Definiciones y características

Hipótesis lineal

: Predicción $\hat{y} = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n = \theta^T x$ (agregando $x_0 = 1$ para absorber el bias). En forma matricial sobre todo el dataset: $\hat{y} = X\theta$, donde X tiene shape $(m, n+1)$.

MSE (Mean Squared Error)

: Costo $\text{MSE}(\theta) = \frac{1}{m} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2$. Se elige porque es convexa (un único mínimo global), diferenciable en todo el dominio y penaliza más errores grandes. RMSE es la raíz, en unidades del target.

Ecuación normal

: Solución cerrada que sale de igualar el gradiente del MSE a cero: $\hat{\theta} = (X^T X)^{-1} X^T y$. Da el óptimo exacto sin iteraciones. Requiere invertir una matriz $(n+1) \times (n+1)$.

Pseudoinversa de Moore-Penrose (X^+)

: Generalización de la inversa que siempre existe, incluso si $X^T X$ es singular (features colineales) o no cuadrada. Se computa vía SVD: $X = U \Sigma V^T \Rightarrow X^+ = V \Sigma^+ U^T$. Sklearn usa np.linalg.lstsq (basado en SVD) — más estable numéricamente que invertir $X^T X$.

Complejidad $O(n^3)$

: Invertir $X^T X$ cuesta entre $O(n^{2.4})$ y $O(n^3)$ según el algoritmo. Con $n = 100$ features va instantáneo; con $n = 100{,}000$ es inviable. La SVD es del mismo orden. En m (muestras) la ecuación normal es lineal, así que escala bien en filas, mal en columnas.

Gradient descent (intuición)

: Algoritmo iterativo: arrancás con θ random y vas restando $\eta \cdot \nabla_\theta \text{MSE}(\theta)$ hasta converger. Cada paso cuesta $O(mn)$ (batch) — mucho menos que invertir cuando n es grande. La tasa de aprendizaje η es el hiperparámetro crítico.

LinearRegression (sklearn)

: Estimador en sklearn.linear_model. No usa ecuación normal — usa scipy.linalg.lstsq (pseudoinversa SVD). Sin hiperparámetros relevantes salvo fit_intercept. Atributos post-fit: coef_ (los $\theta_1 \dots \theta_n$) e

intercept_ (θ_0).

fit_intercept=True

: Agrega automáticamente la columna de unos. Si tus features ya están centradas en cero (mean=0), podés ponerlo en False. Default True — dejalo así salvo que sepas lo que hacés.

coef_

: Array con los pesos aprendidos, una entrada por feature. Su signo y magnitud son interpretables solo si las features están en la misma escala (de ahí la importancia de StandardScaler previo).

Dataset / recursos

Dataset sintético generado con NumPy: $y = 4 + 3x + \text{ruido gaussiano}$, $m = 100$ muestras. Permite comparar el θ recuperado con el verdadero $[4, 3]$.

Ejercicios

1. Hipótesis a mano. Generá X sintético ($m=100$, $n=1$) con $y = 4 + 3x + \mathcal{N}(0, 1)$. Resolvé $\hat{\theta}$ con la ecuación normal usando solo NumPy (np.linalg.inv, @). Verificá que $\hat{\theta} \approx [4, 3]$.

2. Pseudoinversa. Repetí el ejercicio 1 con np.linalg.pinv(X_b) @ y. Compará el resultado con la ecuación normal — deberían dar lo mismo en este caso bien-condicionado.

3. sklearn. Ajustá LinearRegression al mismo dataset. Verificá que lin_reg.intercept_ ≈ 4 y lin_reg.coef_ ≈ [3]. Predecí en $x = [[0], [2]]$.

4. Caso singular. Construí X con dos features colineales ($x_2 = 2 x_1$). Intentá la ecuación normal — np.linalg.inv tira LinAlgError o devuelve basura. Usá np.linalg.pinv y observá que sí funciona (distribuye el peso entre las dos features).

5. Complejidad empírica. Cronometrá LinearRegression().fit(X, y) con $n = 100, 1000, 10000$ features (y m fijo). Graficá el tiempo — debería crecer cúbicamente.

Homework verificable

Notebook con dataset California Housing (sklearn.datasets.fetch_california_housing): (a) split 80/20 con random_state=42; (b) ajustar LinearRegression; (c) reportar R^2 y RMSE en test; (d) imprimir coef_ mapeado a nombres de feature; (e) resolver manualmente la ecuación normal en los datos escalados y verificar que da los mismos coeficientes que sklearn (módulo escalado del intercept).

Criterio de aceptación: RMSE en test < 0.75 (en unidades de la variable target). Los coeficientes manuales coinciden con sklearn con tolerancia 1e-6.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
LinAlgError: Singular matrix al hacer np.l	Features colineales o $m < n$. Fix: usá np
coef_ enormes y de signo contrario a lo es	Features no escaladas y multicolinealidad.
Olvido de la columna de unos en la ecuació	Resolvés sin intercept y el modelo pasa po
LinearRegression da resultado distinto al	Olvidaste agregar la columna de unos en un
"Mi modelo lineal tarda horas con 50k feat	Ecuación normal $O(n^3)$ no escala. Fix: c

Preguntas frecuentes

¿Sklearn usa la ecuación normal por dentro?

No. Usa scipy.linalg.lstsq, que internamente factoriza con SVD (pseudoinversa). Es más estable numéricamente que $(X^T X)^{-1} X^T y$ y funciona aunque la matriz sea singular o rectangular.

¿Por qué MSE y no MAE como costo?

MSE es diferenciable en todo el dominio (MAE no lo es en cero) y convexa, así que el mínimo es único y se llega con gradient descent sin sustos. MAE es más robusto a outliers pero requiere métodos no-suaves (programación lineal o subgradiente).

¿Cuándo elijo ecuación normal vs gradient descent?

Regla práctica: n < ~10,000 features → ecuación normal/SVD (un shot, sin tunear hiperparámetros). n >> 10,000 o no entra en RAM → gradient descent (típicamente SGD). En m (muestras) ambos escalan bien, así que millones de filas con pocas columnas → ecuación normal sin drama.

¿Necesito escalar features para regresión lineal?

Para que el modelo funcione, no — la ecuación normal da exactamente la misma predicción con o sin escalado. Para interpretar coef_ (comparar importancia entre features), sí. Para gradient descent, sí o sí (sin escalar la convergencia es lentísima).

¿Qué pasa si tengo más features que muestras ($n > m$)?

$X^T X$ es singular → la ecuación normal falla. La pseudoinversa SVD devuelve la solución de norma mínima entre las infinitas posibles, pero el modelo va a sobreajustar feísimo. Fix real: regularización (Ridge/Lasso, clase 063) o reducción de dimensionalidad.

Referencias

- Géron, cap. 4 § Linear Regression, The Normal Equation, Computational Complexity.
- sklearn LinearRegression
- NumPy linalg.pinv
- Wikipedia — Moore-Penrose pseudoinverse

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 070 — Clase 070 — Gradient Descent: batch, stochastic, mini-batch

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4. Duración estimada: 60 min.*

Objetivo

Que el alumno entienda gradient descent como motor de optimización para entrenar modelos lineales cuando la ecuación normal no escala, y sepa elegir entre batch (BGD), stochastic (SGD) y mini-batch GD según tamaño del dataset, ruido tolerable y costo por iteración. Que además dimensione el rol del learning rate y del feature scaling, y use SGDRegressor de scikit-learn.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Explicar el gradiente de la MSE en regresión lineal y por qué moverse en - minimiza el costo.
2. Diferenciar BGD vs SGD vs mini-batch en términos de costo por iteración, varianza del paso y memoria.
3. Diagnosticar la curva de costo (divergente, oscilante, lenta, suave) e inferir si el learning_rate está mal seteado.
4. Aplicar feature scaling (StandardScaler) antes de cualquier GD y justificar por qué sin escalado SGD diverge o tarda 100×.
5. Entrenar SGDRegressor con learning_rate='invscaling' y comparar coeficientes contra LinearRegression (ecuación normal).

Temas

#	Tema	Por qué importa
1	Gradiente de la MSE y regla de update θ :=	Es el núcleo de todo ML moderno (incluye r
2	Batch GD: usa todo el dataset por step	Convergencia suave, pero O(m) por iteració
3	Stochastic GD: 1 muestra por step	Muy rápido y escala, pero ruidoso; nunca c
4	Mini-batch GD: lotes de 32–256	Equilibrio práctico, aprovecha vectorizaci
5	Learning rate η y learning schedule	Demasiado alto diverge, demasiado bajo no
6	Feature scaling como prerrequisito	Sin escalar, las curvas de nivel son elips
7	SGDRegressor de sklearn	API estándar, soporta partial_fit (online

Definiciones y características

Gradiente MSE(θ)

: Vector de derivadas parciales del costo respecto a cada parámetro. Apunta en la dirección de máximo crecimiento del error → restarlo (-η·) reduce el costo. Para MSE: = (2/m)·X^T(Xθ - y).

Learning rate η (eta)

: Tamaño del paso. Hiperparámetro crítico. Muy alto: el costo diverge o oscila. Muy bajo: tarda eternidades. Valores típicos: 0.001 a 0.1.

Batch Gradient Descent (BGD)

: Cada update usa todas las m muestras. Costo por iteración O(m·n). Determinista, converge suave al mínimo global (en problemas convexos como MSE). Inviable con m > 10^6.

Stochastic Gradient Descent (SGD)

: Cada update usa una sola muestra (elegida al azar). Costo O(n) por iteración. Camino ruidoso, "rebota" cerca del mínimo sin asentarse → requiere learning_schedule decreciente para que el ruido baje con el tiempo.

Mini-batch Gradient Descent

: Usa lotes de tamaño b (típicamente 32, 64, 128, 256). Combina lo mejor: vectorizable (rápido en GPU), menos ruidoso que SGD, mucho más liviano que BGD. Es el estándar de facto en deep learning.

Epoch

: Una pasada completa sobre el dataset. En SGD, una epoch = m updates. En mini-batch con batch_size b, una epoch = m/b updates.

Learning schedule

: Función que baja η con el tiempo. Patrón común: η_t = η / (1 + decay·t). En sklearn: learning_rate='invscaling' con eta0 y power_t.

Feature scaling

: Llevar las features a escala comparable (StandardScaler: media 0, std 1). Sin esto, una feature con valores [0, 10^6] domina el gradiente y otra con [0, 1] es invisible → GD zigzaguea por un valle alargado.

Dataset / recursos

sklearn.datasets.fetch_california_housing (20.640 filas, 8 features, escalas muy distintas → caso ideal para mostrar la necesidad de scaling).

Ejercicios

1. BGD a mano. Implementá BGD en NumPy para regresión lineal sobre un dataset sintético (y = 4 + 3x + ruido). Loopeá 1000 iteraciones con η=0.1. Graficá la trayectoria de θ, θ y la curva de costo.

2. SGD a mano. Mismo dataset. Implementá SGD con learning_schedule η_t = 5 / (50 + t). Compará la trayectoria contra BGD: tendría que ser visiblemente más ruidosa pero más rápida en wall-clock.

3. Efecto del learning rate. Corré BGD con η ∈ {0.001, 0.01, 0.1, 0.5, 1.0}. Graficá las 5 curvas de costo en un mismo plot. Identificá cuál diverge y cuál es absurdamente lenta.

4. Scaling sí/no. Sobre California Housing, entrená SGDRegressor (a) sin escalar y (b) con StandardScaler. Reportá n_iter_ y score en cada caso. Tiene que haber un orden de magnitud de diferencia.

5. SGDRegressor vs LinearRegression. Entrená ambos sobre California Housing escalado. Compará coeficientes y R^2. Tienen que dar muy parecidos (SGD es aproximación estocástica de la solución cerrada).

Homework verificable

Notebook con California Housing: (a) train/test split 80/20; (b) pipeline StandardScaler + SGDRegressor(max_iter=1000, tol=1e-3, learning_rate='invscaling', eta0=0.01); (c) reportá R^2 en test y n_iter_ real; (d) repetí sin scaler y mostrá que n_iter_ se dispara o que R^2 cae; (e) graficá la curva de loss vs epoch usando partial_fit en loop manual.

Criterio de aceptación: R^2 test ≥ 0.55 con scaler. Sin scaler, R^2 baja al menos 0.1 puntos o aparece ConvergenceWarning.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ConvergenceWarning: Maximum number of iter	Olvidaste el StandardScaler. Fix: meté SGD
El costo diverge (sube en lugar de bajar)	learning_rate demasiado alto. Fix: bajalo
SGD nunca "se asienta", el costo oscila pa	Falta learning_schedule que baje η con el
Coeficientes de SGDRegressor distintos a L	Si entrenaste con datos no escalados, los
partial_fit parece arrancar de cero en cad	Si reinstanciás el modelo dentro del loop,

Preguntas frecuentes

¿BGD, SGD o mini-batch?

Mini-batch en el 90% de los casos. BGD solo si el dataset entra cómodo en RAM y m < ~10^4. SGD puro (batch_size=1) casi nunca en la práctica — sirve para entender el concepto y para online learning donde

llegan muestras una por una. SGDRegressor de sklearn internamente puede comportarse como mini-batch dependiendo del solver.

¿Por qué LinearRegression no usa GD?

Porque para MSE existe solución cerrada (ecuación normal: $\theta = (X^TX)^{-1}X^Ty$). Es exacta y no tiene hiperparámetros. Pero invertir X^TX es $O(n^3)$ en features y $O(m \cdot n^2)$ en muestras → con muchas features ($n > 10^4$) o X^TX mal condicionada, GD gana.

¿Cómo elijo η?

Empezá con 0.01. Si diverge, dividilo por 10. Si converge pero lento, multiplicalo por 3. Mejor aún: grid search sobre eta0 con cross-validation. En deep learning hay schedulers más sofisticados (Adam, cosine annealing), pero para ML clásico invscaling alcanza.

¿Qué tamaño de mini-batch uso?

32, 64, 128 o 256. Potencias de 2 porque la GPU las prefiere. Más chico → más ruido, más updates por epoch. Más grande → más estable, pero menos pasos. 32 es el default histórico (Bengio); 256 es común con datasets grandes.

¿StandardScaler o MinMaxScaler antes de GD?

StandardScaler por default — centra en 0 (importante para que el gradiente no tenga sesgo direccional) y normaliza varianza. MinMaxScaler lo usás si necesitás un rango acotado (ej. imágenes a [0,1]).

Referencias

- Géron, cap. 4 — Training Models, sección "Gradient Descent".
- scikit-learn — SGDRegressor
- scikit-learn — Stochastic Gradient Descent user guide
- Sebastian Ruder — An overview of gradient descent optimization algorithms

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 071 — Clase 071 — Regresión polinomial

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4 § Polynomial Regression. Duración estimada: 50 min.*

Objetivo

Que el alumno ajuste modelos lineales a relaciones no lineales usando PolynomialFeatures de scikit-learn, entienda la combinatoria de features que esto genera, y reconozca el riesgo de overfitting cuando el grado crece.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Transformar features con PolynomialFeatures(degree=d) y entender qué columnas produce.
2. Ajustar un LinearRegression sobre features polinómicas y graficar la curva resultante.
3. Calcular cuántas features genera grado d con n variables originales (combinatoria con repetición).
4. Diagnosticar overfitting comparando RMSE en train vs test al subir el grado.
5. Decidir cuándo usar interaction_only=True vs incluir potencias.

Temas

#	Tema	Por qué importa
1	Modelo lineal sobre features no lineales	Linealidad es en los coeficientes, no en l
2	PolynomialFeatures(degree, include_bias, i	API para expandir el feature space.
3	Combinatoria de features: C(n+d, d)	Explosión cuadrática/cúbica con n features
4	Overfitting con grado alto	Curva oscila para pasar por todos los punt
5	Validación con train/test split	Sin split, grado alto siempre "gana" en tr
6	Interaction-only vs full polynomial	Cuándo importan las potencias y cuándo sol

Definiciones y características

PolynomialFeatures

: Transformer de sklearn.preprocessing que genera todas las combinaciones polinómicas de las features de entrada hasta grado degree. Para x con grado 2: [1, x, x²]. Para [x, x] grado 2: [1, x, x, x², xx, x²].

degree

: Grado máximo del polinomio. Grado 1 no transforma (más bias). Grado 2–3 cubre la mayoría de curvaturas suaves. Grado ≥10 casi siempre es overfit.

interaction_only=True

: Genera solo productos cruzados entre features distintas, sin potencias (x², x² quedan fuera). Útil cuando el efecto no lineal viene de interacciones, no de curvatura individual.

include_bias=True (default)

: Agrega columna de 1's. Conviene ponerlo en False si después usás LinearRegression (que ya tiene fit_intercept=True) para no duplicar.

Explosión combinatoria

: La cantidad de features generadas es C(n+d, d) = (n+d)! / (n! d!). Con n=10 y d=5: 3003 features. Con n=100 y d=3: 176851. Escala feo.

Generalización

: Capacidad del modelo de funcionar en datos no vistos. Subir grado mejora train pero llega un punto donde test empeora — es el síntoma clásico de overfitting.

Modelo lineal sobre features no lineales

: Ajustar y = w + wx + wx² es resolver un problema lineal en (w, w, w) aunque la curva en x sea una parábola. Por eso LinearRegression basta tras transformar.

Dataset / recursos

Sintético: y = 0.5 x² + x + 2 + ruido_gaussiano, con x [-3, 3], n=100. Ideal para visualizar curva ajustada y comparar grados.

Ejercicios

1. Generar dataset cuadrático. x = np.linspace(-3, 3, 100) + ruido. Graficar scatter.

2. Ajustar grado 2. PolynomialFeatures(degree=2, include_bias=False) + LinearRegression. Imprimir coef_ e intercept_ y compararlos con los del DGP (0.5, 1, 2).

3. Grafo de curvas. Ajustar grados 1, 2, 5, 30 sobre el mismo dataset y plotear las 4 curvas superpuestas al scatter. Observar oscilaciones en grado 30.

4. Curva train/test vs grado. Para grado 1 a 20, calcular RMSE en train y en test. Graficar ambas curvas en función del grado. Identificar el punto donde test empieza a subir.

5. Combinatoria. Con n_features=3 de entrada, contar columnas que devuelve PolynomialFeatures(degree=4, include_bias=False). Verificar con la fórmula C(n+d, d) - 1.

Homework verificable

Notebook con: (a) dataset sintético y = sin(x) + ruido en x [0, 2π]; (b) ajustar polinomios de grado 1 a 15; (c) split 80/20; (d) graficar RMSE train vs test por grado; (e) reportar el grado óptimo según test; (f) graficar curva del grado óptimo encima del scatter.

Criterio de aceptación: El grado óptimo reportado está entre 3 y 7 (zona razonable para sin con ruido). Curva train decrece monótona; curva test tiene forma de U.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
LinearRegression da coef_ con 2 columnas d	Dejaste include_bias=True y LinearRegressi
RMSE train baja pero test explota al subir	Overfitting clásico. Fix: bajá el grado, o
MemoryError con grado 10 y 50 features	Combinatoria: C(60, 10) ≈ 75M. Fix: reducí
Coeficientes enormes (1e8) con grado alto	Síntoma de overfitting + colinealidad entr
Olvidaste transformar el test set	Aplicaste .fit_transform en train y .fit_t

Preguntas frecuentes

¿PolynomialFeatures es un modelo no lineal?

No. Es una transformación del input. El modelo (LinearRegression) sigue siendo lineal en los coeficientes. Lo que es no lineal es la relación entre x original e y.

¿Qué grado elegir por default?

Empezá con 2. Si el residual muestra patrón curvo, subí a 3. Más allá de 4–5 rara vez es necesario en problemas reales — si lo necesitás, probablemente quieras un modelo no lineal (árboles, kernel) en vez de subir grado.

¿Cuándo interaction_only=True?

Cuando sabés (o sospechás) que el efecto no lineal viene de combinaciones entre features (ej: precio × cantidad), no de curvatura individual de cada una. Reduce muchísimo la cantidad de columnas.

¿Por qué siempre escalar antes de PolynomialFeatures?

Porque x^2 y x^3 agrandan rangos: si x [0, 1000], entonces x^3 [0, 1e9]. Eso degrada el condicionamiento numérico y mata cualquier regularización posterior. Escalá primero con StandardScaler.

¿Cómo se relaciona esto con la clase 064?

Polinomial es el ejemplo canónico de modelo con alto bias en grado bajo y alta varianza en grado alto. La 064 formaliza ese trade-off con curvas de aprendizaje.

Referencias

- Géron, cap. 4 § Polynomial Regression y § Learning Curves.
- sklearn PolynomialFeatures
- sklearn user guide § Polynomial regression

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 072 — Clase 072 — Curvas de aprendizaje y bias-variance tradeoff

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4. Duración estimada: 60 min.*

Objetivo

Diagnosticar si un modelo sufre de alto sesgo o alta varianza leyendo curvas de aprendizaje (sklearn.model_selection.learning_curve) y decidir, con criterio, si conviene conseguir más datos, aumentar la capacidad del modelo o regularizar.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. Graficar una curva de aprendizaje (RMSE train vs. RMSE validación en función de train_size) con learning_curve.
2. Identificar patrones canónicos: underfitting (curvas altas y juntas) vs. overfitting (gap persistente).
3. Descomponer conceptualmente el error esperado en bias² + variance + irreducible noise.
4. Decidir acción correctiva apropiada: más datos, más capacidad, features nuevas, o regularización.
5. Justificar por qué "más datos" no siempre es la solución (caso de high bias).

Temas

- Curva de aprendizaje: qué se plotea y cómo se lee.
- Patrón de underfitting: ambas curvas convergen alto → más datos no ayuda.
- Patrón de overfitting: gap grande train/val → más datos sí ayuda, o regularizar.
- Descomposición bias-variance del error de generalización.
- Error irreducible (ruido de Bayes): cota inferior inevitable.
- Tradeoff: aumentar capacidad ↓ bias pero ↑ variance.
- Diagnóstico operacional con learning_curve y validation_curve.

Definiciones y características

- Learning curve (curva de aprendizaje) — gráfico del error de entrenamiento y validación en función del tamaño del conjunto de entrenamiento. Permite diagnosticar capacidad vs. datos.
- Bias (sesgo) — error por supuestos erróneos del modelo (p. ej., asumir lineal lo no-lineal). Modelos con alto sesgo subajustan.
- Variance (varianza) — sensibilidad del modelo a pequeñas variaciones en los datos de entrenamiento.

Modelos con alta varianza sobreajustan.

- Irreducible error — ruido intrínseco de los datos (Var(ε)); ninguna mejora de modelo lo elimina. Es la cota inferior.
- Diagnóstico high-bias — train_error alto y val_error alto, ambas curvas convergen a un valor alto cuando m crece. Síntoma: más datos no mueven la aguja.
- Diagnóstico high-variance — train_error bajo, val_error notablemente más alto, gap persistente. Síntoma: el modelo memoriza; más datos o regularización deberían cerrar el gap.
- Bias-variance tradeoff — E[(y - ŷ)²] = Bias² + Var + σ². Reducir uno suele aumentar el otro; el óptimo está en el medio.
- Capacidad del modelo — grados de libertad efectivos (grado del polinomio, profundidad del árbol, nº de parámetros). Más capacidad menos bias, más variance.

Dataset / recursos

- Dataset sintético tipo "noisy quadratic": y = 0.5·x² + x + 2 + ε con np.random.randn. Permite controlar la complejidad
- sklearn.datasets.fetch_california_housing (subset) para una corrida sobre datos reales.
- API clave: sklearn.model_selection.learning_curve, validation_curve.

Ejercicios

1. Curva base. Generá 200 puntos del dataset cuadrático ruidoso. Ajustá una regresión lineal y graficá la curva de aprendiz
1. Aumentar capacidad. Repetí con PolynomialFeatures(degree=2) + LinearRegression. Comparalo con degree=10. Identificá cu
1. ¿Más datos ayudan? Para el polinomio de grado 10, extendé el dataset a 2000 puntos y volvé a plotear. ¿Se cierra e
1. Validation curve. Usá validation_curve para barrer degree de 1 a 15 sobre el mismo dataset. Encontrá el "sweet spot
1. Bias-variance empírico. Entrená 100 modelos degree=10 sobre bootstraps del dataset y calculá, para una grilla de x de test

bias² + var se acerca al MSE total menos σ².

Homework verificable

Sobre el dataset california_housing:

1. Entrená DecisionTreeRegressor(max_depth=d) para d {2, 5, 10, None}.
2. Para cada uno, generá la curva de aprendizaje con learning_curve(cv=5, scoring='neg_root_mean_squared_error').
3. Clasificá cada modelo como underfit, fit razonable o overfit justificando con el gap final y el nivel de error.
1. Recomendá explícitamente, para cada caso, una de tres acciones: ["más datos", "más capacidad", "regularizar/podar"].

Criterio de aceptación: un .py o notebook que, al ejecutarse, imprima una tabla con columnas

depth | train_rmse_final | val_rmse_final | gap | diagnostico | accion_recomendada y guarde los 4 gráficos en figs/learning_curve_depth_{d}.png.

Errores comunes

- Confundir curva de aprendizaje con validation curve. La primera varía m (tamaño de train); la segunda varía un hiperparám
- Leer el error de train absoluto como diagnóstico. Lo relevante es el gap y la tendencia asintótica, no un punto ai
- Recomendar "más datos" frente a high bias. Si ambas curvas ya convergieron alto, sumar filas no baja el error; hay que ca
- No promediar sobre CV. Una sola partición es ruidosa; learning_curve ya hace CV interno — usalo.
- Olvidar el irreducible error. Apuntar a val_rmse = 0 es absurdo si σ del ruido es positivo; siempre hay piso.

Preguntas frecuentes

- ¿Más datos o modelo más complejo? Mirá el gap. Gap grande con train bajo overfit más datos o regularización ayudan. Ga
- ¿Cuántos puntos uso para train_sizes? Por defecto np.linspace(0.1, 1.0, 10) está bien. Para datasets grandes, usá menos
- ¿Por qué el train_error sube al aumentar m? Con pocos datos el modelo memoriza (train_error ≈ 0); al sumar filas le cues
- ¿Sirve para clasificación? Sí. Usá scoring='accuracy' o 'neg_log_loss'; la interpretación del gap es análoga.
- ¿Bias-variance se mide en la práctica? Conceptualmente sí (vía bootstrap, como el ejercicio 5), pero en producción

Referencias

- Géron, A. Hands-On Machine Learning, 3ª ed., cap. 4 — "Learning Curves" y cap. 7 — "Bias/Variance Tradeoff".
- scikit-learn user guide: Validation curves: plotting scores to evaluate models.
- API: sklearn.model_selection.learning_curve.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 073 — Clase 073 — Regularización: Ridge, Lasso, Elastic Net

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4. Duración estimada: 70 min.*

Objetivo

Aprender a controlar el overfitting en modelos lineales mediante regularización L2 (Ridge), L1 (Lasso) y su combinación (Elastic Net), entendiendo el rol del hiperparámetro alpha, la importancia del scaling, y cuándo conviene cada variante.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar qué es la regularización y por qué reduce la varianza de un modelo lineal.
- Implementar Ridge, Lasso y ElasticNet de scikit-learn sobre un dataset escalado.
- Tunear alpha con RidgeCV / LassoCV / ElasticNetCV y leer los coeficientes resultantes.
- Justificar la elección entre L1, L2 y L1+L2 según el problema (multicolinealidad, sparsity, número de features).
- Diagnosticar por qué Lasso "anula" features y Ridge solo las "encoge".

Temas

- Sesgo–varianza y motivación de la regularización.
- Ridge (L2): penalización $\alpha \cdot \Sigma\beta^2$. Encoge coeficientes hacia cero sin anularlos.
- Lasso (L1): penalización $\alpha \cdot \Sigma|\beta|$. Produce soluciones sparse (selección de features).
- Elastic Net: combinación L1+L2 con l1_ratio.
- Hiperparámetro alpha: efecto en bias/varianza; $\alpha=0 \equiv$ OLS; $\alpha \to \infty \equiv$ modelo nulo.
- Scaling obligatorio (StandardScaler) antes de regularizar.
- Selección de α con CV: RidgeCV, LassoCV, ElasticNetCV.

Definiciones y características

- Ridge (regresión L2): añade $\alpha \cdot \Sigma\beta^2$ a la función de costo. Tiene solución cerrada. No anula coeficientes; los acerca a cero proporcionalmente. Robusto frente a multicolinealidad.
- Lasso (regresión L1): añade $\alpha \cdot \Sigma|\beta|$. No tiene solución cerrada (se resuelve con coordinate descent). Produce soluciones esparsas: muchos coeficientes quedan exactamente en 0 → hace selección automática de features.
- Elastic Net: combina L1 y L2: $\alpha \cdot (l1\_ratio \cdot \Sigma|\beta| + (1 - l1\_ratio)/2 \cdot \Sigma\beta^2)$. Útil cuando hay más features que muestras o features muy correlacionadas (Lasso puro "elige una al azar" del grupo correlacionado).
- alpha (α): fuerza de la regularización. Más alto → más penalización → coeficientes más chicos → más bias, menos varianza. Se tunea por CV.
- L1 vs L2: L1 promueve sparsity (esquinas del rombo $|\beta|$); L2 promueve coeficientes pequeños y suaves (círculo β^2). L1 hace selección, L2 no.
- Sparsity: propiedad de un vector de coeficientes con muchos ceros. Útil para interpretabilidad y costo computacional en predicción.
- RidgeCV / LassoCV / ElasticNetCV: variantes que internamente buscan alpha (y l1_ratio) por cross-validation sobre una grilla. Más eficiente que GridSearchCV para estos modelos.
- Scaling: como la penalización suma cuadrados/absolutos de coeficientes, una feature con escala grande recibe penalización injustamente alta. Siempre escalar antes.

Dataset / recursos

- Dataset sugerido: sklearn.datasets.fetch_california_housing (regresión continua, 8 features con escalas distintas → ideal para mostrar scaling).
- Alternativa sintética: make_regression(n_features=50, n_informative=10, noise=10) para ver cómo Lasso anula las 40 no informativas.

- Stack: numpy, pandas, scikit-learn (Ridge, Lasso, ElasticNet, RidgeCV, LassoCV, ElasticNetCV, StandardScaler, Pipeline).

Ejercicios

1. OLS baseline: entrená LinearRegression sobre California Housing escalado. Reportá RMSE en train y test. Anotá los coeficientes.
2. Ridge: entrená Ridge(alpha=1.0) sobre los mismos datos. Compará RMSE y la magnitud de los coeficientes vs OLS.
3. Lasso: entrená Lasso(alpha=0.1). Contá cuántos coeficientes quedaron exactamente en 0. Subí alpha a 1.0 y a 10.0; observá la sparsity creciente.
4. Elastic Net: entrená ElasticNet(alpha=0.1, l1_ratio=0.5). Compará con Ridge y Lasso puros.
5. Tuning con CV: usá RidgeCV(alphas=np.logspace(-3, 3, 50)) y LassoCV(cv=5) para encontrar el mejor alpha. Graficá el path de coeficientes vs alpha (Lasso path).

Homework verificable

Sobre make_regression(n_samples=200, n_features=50, n_informative=10, noise=10, random_state=42):

1. Entrená OLS, Ridge, Lasso y Elastic Net (todos con StandardScaler en Pipeline).
2. Tuneá alpha con la variante *CV correspondiente.
3. Reportá: RMSE en test (split 80/20, random_state=42), número de coeficientes ≠ 0, y top-5 features por |coef| en Lasso.

Criterio de verificación: Lasso debe identificar al menos 8 de las 10 features informativas (coef ≠ 0) y dejar en 0 al menos 30 de las 40 ruidosas. RMSE de Ridge y Elastic Net dentro de ±10% del de Lasso.

Errores comunes

1. No escalar features antes de Ridge/Lasso/Elastic Net: la penalización castiga a las features de mayor escala. Siempre StandardScaler (o equivalente) en un Pipeline.
2. Usar alpha=0: equivale a OLS y además dispara warnings/inestabilidad numérica en Lasso. Si querés OLS, usá LinearRegression.
3. Confundir alpha de sklearn con λ de la teoría: en sklearn alpha ya incluye el factor 1/(2n) o 1/n según el modelo. No es directamente comparable entre Ridge y Lasso.
4. Tunear alpha sobre el test set: usá CV (RidgeCV, LassoCV) o GridSearchCV sobre train. El test se toca una sola vez al final.
5. Esperar que Ridge haga selección de features: no las anula, solo las encoge. Si querés sparsity, usá Lasso o Elastic Net.

Preguntas frecuentes

1. ¿Ridge, Lasso o Elastic Net?

- Ridge: pocas features, todas potencialmente relevantes, posible multicolinealidad.
- Lasso: muchas features, sospechás que muchas son irrelevantes, querés un modelo interpretable.
- Elastic Net: muchas features correlacionadas entre sí, o p > n (más features que muestras).

1. ¿Por qué Lasso anula coeficientes y Ridge no?

Geométrica
restricción
sobre un ej

1. ¿Qué pasa si alpha es muy grande?

Todos los
Underfitting

1. ¿Cómo elijo el rango de alpha para buscar?

Una grilla lo

1. ¿Sirve regularización con árboles o solo con modelos lineales?

Ridge/Lasso
regularizaci

Referencias

- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (3.ª ed.), cap. 4 — sección "Regularized Linear Models".
- scikit-learn — Linear Models: Ridge, Lasso, Elastic Net.
- Hastie, Tibshirani, Friedman — The Elements of Statistical Learning, cap. 3.4 (Shrinkage Methods).
- Zou & Hastie (2005), "Regularization and variable selection via the elastic net".

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 074 — Clase 074 — Early stopping

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4. Duración estimada: 45 min.*

Objetivo

Aplicar early stopping como técnica de regularización implícita en entrenamientos iterativos: monitorear la pérdida de validación durante el descenso por gradiente y detener el ajuste cuando deja de mejorar, conservando el mejor modelo visto.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar por qué detener el entrenamiento antes del óptimo de train actúa como regularización.
- Configurar SGDRegressor(early_stopping=True) con validation_fraction, n_iter_no_change y tol.
- Graficar curvas de train loss vs. validation loss e identificar la best epoch.
- Implementar manualmente un loop con paciencia y snapshot del mejor modelo (partial_fit).
- Decidir cuándo early stopping reemplaza o complementa a Ridge/Lasso.

Temas

- Sobreajuste en entrenamientos iterativos (SGD, gradient boosting, redes neuronales).
- Curva de aprendizaje por época: train baja, validación baja y luego sube.
- Mecánica del early stopping: monitorear val loss + criterio de paciencia.
- API de scikit-learn: SGDRegressor / SGDClassifier con early_stopping=True.
- Implementación manual con partial_fit + copy.deepcopy del mejor estimador.
- Relación con otras regularizaciones (L1, L2, dropout en deep learning).

Definiciones y características

- Early stopping: detener el entrenamiento iterativo cuando una métrica de validación deja de mejorar durante un número fijo de iteraciones, devolviendo los parámetros del mejor punto observado.

- Validation loss: error medido sobre un split separado del de entrenamiento; es la señal que decide cuándo cortar. No debe usarse el test set para esta decisión.
- Patience (paciencia) / n_iter_no_change: cantidad de épocas consecutivas sin mejora superior a tol que se toleran antes de detener. Valores típicos: 5–20.
- Best epoch snapshot: copia de los parámetros (coef_, intercept_) en la época con menor val loss. Sin este snapshot, al cortar nos quedaríamos con un modelo ya degradado.
- tol: umbral mínimo de mejora para considerar que hubo progreso. Si loss_actual > loss_best - tol, la época no cuenta como mejora.
- Regularización implícita: a diferencia de Ridge/Lasso (que penalizan la magnitud de los pesos en la función objetivo), early stopping limita cuánto se ajustan los pesos, controlando capacidad efectiva.

Dataset / recursos

- sklearn.datasets.fetch_california_housing para regresión.
- sklearn.datasets.make_regression(n_samples=2000, n_features=50, noise=20) para experimentos controlados.
- Géron, Hands-On ML (3ª ed.), cap. 4 § "Early Stopping" (figura de la "U" en val loss).

Ejercicios

1. Curva clásica: entrenar SGDRegressor(max_iter=1, warm_start=True, learning_rate='constant', eta0=0.0005) por 500 épocas sobre California Housing escalado. Graficar RMSE de train y validación por época. Marcar la best epoch.
2. Early stopping automático: comparar SGDRegressor(early_stopping=True, validation_fraction=0.2, n_iter_no_change=10, tol=1e-4) contra el modelo sin early stopping. Reportar n.º de épocas reales (n_iter_) y RMSE en test.
3. Paciencia: barrer n_iter_no_change ∈ {1, 5, 20, 100} y mostrar cómo afecta la época final y el error de test.
4. Implementación manual: escribir un loop con partial_fit que mantenga best_loss, best_model = deepcopy(sgd) y un contador de paciencia. Devolver el mejor modelo.
5. Comparación con Ridge: sobre make_regression con ruido, comparar (a) SGD sin regularización + early stopping, (b) Ridge con alpha tuneado por CV. Discutir cuál generaliza mejor y por qué.

Homework verificable

Sobre California Housing (split 60/20/20 train/val/test, features escaladas con StandardScaler):

1. Entrenar un SGDRegressor con early_stopping=True, validation_fraction=0.2, n_iter_no_change=15, tol=1e-4, random_state=42.
2. Guardar la curva de loss_curve reconstruida con partial_fit (paralela, sin early stopping, 1000 épocas) en curva_val.png.
3. Reportar: épocas usadas por el modelo con early stopping (n_iter_), RMSE en test, y época óptima vista en la curva manual.

Criterio de aceptación: RMSE_test ≤ 0.75 (en variable target sin escalar, en cientos de miles de USD), y la época con early stopping debe estar dentro de ±10 % de la best epoch detectada manualmente.

Errores comunes

- Monitorear train loss en vez de val loss: train siempre baja; nunca dispara el corte. Hay que usar un split de validación interno.
- No guardar el snapshot del mejor modelo: si cortás en la época k después de patience épocas malas, los pesos finales no son los mejores. SGDRegressor lo hace por dentro; en implementación manual hay

que hacerlo explícito.
- Confundir validación con test: usar el test set para decidir el corte filtra información y arruina la estimación de generalización. El test se toca una sola vez al final.
- Patience demasiado baja: con ruido en val loss, n_iter_no_change=1 corta prematuro por fluctuaciones aleatorias. Subir a 5–20.
- Olvidar escalar features: SGD es muy sensible a la escala; sin StandardScaler el descenso oscila y la curva de validación no muestra la "U" clara.

Preguntas frecuentes

- ¿Early stopping reemplaza a Ridge/Lasso? No siempre. En modelos lineales con muchos features correlacionados, Ridge suele dar mejor sesgo-varianza. Early stopping brilla en modelos iterativos costosos (boosting, redes) donde tunear alpha por CV es caro.
- ¿Sirve para modelos cerrados como LinearRegression o Ridge con solver='cholesky'? No: esos resuelven en un paso, no hay "épocas" que detener. Aplica a algoritmos iterativos: SGD, gradient boosting (n_estimators con early_stopping_rounds), redes neuronales.
- ¿Cuánto debería ser validation_fraction? 10–20 % suele alcanzar. Con datasets chicos (<1000 filas) usar CV externa en lugar de un split interno fijo.
- ¿n_iter_no_change=5 es estándar? Es el default de sklearn y razonable. Subilo si la val loss es ruidosa o si la curva baja muy lento.
- ¿Por qué la val loss puede subir después de cierto punto? Porque el modelo empieza a memorizar ruido del train: train sigue bajando pero la capacidad efectiva se gastó en patrones espurios que no generalizan.

Referencias

- Géron, A. (2022). Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (3ª ed.), cap. 4, sección "Early Stopping".
- scikit-learn docs: SGDRegressor (parámetros early_stopping, validation_fraction, n_iter_no_change, tol).
- Prechelt, L. (1998). Early Stopping — But When? en Neural Networks: Tricks of the Trade.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 075 — Clase 075 — Regresión logística binaria y softmax

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 4. Duración estimada: 70 min.*

Objetivo

Que el alumno entienda la regresión logística como modelo lineal para clasificación: cómo la sigmoide convierte un score lineal en probabilidad, por qué se entrena minimizando log-loss (cross-entropy), y cómo se generaliza a multiclase con softmax. Además, que sepa diagnosticar si las probabilidades que devuelve un clasificador están bien calibradas y cómo corregirlas si no.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Derivar la sigmoide σ(z) = 1/(1+e^-z) como puente entre score lineal y probabilidad, y explicar por qué no se usa MSE en clasificación.
2. Entrenar LogisticRegression binaria de sklearn, interpretar coeficientes como log-odds y la frontera de decisión.
3. Extender a multiclase con multi_class='multinomial' (softmax) y diferenciar de 'ovr' (one-vs-rest).
4. Evaluar con log-loss y Brier score, no solo accuracy.
5. Diagnosticar y corregir calibración con calibration_curve y CalibratedClassifierCV (Platt / isotonic).

Temas

#	Tema	Por qué importa
1	Sigmoide y log-odds	Conecta regresión lineal con probabilidad
2	Log-loss (cross-entropy binaria)	Función de costo convexa, derivable, penal
3	Regularización (C, penalty)	sklearn regulariza por default — C=1/λ.
4	Softmax para multiclase	Generaliza sigmoide a K clases con probabi
5	multinomial vs ovr	El primero es softmax real; el segundo ent
6	Predict_proba y calibración	El score no siempre es probabilidad confia

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 067a — Calibración de probabilidades: Platt, isotonic, temperature scaling

Definiciones y características

Sigmoide σ(z)

: Función 1/(1+exp(-z)) que mapea $\mathbb{R} \to (0,1)$. Su inverso es el logit log(p/(1-p)). La regresión logística modela logit(p) = w·x + b (linealidad en los log-odds).

Log-loss (cross-entropy binaria)

: -[y·log(p) + (1-y)·log(1-p)]. Función de costo convexa de la logística. Penaliza fuerte la confianza alta cuando te equivocás (predecir 0.99 y que sea 0 cuesta ~4.6).

Softmax

: Generalización de la sigmoide a K clases: softmax(z_k) = exp(z_k) / Σ exp(z_j). Probabilidades positivas que suman 1. Es la activación final de logística multinomial y de la última capa en clasificadores neuronales.

Cross-entropy categórica

: Generalización del log-loss a K clases: -Σ_k y_k · log(p_k) con y one-hot. Pareja natural de softmax.

Calibración

: Propiedad de que P(y=1 | ŷ=p) ≈ p para todo p. Un modelo calibrado al 0.7 acierta como positivo el 70% de las veces que dice "0.7".

Reliability diagram

: Gráfico de fracción observada de positivos vs score promedio en bins. Diagonal = perfecto. Curva en S = típica de árboles; curva inversa = sobre-confianza.

Brier score

: mean((p - y)^2). MSE entre probabilidades y labels 0/1. Resume calibración + discriminación en un escalar. Menor = mejor.

CalibratedClassifierCV

: Wrapper de sklearn que envuelve un clasificador base y calibra sus probabilidades vía cross-validation interno. Métodos: 'sigmoid' (Platt) o 'isotonic'.

Dataset / recursos

- Iris (3 clases) para softmax — sklearn.datasets.load_iris().
- Breast cancer Wisconsin (binario) para logística y calibración — load_breast_cancer().
- Sintético desbalanceado con make_classification(weights=[0.9, 0.1]) para visualizar mis-calibración de un RandomForest.

Ejercicios

1. Logística binaria desde cero. Entrená LogisticRegression() sobre breast cancer. Reportá accuracy, log-loss y matriz de confusión. Imprimí los 5 coeficientes con mayor |w| e interpretá uno como odds-ratio (exp(w)).

2. Frontera de decisión. Con 2 features de iris (solo 2 clases primero), graficá la frontera lineal de la logística y los puntos. Cambiá C entre 0.01 y 100 y observá cómo la frontera se vuelve más/menos rígida.

3. Softmax sobre iris. LogisticRegression(multi_class='multinomial', solver='lbfgs'). Comparalo con multi_class='ovr' en log_loss y accuracy. Imprimí predict_proba de 3 muestras y verificá que sumen 1.

4. Reliability diagram de un RandomForest. Entrená RandomForestClassifier(n_estimators=100) sobre el dataset sintético desbalanceado. Computá calibration_curve con n_bins=10 y graficá vs diagonal. Reportá Brier score.

5. Calibrar con CalibratedClassifierCV. Sobre el mismo RF: aplicá method='sigmoid' y method='isotonic' (cv=5). Re-graficá los tres reliability diagrams (RF crudo, +Platt, +isotonic) y compará Brier scores. Reportá cuál calibra mejor y por qué te parece.

Homework verificable

Notebook que sobre make_classification(n_samples=20000, weights=[0.9, 0.1], random_state=42): (a) entrena LogisticRegression y RandomForestClassifier; (b) reporta accuracy, log-loss y Brier score de ambos en test; (c) grafica reliability diagram de ambos en la misma figura; (d) calibra el RF con CalibratedClassifierCV(method='isotonic', cv=5) y reporta el nuevo Brier; (e) escribe 2-3 líneas interpretando: ¿quedó el RF mejor calibrado que la logística cruda?

Criterio de aceptación: Brier score del RF calibrado debe ser menor que el del RF crudo. El reliability diagram del calibrado debe estar visiblemente más cerca de la diagonal.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Usar predict_proba directamente para tomar	Asumir que el score es probabilidad. Fix:
ConvergenceWarning: lbfgs failed to conver	Features sin escalar o max_iter bajo. Fix:
Coeficientes enormes con C muy alto en dat	Sin regularización, la logística diverge c
multi_class='ovr' y log-loss raro en multi	OvR entrena K binarios independientes — la
Calibrar sobre el mismo set de entrenamien	Leakage — los scores ya están sobreajustad

Preguntas frecuentes

¿Por qué log-loss y no MSE para clasificación?

Con MSE sobre una sigmoide la superficie de costo es no convexa (varios mínimos locales) y los gradientes se saturan en los extremos. Log-loss + sigmoide da costo convexo y gradientes limpios (p - y).

¿Platt o isotonic?

Platt si tenés ~1000 ejemplos de calibración y la curva de mis-calibración parece sigmoidea (típico SVM, NN pequeñas). Isotonic si tenés ~10k+ y/o la mis-calibración no es monótona-sigmoidea (RF, boosting). Regla práctica: probá ambos en validación y quedate con el de menor Brier.

¿predict_proba de LogisticRegression está calibrado?

Generalmente sí, porque la logística optimiza log-loss directamente — sus probabilidades suelen ser razonables. Igual chequealo con calibration_curve antes de usarlas para decisiones críticas; regularización fuerte (C chico) puede sub-confiar el output.

¿Softmax y sigmoide son lo mismo en binario?

Equivalentes: softmax con K=2 colapsa a sigmoide. sklearn con multi_class='multinomial' y 2 clases hace lo mismo que el modo binario por default.

¿Calibrar afecta el accuracy o solo las probabilidades?

Platt e isotonic son monótonas no-decrecientes → no cambian el ranking ni el argmax → accuracy y AUC quedan iguales. Lo que cambia es el log-loss, el Brier score, y el threshold óptimo cuando elegís uno distinto de 0.5.

Referencias

- Géron, cap. 4 § Logistic Regression y § Softmax Regression.
- sklearn — Probability calibration
- sklearn — LogisticRegression
- Niculescu-Mizil & Caruana (2005), Predicting Good Probabilities With Supervised Learning, ICML — paper canónico sobre Platt vs isotonic en RF, SVM, boosting, NB.
- Guo et al. (2017), On Calibration of Modern Neural Networks — temperature scaling.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 076 — Clase 076 — Calibración de probabilidades: Platt, isotonic, temperature scaling

> *Parte: 1 — Machine Learning Clásico · Fuente: Platt (1999) + Niculescu-Mizil & Caruana (2005) + Guo et al. (2017). Duración estimada: 75 min.*

Objetivo

Saber cuándo las probabilidades que devuelve predict_proba son calibradas — es decir, si el modelo dice "70 %" para un grupo, ¿realmente el 70 % es positivo? Modelos como Random Forest y SVM suelen estar mal calibrados; XGBoost mejor. Aplicar Platt scaling (sigmoid) y isotonic regression para corregir. Evaluar con

Brier score, ECE (Expected Calibration Error) y reliability diagrams.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Generar un reliability diagram: agrupar predicciones por bin de probabilidad y plotear "predicho vs real".
- Calcular Brier score = mean((p - y)²). Más bajo = mejor calibración.
- Calcular ECE: Σ (n_b/N) · |acc_b - conf_b|.
- Aplicar sklearn.calibration.CalibratedClassifierCV(estimator, method='sigmoid' | 'isotonic', cv=5).
- Decidir: Platt cuando muestra chica (n < 1000), isotonic cuando hay datos.

Temas

- ¿Por qué importa? Decisiones que dependen de threshold ≠ 0.5 requieren probs reales.
- Reliability diagram: predicción vs frecuencia real.
- Platt scaling: ajustar σ(A·logit + B) con MLE sobre val set.
- Isotonic regression: monótono pero más flexible (puede sobreajustar).
- Temperature scaling: solo divide logits por T aprendido — para multiclase, eficiente.
- Modelos típicamente calibrados (logistic regression) vs no (RF, SVM).

Definiciones y características

- Calibración: P(y=1 | p̂=p) = p para todo p.
- Reliability diagram: histograma de calibración.
- Brier score: (1/N) Σ (p_i - y_i)². Combina calibración + accuracy.
- ECE: weighted gap entre confidence y accuracy por bin.
- Platt: sigmoid-fit; parametric, 2 params (A, B). Bueno con n chico.
- Isotonic: monotonic non-parametric; flexible pero needs más data.
- Temperature scaling: softmax(z / T) con T learnable. Para multiclase.

Dataset / recursos

- fetch_openml('credit-g') o load_breast_cancer.
- Librerías: sklearn.calibration, matplotlib.

Ejercicios

1. Reliability diagram: entrenar RandomForest, generar predict_proba, bin en 10 grupos, plotear curva calibration vs y=x. Suele desviar.
2. Brier + ECE: implementar ambas y comparar entre RF (mala calibración) y LogReg (buena).
3. CalibratedClassifierCV: CalibratedClassifierCV(RF, method='sigmoid', cv=5).fit(X, y). Re-evaluar Brier.
4. Isotonic vs Platt: comparar ambos sobre el mismo modelo. Con n=10_000 ambos similares; con n=300 Platt mejor.
5. Threshold tuning post-calibración: con probs calibradas, F1 vs threshold es más interpretable.

Homework verificable

Sobre credit-g:

1. RandomForest + reliability diagram + Brier + ECE.
2. Calibrar con Platt y con isotonic (CV).
3. Comparar Brier/ECE pre y post.
4. Reliability diagrams superpuestos.

Criterio de aceptación: calibración reduce ECE en ≥ 30 %; reliability diagram post se acerca a la diagonal.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Calibrar sobre train (overfit)	Leak. Fix: usar CV o held-out calibration
Isotonic con < 500 ejemplos por clase	Overfit. Fix: Platt.
Reportar accuracy post-calibración como mé	No mide calibración. Fix: Brier o ECE.
Asumir LogReg está calibrado siempre	Con regularización fuerte o features extra
Calibración en multiclase con sigmoid	Sigmoid es binario. Fix: temperature scali

Preguntas frecuentes

¿Calibración cambia accuracy?

No (mucho). Reordena las probs pero no cambia el argmax típicamente. La accuracy queda igual; las probs son más interpretables.

¿Cuándo importa?

Cuando hacés decisiones threshold ≠ 0.5 (e.g., "alertar si P > 0.8"), reportes de "confianza" al usuario, ensemble por probabilidades.

¿RF tan mal calibrado?

Sí, hacia probs intermedias (0.2-0.8). Boosting (XGBoost) suele estar mejor.

¿DL necesita calibración?

Sí — DL modernos tienden a sobre-confiar. Temperature scaling (Guo 2017) es el estándar.

¿En producción cómo monitoreo calibración?

Logueá (prob_predicha, y_real). Cada N días, calculá Brier y ECE. Si drift, re-calibrar.

Referencias

- Platt (1999), Probabilistic Outputs for Support Vector Machines.
- Niculescu-Mizil & Caruana (2005), Predicting Good Probabilities With Supervised Learning, ICML.
- Guo et al. (2017), On Calibration of Modern Neural Networks, ICML.
- sklearn calibration docs.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 077 — Clase 077 — SVM lineal

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 5. Duración estimada: 60 min.*

Objetivo

Entender el principio de maximización del margen que define a las Support Vector Machines lineales, distinguir entre hard margin y soft margin, y entrenar un clasificador con LinearSVC controlando el trade-off

bias/varianza mediante el hiperparámetro C.

Resultados de aprendizaje

Al finalizar la clase, podrás:

- Explicar qué es el margen y por qué SVM busca el hiperplano que lo maximiza.
- Diferenciar hard margin (datos linealmente separables, sin tolerancia) de soft margin (admite violaciones).
- Interpretar el hiperparámetro C y su efecto sobre el ancho del margen y las violaciones permitidas.
- Entrenar un LinearSVC en scikit-learn con un Pipeline que incluya StandardScaler.
- Identificar los vectores soporte y entender por qué son los únicos puntos que definen la frontera.

Temas

- Intuición geométrica: hiperplano separador y margen.
- Hard margin: condiciones y limitaciones (sensibilidad a outliers, exige separabilidad).
- Soft margin: introducción de variables de holgura (slack).
- Hiperparámetro C: regularización, trade-off margen ancho vs. violaciones.
- Función de pérdida hinge loss.
- API de scikit-learn: LinearSVC, loss, C, dual, max_iter.
- Importancia crítica del escalado de features en SVM.

Definiciones y características

- Margen (margin): distancia perpendicular entre el hiperplano separador y los puntos más cercanos de cada clase. SVM busca maximizarlo.
- Vectores soporte (support vectors): instancias que tocan o violan el margen. Son los únicos puntos que determinan la frontera; el resto del dataset no influye.
- Hard margin classification: exige que todas las instancias estén del lado correcto del margen. Solo funciona si los datos son linealmente separables y es muy sensible a outliers.
- Soft margin classification: permite violaciones del margen (instancias dentro del margen o del lado equivocado) a cambio de un margen más ancho y robusto.
- Hiperparámetro C: controla cuánto se penalizan las violaciones del margen. C alto → poco tolerante, margen estrecho, riesgo de overfitting. C bajo → más tolerante, margen ancho, más regularización.
- LinearSVC: implementación eficiente de SVM lineal en scikit-learn, basada en liblinear. Escala mejor que SVC(kernel="linear") en datasets grandes.
- Hinge loss: función de pérdida que SVM minimiza, definida como max(0, 1 - t·y) donde t es la etiqueta (±1) e y la salida del modelo. Es cero cuando la predicción está del lado correcto y fuera del margen.
- Escalado: SVM es sensible a la escala de los features; sin StandardScaler el margen queda dominado por las variables de mayor rango.

Dataset / recursos

- Iris (sklearn.datasets.load_iris) — filtrado a dos clases (Iris-Virginica vs. resto) usando los features petal length y petal width. Es el ejemplo canónico del capítulo 5 de Géron.
- Opcional: dataset sintético con make_classification o make_blobs para visualizar el efecto de C y de outliers.

Ejercicios

1. Cargá Iris, quedate con dos features (petal length, petal width) y la clase Virginica como problema binario. Entrená un Pipeline([StandardScaler, LinearSVC(C=1, loss="hinge")]) y reportá accuracy sobre un split train/test.

2. Repetí el entrenamiento con C=0.1, C=1, C=100. Compará accuracy, número de vectores soporte estimados y ancho del margen. ¿Qué pasa en los extremos?
3. Graficá la frontera de decisión y el margen para los tres valores de C del ejercicio anterior (scatter de los datos + línea + márgenes punteados).
4. Agregá un outlier artificial a la clase minoritaria y volvé a entrenar con C alto y C bajo. Mostrá cómo C bajo absorbe mejor el outlier.
5. Compará tiempo de entrenamiento de LinearSVC vs. SVC(kernel="linear") sobre un dataset de ~10.000 muestras (make_classification). Confirmá que LinearSVC es más rápido.

Homework verificable

Entregá un script tarea_068.py que:

1. Cargue Iris binarizado (Virginica vs. resto) con petal length y petal width.
2. Construya un Pipeline con StandardScaler + LinearSVC(C=1, loss="hinge", random_state=42).
3. Haga train_test_split(test_size=0.2, random_state=42, stratify=y) y entrene.
4. Imprima accuracy sobre test y los coeficientes del clasificador (coef_, intercept_).

Criterio de aceptación: accuracy >= 0.93 sobre el test set y los coeficientes deben provenir del modelo escalado (no del crudo).

Errores comunes

1. No escalar features. SVM es sensible a la escala; sin StandardScaler el margen lo domina la variable de mayor rango y el modelo rinde mal.
2. Confundir la dirección de C. C alto = menos regularización (margen estrecho, más overfitting). C bajo = más regularización. Es inversa a alpha de Ridge/Lasso.
3. Usar SVC(kernel="linear") en datasets grandes. Es $O(m^2)$ a $O(m^3)$; preferí LinearSVC (basado en liblinear) que escala mejor.
4. Esperar predict_proba de LinearSVC. No lo soporta directamente. Usá CalibratedClassifierCV o decision_function si necesitás scores.
5. Olvidar loss="hinge". El default de LinearSVC es "squared_hinge", que no es la pérdida SVM canónica del libro.

Preguntas frecuentes

1. ¿C alto o bajo? Depende del problema. Empezá con C=1 y ajustá con GridSearchCV. Si hay overfitting, bajá C; si hay underfitting o el modelo es demasiado tolerante, subí C.
2. ¿Por qué se llaman "vectores soporte"? Porque son los únicos puntos que "sostienen" la frontera: si los movés, el hiperplano cambia. Los demás puntos no afectan al modelo.
3. ¿SVM lineal sirve si los datos no son linealmente separables? Con soft margin sí, tolera violaciones. Si la separación claramente no es lineal, pasá a kernels (clase 069).
4. ¿LinearSVC vs. LogisticRegression? Ambos producen fronteras lineales. SVM optimiza margen (hinge loss), LogReg optimiza log-likelihood. En la práctica suelen dar resultados similares; SVM tiende a ser más robusto a outliers cuando C es moderado.
5. ¿Funciona para multiclase? Sí, vía one-vs-rest automáticamente en LinearSVC.

Referencias

- Géron, A. Hands-On Machine Learning, 3ra ed., cap. 5 — "Support Vector Machines", sección "Linear SVM Classification".
- scikit-learn user guide: 1.4. Support Vector Machines.
- API: sklearn.svm.LinearSVC.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 078 — Clase 078 — SVM no lineal: kernel polinomial y RBF

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 5. Duración estimada: 70 min.*

Objetivo

Que el alumno entrene SVMs sobre datos no linealmente separables usando el kernel trick: en lugar de generar features polinómicas a mano (caro en memoria), SVC calcula el producto interno en el espacio expandido vía una función kernel. Foco en kernel polinomial y RBF (Gaussian), y cómo gamma y C controlan el bias-variance trade-off.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Explicar el kernel trick: por qué K(x, x') evita materializar el feature map φ(x).
2. Entrenar SVC(kernel='poly') y SVC(kernel='rbf') en datasets no lineales (moons, circles).
3. Tunear gamma y C con GridSearchCV entendiendo el efecto en la frontera.
4. Elegir kernel según geometría del problema (polinomial vs RBF vs lineal).
5. Reconocer el límite computacional de SVC: complejidad entre O(n²) y O(n³), no escala a >100k filas.

Temas

#	Tema	Por qué importa
1	Datos no linealmente separables (moons, ci	Motivación: lineal no alcanza.
2	Polynomial features manuales vs kernel tri	Kernel evita explosión combinatoria.
3	Kernel polinomial: degree, coef0, gamma	Captura interacciones de orden d.
4	Kernel RBF (Gaussian): gamma como inverso	Default robusto en sklearn.
5	C (regularización) × gamma (forma)	Grid 2D clásico.
6	Complejidad O(n²)–O(n³) y LinearSVC / SGD(	Cuándo NO usar SVC.

Definiciones y características

Kernel trick

: Truco matemático que reemplaza el producto interno φ(x)·φ(x') en el espacio expandido por una función K(x, x') calculable directamente en el espacio original. Permite trabajar en espacios de dimensión infinita (RBF) sin materializarlos. Requisito: K debe ser definida positiva (condición de Mercer).

Kernel RBF (Gaussian)

: K(x, x') = exp(-γ · ||x - x'||²). Mide similitud por distancia: 1 si x = x', → 0 cuando se alejan. Equivale a un feature map infinito-dimensional. Default razonable cuando no sabés qué kernel usar.

gamma (γ)

: Inverso del ancho del kernel RBF. γ alto → cada punto influye solo en su vecindad inmediata → frontera muy ondulada → overfitting. γ bajo → influencia amplia → frontera suave → underfitting. En sklearn: gamma='scale' (default) = 1 / (n_features · X.var()).

Kernel polinomial

: K(x, x') = (γ · x·x' + coef0)^degree. Captura interacciones hasta orden degree. coef0 controla el peso de los términos de orden alto vs bajo. Típicamente degree=2 o 3; más alto suele overfittear.

Kernel sigmoid

: K(x, x') = tanh(γ · x·x' + coef0). Rara vez supera a RBF en la práctica; se mantiene por razones históricas (similitud con redes neuronales).

Complejidad O(n²)–O(n³)

: SVC resuelve un QP cuadrático sobre los n ejemplos. En la práctica, libsvm escala entre O(n²) y O(n³) según la fracción de support vectors. Para >50k–100k filas: usar LinearSVC (si es lineal) o SGDClassifier(loss='hinge').

Condición de Mercer

: Para que una función K sea un kernel válido, su matriz de Gram $[K(x_i, x_j)]$ debe ser semidefinida positiva para cualquier conjunto finito de puntos. RBF, polinomial y lineal la cumplen; sigmoid solo bajo ciertos parámetros.

C (regularización)

: Inverso del parámetro de regularización L2. C alto → poca tolerancia a violaciones del margen → margen estrecho, posible overfit. C bajo → margen amplio, más violaciones permitidas, modelo más simple.

Dataset / recursos

- sklearn.datasets.make_moons(n_samples=500, noise=0.15) — clásico no lineal.
- sklearn.datasets.make_circles(n_samples=500, noise=0.1, factor=0.4) — radial puro, ideal para mostrar RBF.

Ejercicios

1. Lineal falla. Entrená SVC(kernel='linear') sobre make_moons. Reportá accuracy y graficá la frontera. Mostrá visualmente que es inadecuada.

2. Polynomial kernel. SVC(kernel='poly', degree=3, coef0=1, C=5) sobre moons. Comparar accuracy y forma de la frontera vs lineal.

3. RBF kernel. SVC(kernel='rbf', gamma=5, C=1) sobre circles. Variar gamma ∈ {0.1, 1, 10, 100} con el mismo C y graficar las 4 fronteras lado a lado.

4. Grid search 2D. GridSearchCV sobre gamma ∈ {0.01, 0.1, 1, 10} × C ∈ {0.1, 1, 10, 100} con cv=5 sobre moons. Reportar el mejor par y matriz de scores como heatmap.

5. Pipeline con StandardScaler. SVMs son sensibles a escala. Armá Pipeline([('sc', StandardScaler()), ('svc', SVC(kernel='rbf'))]) y comparar accuracy con y sin scaler en un dataset con features de magnitudes muy distintas.

Homework verificable

Notebook con make_moons(n_samples=1000, noise=0.2). Train/test split 80/20. Entrenar tres modelos: (a) SVC(kernel='linear'), (b) SVC(kernel='poly', degree=3), (c) SVC(kernel='rbf') con gamma y C tuneados vía GridSearchCV(cv=5). Reportar accuracy en test de los tres y graficar las tres fronteras de decisión en una

grilla 1×3.

Criterio de aceptación: El modelo RBF tuneado supera al lineal en al menos +15 puntos de accuracy. El notebook incluye el heatmap de GridSearchCV y las tres fronteras visualizadas.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Usar SVC en datasets >100k filas y que el	Complejidad O(n²)–O(n³). Fix: si el proble
Accuracy pésima sin escalar features	RBF y polinomial dependen de distancias. U
gamma alto da 100% en train y 60% en test	Overfitting clásico: γ alto = frontera ond
SVC(kernel='poly', degree=10) tarda eterni	Polinomios de alto grado explotan numérica
No setear random_state y resultados irrepr	SVC con probability=True o algunos solvers

Preguntas frecuentes

¿RBF o polynomial?

Si no sabés nada del problema: RBF (default robusto, un solo hiperparámetro relevante gamma). Polynomial conviene si tenés razones para creer que las interacciones son de orden bajo y fijo (ej. features físicas que se multiplican). RBF tiende a ganar en benchmarks tabulares.

¿Qué pasa si γ → 0 en RBF?

El kernel tiende a constante → todos los puntos parecen iguales → modelo equivale a clasificar por mayoría. Frontera trivial (clase mayoritaria en todo el espacio).

¿Cuántos support vectors es "muchos"?

Si len(svm.support_) > 0.5 · n_train, el modelo está overfitteando o C es muy alto. SVMs eficientes tienen pocos SVs (los del borde).

¿Por qué gamma='scale' y no 'auto'?

'scale' = 1 / (n_features · X.var()) — tiene en cuenta la varianza de los datos, robusto a escala. 'auto' = 1 / n_features (default viejo, deprecado conceptualmente). Usá 'scale' o un valor explícito tuneado.

¿Puedo usar SVM kernelizada con 1M de filas?

No con SVC. Alternativas: (a) LinearSVC si el problema es separable linealmente tras feature engineering; (b) Nystroem + LinearSVC para aproximar RBF en O(n); (c) SGDClassifier(loss='hinge'); (d) cambiar de modelo (Gradient Boosting suele ganar en tabular grande).

Referencias

- Géron, cap. 5 — Nonlinear SVM Classification y The Kernel Trick.
- sklearn — SVC
- sklearn — RBF kernel intuition
- sklearn — Kernel approximation (Nystroem)

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 079 — Clase 079 — SVM para regresión (SVR)

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 5. Duración estimada: 50 min.*

Objetivo

Aplicar Support Vector Machines al problema de regresión: entender el truco del epsilon-insensitive loss (ajustar dentro de un tubo de tolerancia en vez de minimizar el error puntual), entrenar LinearSVR y SVR con kernel, y elegir cuándo conviene SVR sobre regresión lineal clásica.

Resultados de aprendizaje

Al finalizar la clase, vas a poder:

- Explicar la lógica de SVR: maximizar el ancho del tubo ε mientras se contienen la mayoría de los puntos dentro.
- Entrenar un LinearSVR y un SVR(kernel="rbf") de scikit-learn con sus hiperparámetros (epsilon, C, gamma).
- Interpretar el efecto de epsilon (ancho del tubo) y C (penalización por puntos fuera del tubo) en el bias-variance trade-off.
- Comparar SVR vs LinearRegression / Ridge en un dataset con outliers.
- Justificar cuándo SVR escala mal (SVR es O(m²–m³)) y conviene LinearSVR u otro modelo.

Temas

1. De SVM clasificación a SVM regresión: invertir el objetivo.
2. Epsilon-insensitive loss: errores menores a ε no se penalizan.
3. LinearSVR: caso lineal, escala bien (O(m)).
4. SVR con kernel RBF: regresión no lineal, costo cuadrático.
5. Hiperparámetros clave: epsilon, C, gamma, kernel.
6. Robustez frente a outliers comparada con OLS.

Definiciones y características

- SVR (Support Vector Regression): algoritmo de regresión que busca una función que se desvíe a lo sumo ε de cada target, manteniéndola lo más "plana" posible (margen máximo).
- Epsilon-tube (tubo ε): banda de ancho 2ε alrededor de la predicción donde los errores no cuentan. Solo los puntos fuera del tubo aportan a la pérdida.
- LinearSVR: implementación lineal de SVR basada en liblinear. Escala linealmente con el tamaño del dataset; no soporta kernels.
- Kernel SVR: SVR de scikit-learn usa libsvm; soporta kernels (rbf, poly, sigmoid) para capturar no linealidad, pero su complejidad es entre O(m²) y O(m³).
- Hiperparámetro C: controla la penalización por puntos fuera del tubo. C chico → modelo más regularizado (tubo "flexible"); C grande → ajuste más estricto.
- Hiperparámetro epsilon: define el ancho del tubo de insensibilidad. ε grande → menos vectores de soporte, modelo más simple; ε chico → ajuste más fino, más vectores.
- Robust regression: SVR es menos sensible que OLS a outliers porque la pérdida es lineal fuera del tubo (no cuadrática).

Dataset / recursos

- California Housing (sklearn.datasets.fetch_california_housing) para regresión realista.
- Dataset sintético con outliers (make_regression + ruido pesado) para mostrar robustez.
- Géron, Hands-On ML (3ª ed.), cap. 5, sección "SVM Regression".

Ejercicios

1. Tubo ε en 2D. Generá y = 0.5x + ruido, entrená LinearSVR(epsilon=0.5) y graficá la recta junto al tubo ±ε. Marcá los vectores de soporte (puntos fuera del tubo).
2. Efecto de epsilon. Repetí el ejercicio 1 con ε ∈ {0.1, 0.5, 1.5}. ¿Cómo cambia la cantidad de vectores de soporte y el MSE en test?
3. Kernel RBF. En un dataset no lineal (y = sin(x) + ruido), compará LinearSVR vs SVR(kernel="rbf", gamma="scale"). Reportá MAE y graficá ambas curvas.
4. Grid search. Sobre California Housing, hacé GridSearchCV con SVR variando C ∈ {0.1, 1, 10} y gamma ∈ {"scale", 0.01, 0.1}. No uses más de 5 000 muestras (cuidado con el costo cuadrático).
5. Robustez vs OLS. Inyectá 5% de outliers en un dataset lineal y compará LinearRegression, Ridge y LinearSVR. ¿Cuál degrada menos?

Homework verificable

Entregar un script svr_california.py que:

1. Cargue California Housing y aplique StandardScaler (¡SVR exige escalado!).
2. Entrene LinearSVR(epsilon=0.5, C=1.0, random_state=42) y un SVR(kernel="rbf", C=10, gamma="scale") sobre un subset de 5 000 muestras.
3. Reporte RMSE en test para ambos modelos.

Criterio de aceptación: LinearSVR RMSE < 0.85 y SVR-RBF RMSE < 0.65 sobre el split de test (train_test_split(random_state=42, test_size=0.2)).

Errores comunes

- No escalar los features. SVR es extremadamente sensible a la escala: sin StandardScaler, los hiperparámetros pierden sentido y el entrenamiento no converge.
- Usar SVR con kernel sobre datasets grandes. Para más de ~10 000 muestras, SVR se vuelve impracticable. Usá LinearSVR o SGDRegressor(loss="epsilon_insensitive").
- Confundir epsilon de SVR con epsilon de optimizadores. Acá ε es el ancho del tubo de tolerancia, no una tolerancia numérica de convergencia.
- Dejar C en el default sin tunear. El default C=1.0 rara vez es óptimo; siempre validá con CV.
- Comparar tiempos sin dual="auto". En LinearSVR, el parámetro dual cambia drásticamente el tiempo según n_samples vs n_features.

Preguntas frecuentes

- ¿SVR vs Linear Regression cuándo conviene cuál? OLS minimiza el error cuadrático medio (sensible a outliers, óptimo si el ruido es gaussiano). SVR ignora errores chicos (dentro del tubo ε) y penaliza linealmente los grandes: más robusto a outliers y suele generalizar mejor con pocos datos ruidosos. Si tu dataset es limpio y grande, OLS/Ridge suele ser más simple y rápido.
- ¿Cómo elijo epsilon? Empezá con una fracción del desvío estándar del target (p. ej. ε ≈ 0.1·σ(y)) y ajustá con CV. Un ε muy chico convierte SVR en algo cercano a regresión cuadrática; uno muy grande genera un modelo casi constante.
- ¿SVR devuelve probabilidades o intervalos? No. SVR predice un valor puntual. Para intervalos de predicción usá quantile regression, conformal prediction o un modelo bayesiano.
- ¿Por qué SVR con kernel es tan lento? Calcula la matriz kernel (m × m) y resuelve un QP. La complejidad va de O(m²) a O(m³). Para datasets grandes, LinearSVR o Nystroem + LinearSVR (kernel approximation) son alternativas viables.
- ¿Conviene SVR para deep learning / problemas con millones de muestras? No. Para esas escalas, gradient boosting (XGBoost, LightGBM) o redes neuronales superan a SVR tanto en performance como

en costo.

Referencias

- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (3ª ed.), O'Reilly, cap. 5 — "SVM Regression".
- Smola, A. & Schölkopf, B. A Tutorial on Support Vector Regression (2004).
- scikit-learn docs: sklearn.svm.SVR y sklearn.svm.LinearSVR.
- scikit-learn user guide: Support Vector Machines — Regression.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 080 — Clase 080 — Árboles de decisión: entrenamiento, visualización, CART

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 6 § Training and Visualizing a Decision Tree.*
>
> *Duración estimada: 60 min.*

Objetivo

Que el alumno entrene un DecisionTreeClassifier con el algoritmo CART, entienda cómo se elige cada split (criterio Gini/Entropy), y sepa leer el árbol — tanto el dibujo (plot_tree) como el código (export_graphviz) — para auditar las decisiones del modelo.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Entrenar un DecisionTreeClassifier de scikit-learn sobre un dataset tabular (Iris) y predecir clases / probabilidades.
2. Explicar el algoritmo CART: greedy, binario, busca el par (feature, threshold) que minimiza la impureza ponderada de los dos hijos.
3. Calcular Gini y entropía a mano sobre un nodo con k clases y elegir el split óptimo entre candidatos.
4. Visualizar el árbol entrenado con plot_tree (matplotlib) y export_graphviz (DOT → PNG/SVG), interpretando samples, value, class, gini.
5. Identificar la decision boundary axis-aligned: cada split es ortogonal a un eje, lo que limita al árbol frente a fronteras oblicuas.

Temas

#	Tema	Por qué importa
1	DecisionTreeClassifier API	El estimador base del capítulo y de Random
2	Algoritmo CART	Saber qué hace el .fit() por debajo evita
3	Criterio Gini vs Entropy	Casi siempre dan el mismo árbol; saber cuá
4	Visualización: plot_tree y Graphviz	Auditar el modelo y comunicarlo a no-técni
5	Interpretación de nodos	Leer value=[n0, n1, n2], samples, class pr

6	Decision boundary axis-aligned	Limitación geométrica que explica por qué

Definiciones y características

CART (Classification And Regression Trees)

: Algoritmo que entrena árboles binarios (cada split tiene exactamente 2 hijos). Greedy: en cada nodo elige el par (feature k, threshold t_k) que minimiza el costo J = (m_left/m) · G_left + (m_right/m) · G_right. No vuelve atrás (no es óptimo global).

Impureza Gini

: G_i = 1 - Σ p_{i,k}² sobre las k clases del nodo i. Vale 0 si el nodo es puro (una sola clase) y máximo cuando las clases están balanceadas. Es el default en sklearn.

Entropía (information gain)

: H_i = - Σ p_{i,k} · log(p_{i,k}). Mide desorden en bits. Se activa con criterion='entropy'. En la práctica produce árboles muy similares a Gini; Gini es marginalmente más rápido.

Nodo

: Punto del árbol donde se evalúa una condición feature ≤ threshold. Si se cumple, va al hijo izquierdo; si no, al derecho.

Hoja (leaf)

: Nodo terminal, sin hijos. Predice la clase mayoritaria de las muestras de entrenamiento que cayeron en él. La probabilidad estimada es la proporción de cada clase en la hoja.

Profundidad (max_depth)

: Distancia desde la raíz hasta la hoja más lejana. Controla complejidad: sin tope, CART crece hasta que cada hoja es pura → overfitting casi garantizado. Se ataca en la Clase 072.

criterion

: Hiperparámetro de sklearn: 'gini' (default) o 'entropy'. Define la función de impureza que minimiza CART al elegir splits.

Decision boundary axis-aligned

: Cada split parte el espacio con un hiperplano ortogonal a un eje (x_k = t_k). La frontera resultante es una unión de rectángulos. Por eso un árbol no puede aprender una diagonal de forma compacta — necesita escaleras.

Dataset / recursos

- Iris (sklearn.datasets.load_iris): 150 muestras, 4 features, 3 clases. Géron lo usa porque permite dibujar el árbol completo en una página y la frontera en 2D (pétalo largo × ancho).
- Opcional: moons (make_moons) para ver la limitación axis-aligned.

Ejercicios

1. Fit + score baseline. Entrená DecisionTreeClassifier(max_depth=2, random_state=42) sobre Iris (todas las features). Reportá accuracy en train. Probabilidades de la primera flor con .predict_proba.

2. Gini a mano. Para el nodo raíz de Iris (50/50/50), calculá Gini. Verificá contra tree_.impurity[0] del estimador entrenado.

3. Gini vs Entropy. Entrená dos árboles max_depth=3, uno con cada criterio. Compará accuracy y el set de features usadas en los splits (tree_.feature). ¿Cambia algo material?

4. plot_tree. Renderizá el árbol del ejercicio 1 con sklearn.tree.plot_tree(clf, feature_names=..., class_names=..., filled=True). Identificá: feature del root split, threshold, y la clase predicha por cada hoja.

5. Boundary axis-aligned. Entrená un árbol max_depth=4 sobre make_moons(n_samples=300, noise=0.2). Graficá la decision boundary con un meshgrid. Observá los rectángulos.

Homework verificable

Notebook que: (a) carga Iris (solo pétalo largo y ancho), (b) entrena DecisionTreeClassifier(max_depth=2), (c) reporta accuracy y predict_proba para una flor nueva [5, 1.5], (d) exporta el árbol con export_graphviz a un archivo iris_tree.dot y lo convierte a PNG (o usa plot_tree), (e) responde por escrito: "¿por qué la hoja izquierda tiene Gini=0?".

Criterio de aceptación: accuracy ≥ 0.95 en train, el árbol exportado tiene exactamente 3 hojas (porque max_depth=2 con el split que CART elige sobre pétalos), y la respuesta menciona que esa hoja contiene solo setosa (pétalos chicos → linealmente separable).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
accuracy = 1.0 en train y mucho menos en t	No pusiste max_depth ni regularización. CA
graphviz: command not found al correr expo	El paquete Python graphviz no trae el bina
Cambio criterion='gini' ↔ 'entropy' y el á	Casi nunca pasa en datasets reales — si pa
Resultados distintos en cada .fit() con el	CART desempata splits con un componente al
plot_tree sale ilegible (texto pisado, min	Árbol grande sin figsize. Fix: plt.figure(

Preguntas frecuentes

¿Hay que escalar las features antes de un árbol?

No. CART compara feature ≤ threshold por feature individualmente — escalar/centrar no cambia el orden, así que el árbol es invariante a transformaciones monótonas de cada feature. Esto lo diferencia de SVM, KNN y regresión regularizada.

¿Gini o Entropy, cuál uso?

Gini (default). Es marginalmente más rápido (no calcula log) y produce árboles casi idénticos. Géron lo dice explícito: la elección rara vez importa en la práctica.

¿Por qué los árboles de sklearn son siempre binarios? ID3 hacía multi-way.

Porque sklearn implementa CART, no ID3/C4.5. CART es binario por diseño: un split por nodo, dos hijos. Es más simple, se generaliza a regresión sin cambios, y los multi-way se simulan encadenando binarios.

¿Los árboles manejan features categóricas?

sklearn no nativamente (hasta versión 1.4 hay soporte experimental). Hay que codificar: ordinal si hay orden natural, one-hot si no. Cuidado con one-hot de alta cardinalidad: el árbol nunca puede agrupar dos categorías en el mismo split, lo que sesga la selección hacia features numéricas.

¿Qué predice una hoja cuando empatan dos clases?

Devuelve la clase con índice menor entre las empatadas (orden de clf.classes_). En predict_proba te muestra el empate real ([0.5, 0.5, 0]), así que si te importa, usá probabilidades en vez de la clase argmax.

Referencias

- Géron, cap. 6 — Decision Trees, secciones Training and Visualizing y Making Predictions.
- sklearn — Decision Trees user guide
- DecisionTreeClassifier API
- Graphviz — para export_graphviz + dot.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 081 — Clase 081 — Regularización de árboles

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 6. Duración estimada: 50 min.*

Objetivo

Que el alumno controle el overfitting de un DecisionTreeClassifier/Regressor usando hiperparámetros de regularización (pre-pruning) y cost-complexity pruning (post-pruning), y sepa elegir entre ambas estrategias en función del problema.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Identificar overfitting en un árbol sin regularizar (train accuracy ≈ 1, test bajo).
2. Tunear max_depth, min_samples_split, min_samples_leaf, max_leaf_nodes, max_features con GridSearchCV.
3. Aplicar cost-complexity pruning vía ccp_alpha y leer la curva cost_complexity_pruning_path.
4. Diferenciar pre-pruning (frenar el crecimiento) de post-pruning (podar después).
5. Justificar la elección de hiperparámetros con curvas de validación, no a ojo.

Temas

#	Tema	Por qué importa
1	Árboles sin regularizar = overfitting gara	Un árbol crece hasta hojas puras.
2	max_depth y max_leaf_nodes	Los dos frenos más efectivos y baratos.
3	min_samples_split / min_samples_leaf	Frenan splits sobre muestras chicas (ruido
4	max_features	Aleatoriza el split; antesala de Random Fo
5	ccp_alpha (cost-complexity pruning)	Post-pruning principled, basado en α.
6	Pre-pruning vs post-pruning	Trade-off: rapidez vs óptimo bias-variance

Definiciones y características

max_depth

: Profundidad máxima del árbol. Default None (crece hasta hojas puras → overfit). Valor típico: 3–10. El hiperparámetro más impactante y barato de tunear.

min_samples_split

: Mínimo de muestras que un nodo necesita para ser candidato a split. Default 2 (cualquier nodo se parte).

Subirlo (ej. 20) bloquea splits sobre muestras chicas, que suelen ser ruido.

min_samples_leaf

: Mínimo de muestras que debe quedar en cada hoja resultante del split. Default 1 (hojas con 1 sample → memorización). Subirlo a 5–20 suaviza la frontera.

max_leaf_nodes

: Tope absoluto de hojas. El árbol crece en best-first (mejor reducción de impureza primero) hasta alcanzarlo. Equivalente funcional a max_depth pero más fino — el árbol no tiene que ser balanceado.

max_features

: Cantidad de features consideradas en cada split. Default = todas. Bajarlo (sqrt, log2) introduce aleatoriedad, reduce varianza y acelera el fit. Es la idea base de Random Forest.

ccp_alpha (cost-complexity pruning)

: Hiperparámetro $\alpha \geq 0$ que penaliza la complejidad del árbol al podar. Se minimiza $R(T) + \alpha \cdot |T|$ donde $|T|$ = nº de hojas. $\alpha=0$ no poda; α grande poda agresivo. Se elige con cost_complexity_pruning_path() + CV.

Pre-pruning

: Frenar el crecimiento durante el fit con max_depth, min_samples_*, max_leaf_nodes. Rápido y simple — es lo que más se usa.

Post-pruning

: Dejar crecer el árbol completo y después podarlo con ccp_alpha. Más fundamentado teóricamente, pero requiere doble pasada (fit completo + path + CV).

Dataset / recursos

moons de sklearn.datasets.make_moons(n_samples=1000, noise=0.4) — clásico para visualizar fronteras de decisión y el efecto de regularizar.

Ejercicios

1. Overfit baseline. Entrená un DecisionTreeClassifier() sin tocar nada sobre moons. Reportá train vs test accuracy. ¿Cuánto gap hay?

2. max_depth sweep. Para max_depth ∈ {1, 2, 3, 5, 10, None} graficá train y test accuracy. Identificá el punto donde empieza el overfit.

3. GridSearch. Buscá con GridSearchCV(cv=5) la mejor combinación de max_depth, min_samples_leaf y max_leaf_nodes. Reportá mejores params y test score.

4. Cost-complexity path. Llamá tree.cost_complexity_pruning_path(X_train, y_train) para obtener ccp_alphas. Entrená un árbol por cada α y graficá test accuracy vs α. Elegí el óptimo.

5. Visualización de fronteras. Plotteá la decision boundary de (a) árbol sin regularizar y (b) árbol con max_depth=4. Compará overfit visual.

Homework verificable

Notebook con moons (noise=0.4, 1000 samples, split 70/30): (a) baseline sin regularizar — reportar gap train/test; (b) GridSearchCV sobre max_depth y min_samples_leaf con 5-fold; (c) cost-complexity pruning — graficar ccp_alphas vs test accuracy y elegir el mejor α; (d) comparar test accuracy de los 3 modelos (baseline, grid, pruned) en tabla.

Criterio de aceptación: El modelo regularizado tiene gap train-test ≤ 5 pp y test accuracy ≥ baseline. La elección de α está justificada con la curva, no a ojo.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Train accuracy = 1.0, test = 0.75	Árbol sin regularizar memorizó train. Fix:
GridSearchCV tarda eternamente	Grilla enorme sobre 4+ hiperparámetros. Fi
ccp_alpha no hace nada	Default es 0.0 (sin pruning). Fix: corré c
min_samples_split=2 "no es default" — sí l	Confusión típica: 2 es el default y signif
Tuneás max_features=sqrt y los resultados	max_features < n_features introduce aleato

Preguntas frecuentes

¿Pre-pruning o post-pruning?

Pre-pruning (max_depth, min_samples_leaf) en el 90% de los casos: rápido, simple, suficiente. Post-pruning (ccp_alpha) cuando querés un árbol único, interpretable, óptimo bias-variance — típico en problemas tabulares chicos donde el árbol es el modelo final, no un building block. Para ensembles (RF, GBM), pre-pruning y listo.

¿max_depth o max_leaf_nodes?

max_depth es más fácil de interpretar ("profundidad ≤ 5"). max_leaf_nodes es más flexible — deja que el árbol sea asimétrico y crezca solo donde realmente reduce impureza. Si te importa interpretabilidad: max_depth. Si te importa performance: max_leaf_nodes.

¿Cómo elijo min_samples_leaf?

Regla de dedo: 1–5% de n_samples. Con 1000 filas, probá 10–50. Cuanto más ruidoso el dataset, más alto.

¿max_features regulariza?

Sí, indirectamente: al limitar features candidatas por split, fuerza al árbol a usar features sub-óptimas y reduce varianza. Es el mecanismo central de Random Forest, no tanto de árbol único.

¿Por qué ccp_alpha se usa poco en la práctica?

Porque (a) requiere doble pasada (path + CV), (b) en ensembles ya tenés regularización implícita por agregación, (c) max_depth + min_samples_leaf suele alcanzar. Pero es la podada con mejor fundamento teórico (Breiman et al., CART).

Referencias

- Géron, cap. 6 § "Regularization Hyperparameters".
- scikit-learn — Decision Trees user guide
- scikit-learn — Post pruning decision trees with cost complexity pruning
- Breiman, Friedman, Olshen & Stone (1984), Classification and Regression Trees — el paper original de CART y cost-complexity pruning.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 082 — Clase 082 — Regresión con árboles

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 6. Duración estimada: 45 min.*

Objetivo

Entrenar árboles de decisión para problemas de regresión con DecisionTreeRegressor, entender por qué sus predicciones son escalonadas (constantes por hoja) y reconocer su incapacidad para extrapolar fuera del rango de entrenamiento.

Resultados de aprendizaje

Al finalizar la clase, el estudiante podrá:

- Ajustar un DecisionTreeRegressor de scikit-learn y predecir sobre datos nuevos.
- Explicar cómo el criterio MSE (squared error) decide los splits en regresión.
- Visualizar la predicción escalonada del árbol contra el target real en 1D.
- Identificar el problema de no-extrapolación y contrastarlo con regresión lineal.
- Regular max_depth / min_samples_leaf para balancear bias y varianza.

Temas

- DecisionTreeRegressor: API e hiperparámetros principales.
- Criterio de split: squared_error (MSE) y friedman_mse.
- Predicción como media del target dentro de cada hoja → función escalonada.
- Sobreajuste con árboles profundos y regularización vía max_depth, min_samples_leaf, min_samples_split.
- Limitación clave: el árbol no extrapola — predice el último valor visto en los extremos.
- Comparación rápida con regresión lineal en datos con tendencia.

Definiciones y características

- DecisionTreeRegressor: variante de árbol que en cada hoja predice un escalar (la media del target de las muestras que cayeron ahí), no una clase.
- squared_error (MSE): criterio por defecto; en cada split elige el corte que minimiza la suma ponderada de varianzas en los hijos. Es el análogo de Gini/entropía pero para regresión.
- friedman_mse: variante de MSE que ajusta una mejora propuesta por Friedman; suele dar splits levemente distintos, útil con boosting.
- Predicción escalonada (step prediction): como cada hoja devuelve una constante, la función ŷ(x) es piecewise constant. En 1D se ve como una escalera, nunca como una curva suave.
- No extrapolación: fuera del rango [min(x_train), max(x_train)] el árbol devuelve la constante de la hoja del extremo. No hay pendiente, no hay tendencia: lineal con slope = 0 afuera.
- Regularización: max_depth limita profundidad, min_samples_leaf exige un mínimo de muestras por hoja (suaviza); sin estos límites el árbol llega a MSE = 0 en train (una hoja por muestra) y sobreajusta.
- Sensibilidad a rotaciones: igual que en clasificación, los splits son axis-aligned; si la relación es diagonal, el árbol necesita muchos cortes para aproximarla.

Dataset / recursos

- Dataset sintético 1D: x = np.linspace(-3, 3, 200), y = np.sin(x) + ruido_gaussiano(0, 0.1) — sirve para ver la escalera y la no-extrapolación.
- Alternativa real: sklearn.datasets.fetch_california_housing (regresión, 8 features) para evaluar con cross_val_score.
- Géron, Hands-On ML, cap. 6, sección "Regression" (incluye figura 6-5 con la predicción escalonada).

Ejercicios

1. Ajuste básico: generá los datos sin(x) + ruido, entrená DecisionTreeRegressor(max_depth=2) y max_depth=5, y graficá ambas predicciones sobre los puntos. Observá las escaleras.
2. MSE en train vs test: hacé train_test_split(test_size=0.3), calculá mean_squared_error en train y test para max_depth ∈ {1, 2, 4, 8, None}. ¿Dónde empieza el sobreajuste?
3. No extrapolación: predecí con el modelo entrenado sobre x_new = np.linspace(-5, 5, 200) (rango más ancho que el train). Graficá: vas a ver mesetas planas en los extremos.
4. Árbol vs lineal: entrená LinearRegression sobre los mismos datos. Comparalo con el árbol fuera del rango de entrenamiento. ¿Cuál extrapola "bien" y por qué?
5. California Housing: corré cross_val_score(DecisionTreeRegressor(max_depth=6), X, y, scoring='neg_mean_squared_error', cv=5) y compará contra max_depth=None.

Homework verificable

Entregar un script tarea_073.py que:

1. Genere y = 0.5 * x + sin(x) + ruido con x en [0, 10].
2. Entrene DecisionTreeRegressor(max_depth=4, random_state=42) y LinearRegression.
3. Prediga sobre x_new en [0, 15] (excede el rango de train).
4. Imprima el MSE en test (dentro del rango) y el valor predicho por el árbol en x=15.

Criterio de aceptación: el script corre sin error, el MSE del árbol en test es menor al de la lineal dentro del rango, y la predicción del árbol en x=15 es idéntica a su predicción en x=10 (demuestra no-extrapolación).

Errores comunes

- Esperar una curva suave: el árbol nunca produce una recta ni una curva — siempre escalones. Si necesitás suavidad, usá regresión lineal, polinómica o un ensamble como Gradient Boosting.
- Olvidar regularizar: dejar max_depth=None con pocos datos da MSE = 0 en train y MSE altísimo en test. Siempre validar con CV.
- Usar el árbol para forecasting con tendencia: si la serie tiene drift, el árbol queda clavado en el último valor del rango de train. Para tendencia, primero diferenciar o usar otro modelo.
- Confundir el criterio: criterion='gini' es para clasificación; en regresión usá squared_error o absolute_error (este último es más robusto a outliers pero más lento).
- Comparar MSE entre datasets: el MSE no es adimensional. Para comparar entre problemas usá R^2 o RMSE relativo al desvío del target.

Preguntas frecuentes

- ¿Árbol o regresión lineal? Si la relación es monótona y aproximadamente lineal y querés extrapolar, lineal. Si hay no-linealidades, interacciones y umbrales y trabajás dentro del rango de entrenamiento, árbol (o mejor: un ensamble).
- ¿Por qué la predicción es constante por tramos? Porque cada hoja almacena un único número (la media del target del subconjunto que cayó ahí). Toda muestra que termine en esa hoja recibe el mismo valor.
- ¿Puedo usar absolute_error en lugar de MSE? Sí, optimiza la mediana por hoja en vez de la media; es más robusto a outliers pero entrena más lento y suele dar splits distintos.
- ¿Cómo elijo max_depth? Con validación cruzada sobre una grilla pequeña ([2, 4, 6, 8, 10, None]). En general entre 4 y 10 alcanza para la mayoría de datasets tabulares.
- ¿Sirve un solo árbol en producción? Casi nunca. Su varianza es alta. En la práctica se usa como bloque base de Random Forest o Gradient Boosting (próximas clases).

Referencias

- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow, 3ra ed., cap. 6 — sección "Regression".
- scikit-learn: DecisionTreeRegressor.
- scikit-learn user guide: Decision Trees — Regression.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 083 — Clase 083 — Voting classifiers: hard y soft

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 7. Duración estimada: 45 min.*

Objetivo

Combinar varios modelos heterogéneos en un ensemble por votación y entender cuándo conviene hard voting (voto por mayoría) versus soft voting (promedio de probabilidades), apoyándonos en el principio de wisdom of the crowd.

Resultados de aprendizaje

Al finalizar la clase, el estudiante podrá:

- Explicar por qué un ensemble de clasificadores diversos suele superar al mejor modelo individual.
- Implementar un VotingClassifier de scikit-learn con voting="hard" y voting="soft".
- Decidir entre hard y soft voting según si los modelos base estiman probabilidades calibradas.
- Comparar la accuracy del ensemble contra la de cada modelo base en un dataset de clasificación.
- Identificar errores frecuentes al armar el ensemble (modelos correlacionados, probabilidades mal calibradas, pesos sin justificar).

Temas

- Wisdom of the crowd: por qué combinar predicciones reduce error.
- Hard voting: predicción final = clase con más votos entre los modelos base.
- Soft voting: predicción final = clase con mayor promedio de probabilidades estimadas.
- Requisitos para soft voting: que cada modelo exponga predict_proba y esté bien calibrado.
- Diversidad entre modelos base (algoritmos distintos, no sólo hiperparámetros distintos).
- VotingClassifier(estimators=[...], voting=..., weights=...).
- Limitaciones: si los modelos se equivocan en los mismos ejemplos, el ensemble no ayuda.

Definiciones y características

- Hard voting: cada clasificador del ensemble emite una predicción discreta y se elige la clase con más votos (moda). No requiere predict_proba.
- Soft voting: se promedian las probabilidades estimadas por cada modelo (opcionalmente ponderadas) y se elige la clase con mayor probabilidad promedio. Suele rendir mejor que hard voting cuando las probabilidades son confiables.
- VotingClassifier (sklearn.ensemble): meta-estimador que envuelve una lista de (nombre, estimador) y expone fit, predict, predict_proba (sólo si voting="soft").

- Weak learner / strong learner: un weak learner apenas supera el azar; al combinar muchos weak learners diversos se obtiene un strong learner. Ley de los grandes números aplicada a clasificadores independientes.
- Wisdom of the crowd: si los errores de los modelos son independientes, la probabilidad de que la mayoría se equivoque cae exponencialmente con el número de modelos.
- Diversidad: condición clave para que el ensemble funcione. Se logra usando algoritmos distintos (logística, SVM, árbol, kNN, etc.), no copias del mismo modelo con seeds distintos.
- Calibración: un modelo está calibrado si predict_proba ≈ frecuencia empírica. SVM con probability=True y árboles puros suelen estar mal calibrados, lo que degrada el soft voting.

Dataset / recursos

- Dataset sugerido: make_moons(n_samples=500, noise=0.30, random_state=42) (mismo que usa Géron en el cap. 7).
- Alternativa: load_breast_cancer() de sklearn.datasets para un caso real.
- Imports clave: LogisticRegression, RandomForestClassifier, SVC, VotingClassifier, train_test_split, accuracy_score.

Ejercicios

1. Cargar make_moons y partir en train/test (80/20, random_state=42). Entrenar por separado LogisticRegression, RandomForestClassifier y SVC(probability=True). Reportar accuracy de cada uno.
2. Armar un VotingClassifier con esos tres modelos y voting="hard". Comparar accuracy contra cada modelo base.
3. Repetir con voting="soft" (recordá SVC(probability=True)). ¿Mejora? ¿Por qué?
4. Probar weights=[1, 2, 1] favoreciendo al Random Forest. ¿Cómo cambia la performance? Justificar.
5. Reemplazar uno de los modelos por una copia casi idéntica de otro (ej.: dos regresiones logísticas con C muy parecido). Observar y explicar por qué el ensemble deja de ganar.

Homework verificable

Entregar un script voting.py que:

1. Cargue make_moons(n_samples=500, noise=0.30, random_state=42).
2. Entrene tres modelos base distintos y un VotingClassifier en modo hard y otro en modo soft.
3. Imprima la accuracy en test de los cinco (3 base + 2 ensembles).

Criterio de aceptación: el VotingClassifier en modo soft debe alcanzar accuracy ≥ que el mejor modelo base individual sobre el test split. Si no lo logra, justificar en un comentario qué modelo está rompiendo la diversidad o la calibración.

Errores comunes

- Usar voting="soft" con modelos no calibrados (ej.: SVC sin probability=True, o árboles muy profundos). Las probabilidades estimadas son ruido y el promedio empeora la predicción.
- Olvidar probability=True en SVC: sin eso, SVC no expone predict_proba y voting="soft" falla con AttributeError.
- Modelos base correlacionados (tres árboles, o tres regresiones logísticas con hiperparámetros parecidos): no hay diversidad, el ensemble no aporta nada.
- Pesos arbitrarios en weights= sin validación cruzada que los respalde. Mejor empezar con pesos iguales y ajustar con evidencia.
- No fijar random_state en los modelos estocásticos: los resultados no son reproducibles y dificultan la

comparación.

Preguntas frecuentes

- ¿Soft voting es siempre mejor que hard? No. Es mejor si los modelos base están bien calibrados. Con modelos mal calibrados, hard voting puede ganarle.
- ¿Cuántos modelos conviene meter? Géron muestra que con 3-5 modelos diversos ya hay ganancia notoria. Más allá, los rendimientos decrecen.
- ¿Puedo mezclar modelos lineales y no lineales? Sí, y es lo recomendado: la diversidad de sesgos inductivos es justo lo que hace funcionar al ensemble.
- ¿Diferencia con bagging? Bagging usa el mismo algoritmo entrenado en distintos bootstraps. Voting usa algoritmos distintos sobre el mismo dataset. Se cubre en la clase 075.
- ¿Y si un modelo base es mucho peor que los demás? Suele convenir sacarlo o bajarle el peso; arrastra al ensemble, sobre todo en hard voting.

Referencias

- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (3ª ed.), cap. 7 — sección "Voting Classifiers".
- scikit-learn — VotingClassifier.
- scikit-learn — Ensemble methods: voting classifier.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 084 — Clase 084 — Bagging y pasting

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 7. Duración estimada: 55 min.*

Objetivo

Que el alumno entrene ensembles entrenando el mismo algoritmo sobre distintos subsets del training set —bagging (con reemplazo) y pasting (sin reemplazo)— y evalúe sin held-out usando out-of-bag (OOB). Cierre con sampling de features (random patches y random subspaces) como puente conceptual a Random Forests.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Distinguir bagging vs pasting y justificar cuándo elegir uno u otro.
2. Entrenar un BaggingClassifier de scikit-learn con un base estimator y n_estimators razonable.
3. Usar oob_score=True para estimar el error de generalización sin tocar el test set.
4. Aplicar sampling de features (max_features < 1.0, bootstrap_features=True) para random patches / random subspaces.
5. Comparar bias/variance del ensemble vs un único árbol, en accuracy y en frontera de decisión.

Temas

#	Tema	Por qué importa
1	Bagging (bootstrap aggregating)	Reduce varianza promediando predictores en
2	Pasting	Igual idea pero sin reemplazo — útil cuand
3	BaggingClassifier API	estimator, n_estimators, max_samples, boot
4	OOB evaluation	Cada predictor ve ~63% de las instancias;
5	Random patches	Sampling de instancias y features.
6	Random subspaces	Sampling solo de features (bootstrap=False
7	Paralelización con n_jobs=-1	Los predictores son independientes — escal

Definiciones y características

Bagging (bootstrap aggregating)

: Entrenar el mismo algoritmo sobre múltiples muestras con reemplazo del training set y agregar predicciones por voto mayoritario (clasificación) o promedio (regresión). Reduce varianza sin aumentar bias.

Pasting

: Igual que bagging pero el sampling es sin reemplazo. Cada instancia aparece a lo sumo una vez por predictor. Bagging suele ganar porque introduce más diversidad.

Bootstrap

: Sampling con reemplazo del mismo tamaño que el original. En promedio cubre ~63.2% de las instancias únicas; el resto son repetidas.

Out-of-bag (OOB) score

: Para cada predictor, las instancias no muestreadas (~37%) actúan como validation set. Promediando OOB scores se obtiene una estimación del error de generalización sin separar test set. Activado con oob_score=True.

BaggingClassifier

: Meta-estimador de sklearn. Params clave: estimator (base learner), n_estimators (cuántos), max_samples (tamaño de cada muestra, default 1.0 = igual al training set), bootstrap=True (bagging) / False (pasting), oob_score, n_jobs.

Random patches

: Sampling simultáneo de instancias y features. bootstrap=True, bootstrap_features=True, max_features<1.0. Útil cuando hay muchas features (imágenes, alta dimensionalidad).

Random subspaces

: Sampling solo de features, manteniendo todas las instancias. bootstrap=False, max_samples=1.0, bootstrap_features=True, max_features<1.0.

n_estimators

: Número de predictores del ensemble. Más estimators = menos varianza, más cómputo. Típico: 100–500. La curva de mejora se aplana rápido.

Dataset / recursos

make_moons(n_samples=500, noise=0.30) de sklearn — frontera no lineal, ideal para visualizar el efecto smoothing del ensemble vs un árbol único.

Ejercicios

1. Bagging vs árbol único. Entrená un DecisionTreeClassifier y un BaggingClassifier(DecisionTreeClassifier(), n_estimators=500, max_samples=100, bootstrap=True) sobre make_moons. Compará accuracy en test.

2. Bagging vs pasting. Repetí (1) con bootstrap=False. Reportá diferencia de accuracy y discutí.

3. OOB. Entrená con oob_score=True, bootstrap=True. Imprimí bag.oob_score_ y compará con accuracy en test — deberían ser parecidos.

4. Curva de n_estimators. Variá n_estimators ∈ {1, 10, 50, 100, 500, 1000}. Plotteá accuracy test vs n_estimators.

5. Random subspaces. Sobre load_digits() (64 features), entrená con bootstrap=False, max_samples=1.0, bootstrap_features=True, max_features=0.5. Compará con bagging clásico.

Homework verificable

Notebook que: (a) cargue make_moons(500, noise=0.30), split 80/20; (b) entrene árbol único y BaggingClassifier(n_estimators=500, max_samples=100, oob_score=True, n_jobs=-1); (c) reporte accuracy test de ambos y oob_score_ del bag; (d) grafique las dos fronteras de decisión lado a lado; (e) repita con bootstrap=False (pasting) y compare en 1 párrafo.

Criterio de aceptación: Bagging supera al árbol único en test. oob_score_ ≈ accuracy test (diferencia < 0.05). La frontera del bag es visiblemente más suave.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ValueError: Out of bag estimation only ava	Pediste oob_score=True con bootstrap=False
n_estimators=10 y el ensemble no mejora al	Pocos predictores — la varianza no se prom
BaggingClassifier lento con árboles profun	Cada predictor crece sin podar. Fix: n_job
Confundir max_samples con max_features	max_samples sortea filas, max_features sor
OOB score muy distinto al test score	Dataset chico (OOB tiene alta varianza) o

Preguntas frecuentes

¿Bagging o pasting?

Bagging casi siempre. El reemplazo introduce más diversidad entre predictores, que es exactamente lo que reduce la varianza del ensemble. Pasting puede ganar marginalmente si el dataset es enorme y querés evitar repetir instancias, pero la diferencia es chica. Como bonus, bagging te da OOB gratis.

¿Cuántos n_estimators uso?

Empezá con 100. Subí a 500 si tenés cómputo. Más allá de eso, retornos decrecientes — la mejora marginal por estimator se aplana.

¿OOB reemplaza al test set?

Reemplaza al validation set durante tuning. Igual conviene reservar un test set final para reportar el número honesto al stakeholder.

¿Bagging sirve con cualquier base estimator?

Sí, pero brilla con learners de alta varianza (árboles profundos sin podar). Con learners de bajo varianza (regresión logística, naive Bayes) el bagging casi no mueve la aguja.

¿Esto es lo mismo que Random Forest?

Casi. Random Forest = bagging de árboles + sampling de features en cada split (no por árbol). Lo vemos en la clase siguiente.

Referencias

- Géron, Hands-On ML, cap. 7 § "Bagging and Pasting", "Out-of-Bag Evaluation", "Random Patches and Random Subspaces".
- sklearn BaggingClassifier
- sklearn user guide — Bagging
- Breiman, L. (1996). Bagging predictors. Machine Learning, 24(2).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 085 — Clase 085 — Random Forests y Extra Trees

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 7. Duración estimada: 60 min.*

Objetivo

Entender Random Forests y Extra Trees como ensambles de árboles decorrelacionados, diferenciar sus mecanismos de aleatoriedad y elegir hiperparámetros razonables para problemas reales de clasificación y regresión.

Resultados de aprendizaje

Al terminar la clase vas a poder:

1. Explicar por qué un Random Forest reduce varianza respecto a un árbol único, usando los conceptos de bagging y subsampling de features.
2. Diferenciar Random Forest vs Extra Trees (thresholds óptimos vs aleatorios) y argumentar cuándo conviene cada uno.
3. Configurar los hiperparámetros clave (n_estimators, max_features, max_depth, bootstrap, min_samples_leaf) con criterios fundados.
4. Entrenar y comparar RandomForestClassifier y ExtraTreesClassifier de scikit-learn en un dataset tabular, midiendo accuracy y tiempo.
5. Interpretar oob_score_ como estimación honesta del error de generalización sin necesidad de validación cruzada.

Temas

- Repaso de bagging: bootstrap + agregación → reducción de varianza.
- Random Forest: bagging de árboles + subsampling de features en cada split.
- Extra Trees (Extremely Randomized Trees): thresholds aleatorios en lugar de óptimos.
- Hiperparámetros: n_estimators, max_features, max_depth, min_samples_leaf, bootstrap, oob_score.
- Out-of-Bag (OOB) score y su uso como reemplazo de CV.
- Comparativa empírica: bias, varianza, tiempo de entrenamiento.

Definiciones y características

- Random Forest: ensamble de árboles de decisión entrenados sobre muestras bootstrap del dataset; en cada split solo se considera un subconjunto aleatorio de features. La predicción final es voto mayoritario (clasificación) o promedio (regresión).
- Extra Trees: variante de Random Forest donde, para cada feature candidata en un split, el threshold se elige al azar en lugar de buscar el óptimo. Esto añade más aleatoriedad → mayor reducción de varianza a costa de un poco más de sesgo, y es más rápido de entrenar.
- n_estimators: cantidad de árboles. Más árboles → mejor performance y mayor estabilidad, pero rendimiento decreciente. Valores típicos: 100–500.
- max_features: cantidad de features consideradas por split. Default sqrt(n_features) para clasificación, n_features (o 1.0) para regresión. Bajarlo aumenta decorrelación entre árboles.
- bootstrap: si True (default en RF), cada árbol se entrena sobre una muestra bootstrap; si False (default en ExtraTrees), sobre todo el dataset. Solo con bootstrap=True se puede calcular OOB.
- Feature subsampling: técnica clave que diferencia RF de un simple bagging de árboles; obliga a cada árbol a explorar diferentes combinaciones, decorrelacionando sus errores.
- Decorrelación: cuanto menos correlacionados estén los errores de los árboles, mayor la reducción de varianza del ensamble (el promedio de variables correlacionadas reduce varianza más lento que el de independientes).
- OOB score (oob_score_): precisión calculada sobre las muestras que no entraron en el bootstrap de cada árbol; estima generalización sin gastar un split de validación.

Dataset / recursos

- Dataset principal: sklearn.datasets.load_breast_cancer() (569 muestras, 30 features, binario).
- Dataset secundario (regresión): sklearn.datasets.fetch_california_housing().
- Librerías: scikit-learn >= 1.3, numpy, pandas, matplotlib.
- Notebook: notebook.ipynb con ejemplos guiados.

Ejercicios

1. Baseline: entrená un DecisionTreeClassifier único sobre load_breast_cancer con random_state=42 y reportá accuracy en test (split 80/20).
2. Random Forest: entrená un RandomForestClassifier(n_estimators=200, random_state=42) y compará accuracy y tiempo contra el árbol único.
3. Extra Trees: entrená un ExtraTreesClassifier(n_estimators=200, random_state=42) y compará contra RF en accuracy y tiempo de entrenamiento.
4. OOB: entrená un RF con oob_score=True, bootstrap=True y compará oob_score_ contra el accuracy de test; ¿se parecen?
5. Sensibilidad a max_features: variá max_features en [1, 'sqrt', 'log2', 0.5, 1.0] y graficá accuracy de CV; identificá el óptimo.

Homework verificable

Sobre fetch_california_housing:

1. Entrená RandomForestRegressor y ExtraTreesRegressor con n_estimators=300, random_state=42.
2. Reportá R^2 en test (split 80/20) y tiempo de entrenamiento de cada uno.
3. Tuneá max_features con GridSearchCV (cv=3) sobre [0.3, 0.5, 0.7, 1.0] para el mejor de los dos.

Criterio de aprobación: el mejor modelo tuneado alcanza $R^2 \geq 0.80$ en test, y entregás una tabla comparativa con accuracy, tiempo y max_features óptimo.

Errores comunes

- Confundir bootstrap=False con desactivar el ensamble: los árboles siguen siendo distintos por el subsampling de features y, en ExtraTrees, por los thresholds aleatorios.
- Pedir oob_score=True con bootstrap=False: scikit-learn lanza error; el OOB requiere muestras fuera del bootstrap.
- Inflar n_estimators sin medir: pasar de 500 a 2000 árboles rara vez mueve la aguja y multiplica tiempo/memoria. Medí la curva.
- No fijar random_state: hace que los resultados no sean reproducibles entre corridas; siempre fijalo en ejercicios y benchmarks.
- Usar max_features=1.0 en clasificación: anula el subsampling de features y degrada la decorrelación; quedate con 'sqrt' salvo evidencia en contra.

Preguntas frecuentes

- ¿RF o ExtraTrees? Si tenés mucho ruido en los features o el dataset es grande y el tiempo importa, probá ExtraTrees (más rápido, más reducción de varianza). Si querés OOB score o el dataset es chico/limpio, RF suele ser un poco mejor. En la práctica: probá los dos, la diferencia suele ser < 1%.
- ¿Cuántos árboles necesito? Empezá con 100, subí a 300–500 si ves margen. Más allá, ganancias marginales.
- ¿Random Forest sufre overfitting? Mucho menos que un árbol único, pero sí puede sobreajustar con datasets muy chicos o con árboles demasiado profundos sin regularización (min_samples_leaf).
- ¿Sirven para features categóricas? Sí, pero scikit-learn requiere encoding previo (OneHot o Ordinal). Otros frameworks (LightGBM, CatBoost) los manejan nativamente.
- ¿Por qué el OOB es una buena estimación? Cada muestra queda fuera de ~37% de los árboles (por la matemática del bootstrap), lo que da un estimador casi insesgado del error de generalización, gratis.

Referencias

- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (3ra ed.), cap. 7 — Ensemble Learning and Random Forests, sección "Random Forests".
- Breiman, L. (2001). Random Forests. Machine Learning, 45(1), 5–32.
- Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely Randomized Trees. Machine Learning, 63(1), 3–42.
- Documentación scikit-learn: RandomForestClassifier y ExtraTreesClassifier.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 086 — Clase 086 — Feature importance

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 7 + sklearn permutation_importance docs. Duración estimada: 70 min.*

Objetivo

Aprender a medir y comunicar qué variables aportan a un modelo basado en árboles, distinguiendo entre Mean Decrease in Impurity (MDI) —el feature_importances_ por default de scikit-learn— y permutation

importance, entendiendo los sesgos de cada método. Como complemento, conocer las herramientas modernas de interpretabilidad SHAP y LIME para explicar predicciones individuales.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Calcular feature_importances_ en un RandomForestClassifier / GradientBoostingRegressor y explicar qué mide MDI.
- Aplicar sklearn.inspection.permutation_importance sobre el set de validación y comparar el ranking con MDI.
- Identificar los sesgos de MDI (favorece features de alta cardinalidad y continuas) y de permutation (problemas con features correlacionadas).
- Generar explicaciones locales con shap.TreeExplainer e interpretar summary_plot y waterfall_plot.
- Decidir cuándo usar MDI, permutation, SHAP o LIME según contexto (auditoría, debugging, comunicación).

Temas

- Recordatorio: cómo los árboles eligen splits (impurity / variance reduction).
- MDI: suma ponderada de la reducción de impureza por feature, promediada sobre los árboles.
- Permutation importance: caída del score al permutar aleatoriamente los valores de una feature.
- Sesgos: cardinalidad alta infla MDI; correlación infla/desinfla permutation.
- MDI se calcula sobre train (riesgo de overfitting); permutation se calcula sobre test/valid.
- Interpretabilidad global vs local.
- Complemento moderno: SHAP, LIME, PDP, ICE.

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 077a — SHAP en profundidad: TreeExplainer, KernelExplainer, DeepExplainer

Definiciones y características

- feature_importances_: atributo de los estimadores tree-based de sklearn que devuelve un array normalizado (suma 1) con la importancia MDI de cada feature.
- MDI (Mean Decrease in Impurity): para cada feature, suma de las reducciones de impureza (Gini/entropy/MSE) en cada split que la usa, ponderadas por la fracción de muestras que pasan por ese nodo, promediado sobre todos los árboles del ensemble.
- Sesgo MDI por cardinalidad: features con muchos valores únicos (IDs, continuas) ofrecen más splits candidatos y suelen aparecer infladas aunque no tengan poder predictivo real.
- Permutation importance: diferencia entre el score del modelo con la columna original y el score tras permutar aleatoriamente esa columna. Se calcula sobre datos no vistos (test/valid).
- SHAP value: contribución de una feature a la predicción de una instancia específica, promediada sobre todos los órdenes posibles de inclusión (Shapley values de teoría de juegos).
- LIME: aproximación local con un modelo interpretable (regresión lineal/árbol pequeño) entrenado sobre perturbaciones de la instancia ponderadas por cercanía.
- PDP (Partial Dependence Plot): efecto marginal promedio de una o dos features sobre la predicción, marginalizando sobre el resto.
- ICE (Individual Conditional Expectation): como PDP pero una curva por instancia, revela heterogeneidad oculta por el promedio.

Dataset / recursos

- Dataset: sklearn.datasets.fetch_california_housing (regresión) y/o load_breast_cancer (clasificación).
- Librerías: scikit-learn, shap, lime, matplotlib.
- Instalación: pip install shap lime.

Ejercicios

1. Entrená un RandomForestRegressor sobre California Housing. Imprimí feature_importances_ ordenado y graficalo como barh.
2. Calculá permutation_importance sobre el set de test con n_repeats=10. Comparalo con MDI en un DataFrame lado a lado. ¿Coincide el top-3?
3. Agregá una columna random_id = np.arange(len(X)) y reentrená. Mostrá cómo MDI le asigna importancia espuria mientras permutation la ignora.
4. Generá shap.summary_plot con TreeExplainer y explicá la diferencia entre el plot tipo beeswarm (impacto + dirección) y un bar plot de MDI.
5. Elegí una instancia mal clasificada (o con error grande en regresión) y explicala con shap.waterfall_plot. Anotá las 3 features que más empujaron la predicción.

Homework verificable

Sobre el dataset load_breast_cancer, entrená un RandomForestClassifier y entregá un notebook que:

1. Reporte el top-5 de features según MDI y según permutation (sobre test).
2. Genere shap.summary_plot y guarde el PNG.
3. Explique con SHAP una predicción correcta y una incorrecta (waterfall_plot).

Criterio de aceptación: los rankings MDI y permutation pueden diferir, pero el alumno debe justificar la diferencia en 2-3 líneas mencionando cardinalidad/correlación. El SHAP debe mostrar que las contribuciones suman (aproximadamente) la diferencia entre expected_value y la predicción.

Errores comunes

1. Interpretar MDI con features de alta cardinalidad: IDs, fechas como int, o continuas con muchos decimales aparecen infladas. Siempre validá con permutation.
2. Calcular permutation importance sobre train: si el modelo overfittea, el ranking es inútil. Siempre sobre test/valid.
3. Confiar en permutation con features correlacionadas: si dos features llevan info redundante, permutar una sola subestima ambas (el modelo se apoya en la otra). Considerá agrupar features correlacionadas.
4. Usar KernelExplainer de SHAP sobre un Random Forest: lento e innecesario. Usá TreeExplainer, que es exacto y ~100x más rápido.
5. Reportar feature importance sin escalado/contexto: un valor de 0.15 no significa nada si no decís contra qué se compara. Mostrá ranking o porcentajes acumulados.

Preguntas frecuentes

1. ¿SHAP o LIME? SHAP si trabajás con tabular y querés rigor (especialmente con árboles, por TreeExplainer). LIME si necesitás explicar texto/imagen o un modelo donde SHAP es prohibitivamente lento.
2. ¿Por qué MDI suma 1 y permutation no? MDI está normalizado por construcción; permutation devuelve la caída absoluta de score, que depende de la métrica usada.
3. ¿Puedo usar feature importance para selección de features? Sí, pero con cuidado: usá permutation sobre validación, no MDI sobre train. Mejor aún: SelectFromModel o RFE con CV.

4. ¿Qué pasa si dos features están perfectamente correlacionadas? MDI las reparte arbitrariamente; permutation puede mostrar ambas como poco importantes. SHAP también distribuye entre ellas. Detectalas con corr() antes de entrenar.
5. ¿SHAP funciona con XGBoost/LightGBM/CatBoost? Sí, los tres tienen integración nativa con TreeExplainer. De hecho, XGBoost y LightGBM exponen pred_contribs que son SHAP values calculados internamente.

Referencias

- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow (3rd ed.), cap. 7 "Ensemble Learning and Random Forests" — sección "Feature Importance".
- scikit-learn docs: Permutation feature importance y partial_dependence.
- SHAP docs: <https://shap.readthedocs.io/> — Lundberg & Lee (2017), A Unified Approach to Interpreting Model Predictions, NeurIPS.
- LIME paper: Ribeiro, Singh & Guestrin (2016), "Why Should I Trust You?": Explaining the Predictions of Any Classifier, KDD. Repo: <https://github.com/marcotcr/lime>.
- Molnar, C. Interpretable Machine Learning (libro online gratuito): <https://christophm.github.io/interpretable-ml-book/>.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 087 — Clase 087 — SHAP en profundidad: TreeExplainer, KernelExplainer, DeepExplainer

> *Parte: 1 — Machine Learning Clásico · Fuente: Lundberg & Lee (2017) + shap docs. Duración estimada: 90 min.*

Objetivo

Dominar SHAP (SHapley Additive exPlanations) en profundidad: teoría de Shapley values (teoría de juegos cooperativos), TreeExplainer (rápido y exacto para árboles), KernelExplainer (model-agnostic, lento), DeepExplainer (para NN), y los plots clave: summary_plot, waterfall_plot, force_plot, dependence_plot, decision_plot.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar Shapley value intuitivamente: contribución marginal promedio sobre todos los órdenes de inclusión.
- Aplicar TreeExplainer en XGBoost/LightGBM/RF (segundos para millones de samples).
- Generar e interpretar los 5 plots SHAP principales.
- Diferenciar explicación global (summary_plot beeswarm) de local (waterfall de una predicción).
- Reconocer las limitaciones: SHAP asume cierta forma de "feature attribution" pero no es causal.

Temas

- Shapley value: φ_i = Σ |S|!(M-|S|-1)! / M! · [v(S ∪ {i}) - v(S)].

- 4 propiedades únicas: efficiency, symmetry, dummy, additivity.
- TreeExplainer: exacto para tree-based, O(TLD²).
- KernelExplainer: LIME-style con kernel especial → SHAP values aproximados.
- DeepExplainer: para Keras/PyTorch.
- Permutation explainer: alternativa moderna sin necesidad de árbol/red.

Definiciones y características

- Shapley value: única atribución que satisface las 4 propiedades.
- shap_values: matriz (n_samples, n_features) con contribución de cada feature a cada predicción.
- expected_value: predicción promedio del modelo (baseline). Σ shap_values[i] + expected_value ≈ predicción[i].
- summary_plot beeswarm: para cada feature, distribución de SHAP values; color = valor de la feature.
- waterfall_plot: para una predicción específica, contribución acumulada feature por feature.
- dependence_plot: feature en x, SHAP value en y; revela no-linearidades e interacciones.

Dataset / recursos

- fetch_california_housing o load_breast_cancer.
- Librerías: shap (pip install shap), xgboost, matplotlib.

Ejercicios

1. TreeExplainer: XGBoost en California Housing → explainer = shap.TreeExplainer(model); shap_values = explainer(X_test).
2. Summary plot: shap.summary_plot(shap_values, X_test). Identificar las 3 features más importantes y su dirección.
3. Waterfall: elegir 1 muestra concreta → shap.waterfall_plot(shap_values[0]). Sumar contribuciones y verificar que reconstruye la predicción.
4. Dependence plot: shap.dependence_plot('MedInc', shap_values.values, X_test). Detectar non-linearity.
5. Interaction values: shap_interaction = explainer.shap_interaction_values(X_test). Identificar par de features con mayor interacción.

Homework verificable

XGBoost sobre California Housing + análisis SHAP completo:

1. Modelo entrenado.
2. SHAP values con TreeExplainer.
3. Summary plot + force_plot de 3 predicciones (alta, media, baja).
4. Dependence plots para top-3 features.
5. Reporte de 1 página: insights de qué mueve el precio (en lenguaje no técnico).

Criterio de aceptación: el reporte identifica correctamente las features dominantes; las explicaciones individuales suman al valor del modelo (±0.01).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
KernelExplainer sobre tree → muy lento	Usa el equivocado. Fix: TreeExplainer.
Asumir SHAP es causal	No lo es — atribución es estadística, no c
Interpretar SHAP sobre train data	Sesgo por overfit. Fix: explicar sobre val
force_plot no renderiza en Jupyter	Sin shap.initjs(). Fix: ejecutar al inicio

Features muy correlacionadas → SHAP distri	Limitación inherente. Fix: agrupar correla

Preguntas frecuentes

SHAP vs LIME?

SHAP: fundamento teórico, exacto en tree-based, determinista. LIME: model-agnostic, rápido pero inestable. SHAP gana hoy.

¿explainer(X) o explainer.shap_values(X)?

API moderna (≥ 0.40): explainer(X) devuelve Explanation object. Compatible con todos los plots. Recomendado.

¿SHAP para clasificación multiclase?

shap_values es lista/tensor con valores por clase. Plot por clase con shap.summary_plot(shap_values[class_idx]).

¿SHAP en LLM?

Existe pero costoso por la combinatoria. Para LLMs se usan otras técnicas (attention rollout, integrated gradients).

¿En producción reporto SHAP por predicción?

Sí — para decisiones high-stake (crédito, medicina), el SHAP por predicción ayuda al human-in-the-loop a entender qué pasó.

Referencias

- Lundberg & Lee (2017), A Unified Approach to Interpreting Model Predictions, NeurIPS.
- Lundberg et al. (2020), From local explanations to global understanding with explainable AI for trees, Nature Machine Intelligence.
- SHAP docs.
- Molnar, C. Interpretable Machine Learning — cap. SHAP.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 088 — Clase 088 — Boosting: AdaBoost y Gradient Boosting

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 7. Duración estimada: 70 min.*

Objetivo

Entender la idea de boosting como combinación secuencial de aprendices débiles, dominar los dos enfoques clásicos —AdaBoost (reponderar errores) y Gradient Boosting (ajustar al residuo)— y saber tunear learning_rate, n_estimators y aplicar early stopping con staged_predict.

Resultados de aprendizaje

Al terminar la clase vas a poder:

1. Explicar la diferencia conceptual entre bagging (paralelo, varianza) y boosting (secuencial, sesgo).
2. Entrenar un AdaBoostClassifier y un GradientBoostingClassifier de scikit-learn, leyendo correctamente sus hiperparámetros.
3. Entender por qué learning_rate y n_estimators se mueven en sentidos opuestos (shrinkage).
4. Implementar early stopping en boosting usando staged_predict sobre un set de validación.
5. Decidir cuándo conviene AdaBoost, cuándo Gradient Boosting clásico y cuándo saltar directo a las librerías de la 079.

Temas

- Intuición: muchos clasificadores débiles → uno fuerte.
- AdaBoost: peso a las muestras mal clasificadas; SAMME y SAMME.R.
- Gradient Boosting: cada árbol ajusta el residuo (gradiente de la loss) del ensemble previo.
- Hiperparámetros clave: n_estimators, learning_rate, max_depth, subsample (stochastic gradient boosting).
- staged_predict / staged_predict_proba: predicciones intermedias para early stopping.
- Curvas de error vs. número de estimadores: detectar el "codo".
- Limitaciones: entrenamiento secuencial (no se paraleliza tan bien como bagging).

Definiciones y características

- Boosting: meta-algoritmo que entrena modelos en serie, donde cada uno corrige los errores del anterior. Reduce sesgo (a diferencia del bagging, que reduce varianza).
- AdaBoost (Adaptive Boosting): en cada iteración aumenta el peso de las muestras mal clasificadas y entrena un nuevo weak learner sobre la distribución reponderada. La predicción final es un voto ponderado.
- Gradient Boosting: en cada iteración entrena un árbol nuevo para predecir el residuo (gradiente negativo de la loss) del ensemble actual. Generaliza AdaBoost a cualquier función de pérdida diferenciable.
- learning_rate (η): factor de shrinkage que escala la contribución de cada árbol nuevo. Valores chicos (0.01–0.1) regularizan; obligan a usar más n_estimators.
- n_estimators: cantidad de aprendices débiles. En boosting puede haber sobreajuste si crece demasiado (a diferencia de Random Forest).
- Weak learner: modelo apenas mejor que el azar. En sklearn, por defecto, un árbol de profundidad 1 (decision stump) para AdaBoost y profundidad 3 para Gradient Boosting.
- Residuo: diferencia entre el target real y la predicción acumulada del ensemble; lo que falta por explicar.
- staged_predict: iterador que devuelve la predicción del ensemble usando 1, 2, … N estimadores. Sirve para graficar la curva de error y detectar el número óptimo.
- Shrinkage: técnica de regularización que multiplica cada contribución por learning_rate < 1; baja el riesgo de overfitting al costo de más iteraciones.

Dataset / recursos

- sklearn.datasets.make_moons(n_samples=500, noise=0.30, random_state=42) para la parte visual.
- sklearn.datasets.load_breast_cancer() para comparar AdaBoost vs. Gradient Boosting en un problema real.
- sklearn.ensemble.AdaBoostClassifier, GradientBoostingClassifier, GradientBoostingRegressor.
- Géron, cap. 7, sección "Boosting" (AdaBoost + Gradient Boosting + Early Stopping).

Ejercicios

1. AdaBoost desde cero conceptual: entrená un AdaBoostClassifier(n_estimators=200, learning_rate=0.5) sobre make_moons. Graficá la frontera de decisión con 1, 10, 50 y 200 estimadores.
2. Gradient Boosting paso a paso: usá GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0) sobre datos sintéticos y = x² + ruido. Mostrá las predicciones intermedias entrenando 3 árboles a mano sobre residuos sucesivos y verificá que coinciden con el ensemble.
3. Trade-off learning_rate ↔ n_estimators: comparé (lr=1.0, n=50) vs. (lr=0.1, n=500) vs. (lr=0.01, n=5000) en breast_cancer. Reportá accuracy y tiempo de fit.
4. Early stopping con staged_predict: entrené GradientBoostingClassifier(n_estimators=500, learning_rate=0.05) y, usando staged_predict sobre validación, encontrá el n_estimators óptimo. Re-entrené con ese valor.
5. Stochastic Gradient Boosting: agregale subsample=0.5 al modelo de (4). ¿Mejora la generalización? ¿Por qué?

Homework verificable

Sobre load_breast_cancer() con train_test_split(test_size=0.2, random_state=42, stratify=y):

1. Entrené AdaBoostClassifier(n_estimators=200, learning_rate=0.5, random_state=42) y GradientBoostingClassifier(n_estimators=200, learning_rate=0.1, max_depth=3, random_state=42).
2. Para el Gradient Boosting, encontrá el best_n usando staged_predict_proba sobre el set de test y log_loss.
3. Reportá accuracy de los tres modelos (AdaBoost, GB completo, GB con best_n).

Criterio de aprobación: accuracy ≥ 0.95 para el GB con early stopping y best_n < 200 (debe recortar de verdad).

Errores comunes

1. Subir n_estimators sin bajar learning_rate: termina sobreajustando feo. Si subís uno, bajá el otro.
2. Usar árboles profundos en AdaBoost: el algoritmo asume weak learners. Stumps (max_depth=1) suelen funcionar mejor que árboles grandes.
3. Comparar boosting con random forest a igualdad de n_estimators: no son comparables; en boosting cada árbol depende del anterior.
4. Olvidar random_state: tanto AdaBoost como Gradient Boosting son deterministas dado el seed, pero al cambiar la versión de sklearn los defaults se mueven.
5. No escalar… no hace falta: los métodos basados en árboles no necesitan escalado. El error inverso (escalar de más por las dudas) infla el pipeline sin beneficio.

Preguntas frecuentes

1. ¿AdaBoost o Gradient Boosting? En la práctica, Gradient Boosting gana casi siempre en tabular: es más flexible (cualquier loss), maneja mejor el ruido y tiene early stopping natural. AdaBoost queda como curiosidad histórica y para casos muy específicos con pocos features.
2. ¿learning_rate alto o bajo? Bajo (0.05–0.1) con muchos estimadores regulariza mejor y suele dar el mejor test score, a costa de tiempo de entrenamiento. Alto (0.5–1.0) entrena rápido pero sobreajusta.
3. ¿Cuántos n_estimators poner? Empezá con 500–1000 y dejá que early stopping lo recorte. No es como Random Forest, acá "más" no es siempre mejor.
4. ¿Por qué staged_predict y no validation_fraction directamente? GradientBoostingClassifier ya trae n_iter_no_change y validation_fraction desde sklearn 0.20; staged_predict te sirve para graficar la curva y entender qué pasa, no solo para parar.
5. ¿Es paralelizable? No de forma trivial: cada árbol depende del anterior. Por eso XGBoost / LightGBM

/ CatBoost (clase 079) usan trucos a nivel de construcción de cada árbol para acelerar.

Referencias

- Géron, Hands-On Machine Learning, 3ra ed., cap. 7 — "Ensemble Learning and Random Forests", sección Boosting.
- scikit-learn user guide: Ensemble methods → Boosting.
- Friedman, J. (2001). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics.
- Freund & Schapire (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 089 — Clase 089 — XGBoost, LightGBM y CatBoost

> *Parte: 1 — Machine Learning Clásico · Fuente: docs XGBoost/LightGBM/CatBoost + Géron, cap. 7. Duración estimada: 80 min.*

Objetivo

Conocer las tres librerías de gradient boosting moderno que dominan tabular ML —XGBoost, LightGBM y CatBoost—, entender sus diferencias algorítmicas (level-wise vs leaf-wise, manejo de categóricas, regularización) y saber elegir la adecuada según el problema, usando sus APIs sklearn-compatibles con early_stopping_rounds.

Resultados de aprendizaje

Al terminar la clase, vas a poder:

1. Instalar y usar xgboost, lightgbm y catboost con la API sklearn (fit/predict/predict_proba).
2. Explicar la diferencia entre crecimiento level-wise (XGBoost por defecto) y leaf-wise (LightGBM) y sus implicancias en velocidad y overfitting.
3. Configurar early stopping con eval_set y early_stopping_rounds para evitar overfitting y ahorrar tiempo.
4. Manejar features categóricas: encoding manual para XGBoost, categorical_feature en LightGBM, cat_features nativo en CatBoost con ordered boosting.
5. Elegir la librería apropiada según tamaño de dataset, presencia de categóricas y necesidad de velocidad de entrenamiento o inferencia.

Temas

1. Repaso: gradient boosting como ensemble secuencial (de clase 078).
2. XGBoost: histogram-based, level-wise, regularización L1+L2, sparse-aware.
3. LightGBM: leaf-wise growth, GOSS (Gradient-based One-Side Sampling), EFB (Exclusive Feature Bundling).
4. CatBoost: ordered boosting, manejo nativo de categóricas, symmetric trees.
5. Tabla comparativa de los 3.
6. Instalación: pip install xgboost lightgbm catboost.

7. API sklearn unificada: XGBClassifier, LGBMClassifier, CatBoostClassifier.
8. early_stopping_rounds con eval_set.
9. Categorical features en cada uno.
10. Cuándo cada uno: heurísticas prácticas.

Tabla comparativa

Aspecto	XGBoost	LightGBM	CatBoost
Crecimiento del árbol	Level-wise (default)	Leaf-wise	Symmetric (oblivious)
Velocidad entrenamiento	Media	Muy rápida	Media
Velocidad inferencia	Rápida	Rápida	Muy rápida
Categóricas nativas	(requiere encoding)	(índices, ordinal)	(ordered boosting)
Manejo de NaN			
Overfitting en datasets chicos	Bajo	Más riesgo (leaf-wise)	Bajo
Hiperparámetro clave	max_depth	num_leaves	depth
GPU			

Definiciones y características

- XGBoost (eXtreme Gradient Boosting): implementación optimizada de gradient boosting con regularización L1/L2 explícita, manejo de sparsity y construcción de árboles level-wise (todos los nodos de un nivel antes de bajar). Paper de Chen & Guestrin (2016).
- LightGBM: librería de Microsoft (2017) que crece árboles leaf-wise (expande la hoja con mayor pérdida), histogram-based, mucho más rápida en datasets grandes. Más propensa a overfit en datos chicos.
- CatBoost: librería de Yandex (2017) optimizada para features categóricas; usa ordered boosting para evitar target leakage y symmetric trees (todos los splits del mismo nivel usan la misma condición) que aceleran inferencia.
- Level-wise growth: estrategia de XGBoost; expande todos los nodos de un nivel antes de pasar al siguiente. Árbol balanceado, menos overfitting, más lento.
- Leaf-wise growth: estrategia de LightGBM; expande siempre la hoja con mayor reducción de pérdida. Árbol desbalanceado, más rápido, mayor riesgo de overfit si no se controla num_leaves.
- Histogram-based splitting: bucketea features continuas en max_bin histogramas (típicamente 255) para evaluar splits en O(bins) en lugar de O(n). Usado por los tres.
- GOSS (Gradient-based One-Side Sampling): técnica de LightGBM que retiene todas las instancias con gradiente grande y submuestrea aleatoriamente las de gradiente chico, acelerando sin perder precisión.
- EFB (Exclusive Feature Bundling): en LightGBM, agrupa features sparse mutuamente excluyentes en una sola, reduciendo dimensionalidad efectiva.
- Ordered boosting: técnica de CatBoost; para cada muestra usa un modelo entrenado sin esa muestra para calcular su residual, evitando el target leakage que ocurre al encodear categóricas con target mean usando la misma muestra.
- Target leakage en categóricas: cuando se codifica una categoría usando el target de las mismas filas que se entrenan, inflando artificialmente la performance en train; CatBoost lo evita con ordered TS (target statistics).

Dataset / recursos

- sklearn.datasets.fetch_openml('adult') o 'credit-g' — datasets con mezcla de numéricas y categóricas, ideales para comparar las 3 librerías.
- Alternativa: cualquier dataset tabular con >10k filas y columnas categóricas.

- Notebook: notebook.ipynb (no editar — se entrega).

Ejercicios

1. Instalación y smoke test: instalar las 3 librerías, importarlas e imprimir versiones. Confirmar que cargan sin error.
2. XGBoost básico: entrenar XGBClassifier sobre adult (con OneHotEncoder para categóricas), usar eval_set y early_stopping_rounds=20, reportar accuracy en test y la mejor iteración.
3. LightGBM con leaf-wise: entrenar LGBMClassifier pasando categorical_feature con los índices de columnas categóricas (encoding ordinal previo). Comparar tiempo de entrenamiento vs XGBoost.
4. CatBoost nativo: entrenar CatBoostClassifier pasando cat_features con los nombres de columnas — sin encoding manual. Verificar que la accuracy se mantiene o mejora respecto a 2 y 3.
5. Comparativa final: entrenar los 3 modelos sobre el mismo dataset con los mismos splits, reportar en una tabla: accuracy test, tiempo de fit, tiempo de predict y mejor iteración. Concluir cuál elegirías.

Homework verificable

Construí un script boosting_showdown.py que reciba --dataset <nombre openml> y entrene los tres modelos (XGBoost, LightGBM, CatBoost) con early_stopping_rounds=50, los mismos train/test_split(random_state=42), y emita un CSV resultados.csv con columnas: modelo, accuracy_test, tiempo_fit_seg, tiempo_predict_seg, mejor_iter.

Criterio de aprobado:

- Los 3 modelos entrenan sin error sobre al menos un dataset con categóricas.
- CatBoost se entrena sin OneHotEncoder previo (usa cat_features).
- El CSV contiene las 3 filas con valores no nulos.
- En el README del homework, una conclusión de 3 líneas justificando cuál elegirías.

Errores comunes

1. Olvidar early_stopping_rounds con eval_set: si pasás eval_set pero no early_stopping_rounds, entrenás las n_estimators completas sin parar — desperdiciás tiempo y podés overfittear.
2. Usar max_depth alto en LightGBM: como crece leaf-wise, num_leaves es el parámetro de control real; max_depth=-1 (sin límite) con num_leaves alto explota en overfit.
3. Pasar strings a XGBoost: XGBoost no maneja categóricas como strings por default (sí desde 1.5 con enable_categorical=True, pero limitado); olvidarse y obtener ValueError: could not convert string to float.
4. OneHotEncodear para CatBoost: anula su ventaja principal —el ordered boosting—; pasale las categóricas crudas con cat_features.
5. Comparar con n_estimators distintos: si XGBoost para en 200 y LightGBM en 800 por early stopping, comparar tiempos absolutos sin normalizar es injusto; reportá también iteraciones.

Preguntas frecuentes

1. ¿XGBoost, LightGBM o CatBoost?

- LightGBM si tenés dataset grande (>100k filas) y querés velocidad.
- CatBoost si tenés muchas categóricas de alta cardinalidad.
- XGBoost si necesitás máxima madurez del ecosistema, integración con Spark/Dask, o ya tenés pipeline montado.

1. ¿Cuál gana competencias de Kaggle? Históricamente XGBoost, hoy mezclado: LightGBM y CatBoost ganan terreno. Stacking de los tres suele rendir mejor que cualquiera solo.

1. ¿Son sklearn-compatibles? Sí — todos exponen XGBClassifier/LGBMClassifier/CatBoostClassifier

con fit/predict/predict_proba/score, usables en Pipeline, GridSearchCV, cross_val_score.

1. ¿Soportan GPU? Sí los tres. XGBoost con tree_method='gpu_hist', LightGBM con device='gpu', CatBoost con task_type='GPU'. Útil con datasets grandes.

1. ¿Cómo elijo num_leaves en LightGBM? Regla práctica: num_leaves < 2^max_depth. Empezá con 31 (default) y subí gradualmente monitoreando val loss.

Referencias

- XGBoost docs — <https://xgboost.readthedocs.io/>
- LightGBM docs — <https://lightgbm.readthedocs.io/>
- CatBoost docs — <https://catboost.ai/docs/>
- Chen, T. & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. KDD. <https://arxiv.org/abs/1603.02754>
- Ke, G. et al. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. NeurIPS.
- Prokhorenkova, L. et al. (2018). CatBoost: unbiased boosting with categorical features. NeurIPS. <https://arxiv.org/abs/1706.09516>
- Géron, A. Hands-On Machine Learning, cap. 7 (Ensemble Learning — mención a XGBoost).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 090 — Clase 090 — Stacking (stacked generalization)

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 7. Duración estimada: 60 min.*

Objetivo

Que el alumno combine modelos heterogéneos vía stacking: entrenar varios modelos base, generar predicciones out-of-fold, y entrenar un meta-modelo (blender) sobre esas predicciones — todo con StackingClassifier / StackingRegressor de sklearn, sin leakage.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Explicar la idea de stacking como ensamble donde un meta-modelo aprende a combinar las predicciones de los base learners.
2. Construir un StackingClassifier con varios estimadores base y un blender (típicamente regresión logística).
3. Justificar las predicciones out-of-fold (CV interna) como mecanismo para evitar que el blender vea predicciones in-sample.
4. Comparar stacking contra voting y contra un solo modelo bien tuneado, en accuracy y costo computacional.
5. Usar passthrough=True para que el blender vea también las features originales, no solo las predicciones de los base.

Temas

#	Tema	Por qué importa
1	Idea: ensamble de dos niveles	El meta-modelo aprende los errores correla
2	Base learners heterogéneos	Diversidad reduce varianza del ensamble.
3	Meta-modelo (blender)	Suele ser simple (logística, lineal) para
4	Out-of-fold predictions	Sin esto hay leakage: el blender ve scores
5	StackingClassifier / StackingRegressor	API sklearn que hace el CV interno por vos
6	passthrough	Concatenar features originales al input de
7	Costo y cuándo conviene	N entrenamientos × K folds — caro, no siem

Definiciones y características

Stacking (stacked generalization)

: Técnica de ensamble donde se entrenan M modelos base sobre el train set; sus predicciones (out-of-fold) se usan como features para entrenar un meta-modelo que aprende a combinarlas. Introducido por Wolpert (1992).

Base learners (nivel 0)

: Los modelos individuales del ensamble. Conviene que sean heterogéneos (p. ej. RandomForest + SVM + KNN) — si todos cometen los mismos errores, stackearlos no agrega información.

Meta-modelo / blender (nivel 1)

: Modelo que toma como input las predicciones de los base learners y produce la predicción final. Suele ser simple (LogisticRegression, Ridge) porque su input ya es altamente predictivo y un modelo complejo overfitearía.

StackingClassifier / StackingRegressor

: Implementación de sklearn (sklearn.ensemble). API: StackingClassifier(estimators=[(nombre, modelo), ...], final_estimator=Logistic(), cv=5). Internamente entrena los base con CV para generar predicciones out-of-fold, después reentrena cada base sobre todo el train, y entrena el blender sobre las predicciones OOF.

Out-of-fold predictions (OOF)

: Para cada fila del train, su predicción del base learner se genera cuando esa fila estuvo en el fold de validación (no en el de entrenamiento). Esto garantiza que el blender ve predicciones generadas por un modelo que no vio esa fila — evita el leakage.

cv en stacking

: Número de folds para generar las predicciones OOF. Default 5. Más folds → menos varianza en las features del blender, pero más costo (cada base se entrena cv veces).

passthrough=True

: Concatena las features originales X al input del blender, junto con las predicciones de los base. El blender puede así aprovechar tanto los meta-features como las features crudas. Útil cuando los base learners no capturan toda la señal.

Stacking vs voting

: Voting (hard/soft) promedia o vota las predicciones de los base con pesos fijos. Stacking aprende los pesos (y combinaciones no lineales) vía el blender. Más expresivo pero más caro y más propenso a overfit si el

blender es complejo.

Dataset / recursos

load_breast_cancer o make_classification(n_samples=2000) de sklearn — suficiente para mostrar el patrón sin tiempos largos de CV.

Ejercicios

1. Stacking básico. Construí un StackingClassifier con tres base learners (RandomForest, SVC con probability=True, KNN) y LogisticRegression como blender. Reportá accuracy con cross_val_score(cv=5).

2. Comparación contra base learners solos. Evaluá los tres base learners individualmente con el mismo CV. ¿El stacking supera al mejor solo? ¿Por cuánto?

3. passthrough=True. Repetí el ejercicio 1 con passthrough=True. ¿Mejora la accuracy? Pensá por qué (el blender ve features originales además de las predicciones).

4. Variando cv. Probá cv=3, cv=5, cv=10 en el stacking. Mirá accuracy y tiempo de entrenamiento. Trade-off típico.

5. Stacking vs Voting. Entrená un VotingClassifier(voting='soft') con los mismos tres base. Compará accuracy contra stacking. Discutí costo computacional.

Homework verificable

Notebook con make_classification(n_samples=3000, n_features=20, n_informative=10, random_state=42): (a) entrenar 3 base learners individuales y reportar CV-accuracy; (b) entrenar StackingClassifier con esos 3 + LogisticRegression blender; (c) repetir con passthrough=True; (d) entrenar VotingClassifier(voting='soft') con los mismos base; (e) tabla comparativa final con accuracy, std y tiempo de entrenamiento.

Criterio de aceptación: El stacking iguala o supera al mejor base learner individual. La tabla comparativa muestra los 5 modelos (3 base + stacking + voting) con accuracy media ± std y tiempo.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
El stacking da peor que el mejor base lear	Base learners poco diversos (todos árboles
Blender entrenado sobre las mismas predicc	Pasaste predicciones in-sample al blender
SVC no se puede usar en StackingClassifier	SVC por default no expone probas. Fix: SVC
Stacking tarda 10× más que un solo modelo	Es esperable: M base × K folds + 1 blender
passthrough=True empeora el resultado	El blender se confunde con muchas features

Preguntas frecuentes

¿Stacking vs voting — cuándo cuál?

Voting si querés algo rápido, simple y los base learners son razonablemente parejos: promedia probas (soft) o vota clases (hard) con pesos fijos. Stacking si tenés tiempo de CV y los base learners cometen errores distintos — el blender aprende a explotar esa diversidad. Stacking suele ganar ~1-3 pp de accuracy a costa de 5-10× más cómputo.

¿Qué modelo conviene como blender?

Algo simple y regularizado: LogisticRegression (clasificación) o Ridge (regresión) son la elección estándar. Modelos complejos (RandomForest, XGBoost) como blender suelen overfitear las M predicciones OOF.

¿Cuántos base learners?

3-5 está bien. Más allá, la ganancia marginal cae rápido y el costo escala lineal. Lo importante es que sean heterogéneos (familias distintas), no que sean muchos.

¿Puedo apilar stackings (multinivel)?

Técnicamente sí (Kaggle lo hace), pero la ganancia marginal vs costo es brutal y el riesgo de overfit explota. Para producción: un solo nivel de stacking es el sweet spot.

¿Sirve stacking si tengo poca data?

Mal. Con N chico, las predicciones OOF tienen mucha varianza y el blender no encuentra señal estable. Bajo ese régimen, un modelo solo bien tuneado o un VotingClassifier suelen ganarle.

Referencias

- Géron, cap. 7 § Stacking.
- Wolpert, D. (1992). Stacked Generalization. Neural Networks 5(2).
- sklearn StackingClassifier
- sklearn StackingRegressor
- sklearn user guide — Stacked generalization

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 091 — Clase 091 — La maldición de la dimensionalidad

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 8 § The Curse of Dimensionality.*
>
> *Duración estimada: 45 min.*

Objetivo

Que el alumno entienda por qué los algoritmos basados en distancia (kNN, k-means, SVM-RBF) degradan en alta dimensión: el espacio se vuelve mayormente vacío, las distancias entre puntos colapsan a un mismo valor, y los modelos overfittean. Esto motiva la reducción de dimensionalidad (PCA, manifold learning) que viene en las próximas clases.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Explicar la sparsity exponencial: por qué llenar uniformemente un hipercubo unitario requiere n^d puntos.
2. Calcular numéricamente que en d=100 la razón (d_max - d_min) / d_min entre distancias euclidianas tiende a 0.
3. Identificar qué algoritmos sufren la maldición (basados en distancia/densidad) y cuáles menos (árboles, modelos lineales con regularización).
4. Reconocer la manifold hypothesis como justificación de PCA, t-SNE, UMAP: los datos reales viven en

un subespacio de baja dimensión.

5. Decidir cuándo reducir dimensionalidad vs. cuándo regularizar o usar otro modelo.

Temas

#	Tema	Por qué importa
1	Intuición geométrica: hipercubo y volumen	En d=100, el 99.99% del volumen está pegad
2	Sparsity exponencial	Para cubrir el espacio uniformemente, los
3	Concentración de la medida	Distancias entre puntos aleatorios converg
4	Distancia euclidiana pierde sentido	nearest neighbor deja de ser informativo.
5	Hubness	Algunos puntos aparecen como vecinos de to
6	Manifold hypothesis	Los datos reales no llenan el espacio: viv
7	Implicaciones prácticas para ML	Overfitting, kNN colapsa, k-means inestabl

Definiciones y características

Maldición de la dimensionalidad

: Conjunto de fenómenos contraintuitivos que aparecen al crecer la cantidad de features: el espacio se vuelve mayormente vacío, las distancias se igualan, y la cantidad de datos necesaria para densidad constante crece exponencialmente en d. Acuñada por Bellman (1961).

Sparsity exponencial

: Para mantener una densidad constante de puntos en un hipercubo [0,1]^d, hace falta n^d muestras. Con d=100 y n=10 por eje son 10^100 puntos — más que átomos en el universo observable. Consecuencia: en alta dimensión todos los datasets son chicos.

Concentración de la medida

: En distribuciones de alta dimensión (uniforme, gaussiana), la distancia entre dos puntos aleatorios se concentra fuertemente alrededor de su media. Formalmente, Var(||X-Y||) / E[||X-Y||]^2 → 0 cuando d → ∞. Resultado: d_min ≈ d_max, y la noción de "más cercano" se vuelve ruido.

Distancia euclidiana en alta dimensión

: Pierde poder discriminativo porque acumula varianza por cada feature irrelevante. Alternativas: distancia coseno (escala-invariante), Mahalanobis (decorrela), o reducir antes con PCA.

Hubness

: Fenómeno por el cual, en alta dimensión, ciertos puntos aparecen entre los k vecinos más cercanos de muchísimos otros (hubs), mientras otros nunca son vecinos de nadie (anti-hubs). Rompe el supuesto de simetría que asume kNN.

Manifold hypothesis

: Suposición de que los datos reales de alta dimensión (imágenes, texto, audio) no llenan el espacio ambiente sino que viven en un subespacio (manifold) de dimensión intrínseca mucho menor. Justifica PCA, autoencoders, t-SNE, UMAP. Sin esta hipótesis, reducir dimensionalidad sería destruir información.

Dimensión intrínseca

: Cantidad mínima de variables latentes necesarias para representar los datos sin pérdida apreciable. MNIST tiene d=784 features (28×28 pixels) pero dimensión intrínseca estimada ~10–15.

Dataset / recursos

Sintético: puntos uniformes en [0,1]^d para d {2, 10, 100, 1000}. Permite calcular numéricamente sparsity,

ratio d_max/d_min, y fracción de volumen cerca del borde. Para la parte aplicada, sklearn.datasets.load_digits (d=64) como ejemplo de manifold hypothesis empírica.

Ejercicios

1. Volumen del borde. Calculá la fracción de volumen del hipercubo [0,1]^d que está a menos de 0.01 del borde, para d=2, 10, 100. Fórmula: 1 - 0.98^d.

2. Concentración de distancias. Sampleá 1000 puntos uniformes en [0,1]^d. Para d {2, 10, 100, 1000}, calculá (d_max - d_min) / d_min sobre todas las distancias pairwise. Verificá que tiende a 0.

3. Distancia al vecino más cercano. Con n=1000 puntos uniformes y d variable, graficá la distancia media al 1-NN. Mostrá que crece con d (el "vecino" está cada vez más lejos).

4. kNN degrada. Generá clasificación sintética con n=500, agregando d features de ruido puro (irrelevantes). Evaluá accuracy de KNeighborsClassifier para d = 2, 10, 50, 200. Curva debería caer.

5. Manifold hipótesis empírica. Cargá sklearn.datasets.load_digits. Calculá cuántos componentes PCA explican el 95% de la varianza vs. d=64 originales — estimación grosera de dimensión intrínseca.

Homework verificable

Notebook que: (a) genere puntos uniformes en [0,1]^d para d [1, 200]; (b) calcule y grafique (d_max - d_min)/d_min vs d; (c) entrene KNeighborsClassifier con n=1000 y features añadidas de ruido N(0,1), reporte accuracy vs d; (d) sobre load_digits, calcule cuántos componentes PCA explican 90%, 95%, 99% de varianza.

Criterio de aceptación: El ratio de distancias tiende a 0 para d > 50. Accuracy de kNN cae monotónicamente al agregar ruido. PCA muestra que digits (d=64) tiene dimensión intrínseca ≤ 30 al 95%.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
"Agregué más features y el modelo empeoró"	Features irrelevantes inyectan ruido y emp
kNN con k=1 da resultados aleatorios en al	Concentración de medida: d_min ≈ d_max, "e
"PCA me bajó accuracy"	Te quedaste con muy pocos componentes, o e
Confundir d (features) con n (muestras)	"Tengo 10M filas, no aplica la maldición"
Pensar que árboles también sufren	Random Forest y boosting son mucho más rob

Preguntas frecuentes

¿A partir de qué d empiezo a preocuparme?

Heurística: con kNN/k-means/SVM-RBF, ya con d > 20–30 y n modesto se nota la degradación. Con árboles y modelos lineales regularizados, podés escalar a d = 10000 sin drama (texto TF-IDF, genómica).

¿Más datos resuelve la maldición?

Solo asintóticamente, y la cantidad necesaria crece exponencialmente. Para d=100 no hay suficientes datos en el universo. La salida real es reducir d (PCA, feature selection) o usar modelos que no dependan de densidad global.

¿Por qué deep learning funciona con d enorme (imágenes 224×224×3 = 150k)?

Por la manifold hypothesis: las imágenes naturales no llenan R^150000, viven en un manifold de dimensión intrínseca mucho menor. Las redes profundas aprenden esa estructura jerárquicamente. La maldición sigue

aplicando si hacés kNN sobre pixels crudos — y por eso no se hace.

¿Distancia coseno me salva?

Mitiga, no salva. Es invariante a escala y funciona mejor en texto (TF-IDF, embeddings), pero también sufre concentración si las features son ruido puro. Mejor combinar coseno + reducción + selección.

¿Cómo estimo la dimensión intrínseca?

Métodos: (a) PCA + curva de varianza explicada (rápido, lineal); (b) sklearn.manifold (Isomap, LLE); (c) estimadores específicos como MLE de Levina–Bickel o el paquete skdim. Para empezar, PCA al 95% alcanza como aproximación grosera.

Referencias

- Géron, cap. 8 § The Curse of Dimensionality y § Main Approaches for Dimensionality Reduction.
- Bellman, R. (1961). Adaptive Control Processes: A Guided Tour — origen del término.
- Beyer, K. et al. (1999). When Is "Nearest Neighbor" Meaningful? — paper clásico sobre concentración de distancias.
- Radovanović, M. (2010). Hubs in space: Popular nearest neighbors in high-dimensional data — fenómeno de hubness.
- scikit-learn — Manifold learning

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 092 — Clase 092 — PCA: proyección, varianza explicada, incremental, randomized, kernel

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 8. Duración estimada: 80 min.*

Objetivo

Dominar PCA como técnica de reducción de dimensionalidad lineal: entender la proyección al subespacio de máxima varianza vía SVD, elegir el número de componentes con explained_variance_ratio_, y aplicar las variantes (Incremental, Randomized, Kernel) según el tamaño y la geometría del dataset.

Resultados de aprendizaje

Al finalizar la clase, el estudiante podrá:

- Explicar geométricamente qué hace PCA (proyección al hiperplano que maximiza la varianza).
- Ajustar PCA de scikit-learn y leer components_, explained_variance_ratio_ y singular_values_.
- Elegir el número de componentes vía umbral de varianza acumulada (ej. 95%) o codo del scree plot.
- Decidir entre PCA, IncrementalPCA y PCA(svd_solver="randomized") según memoria y velocidad.
- Aplicar KernelPCA (RBF/poly) para datasets con estructura no-lineal y tunear gamma con GridSearchCV.

Temas

1. Maldición de la dimensionalidad y motivación de PCA.

2. Proyección al subespacio de máxima varianza: intuición geométrica.
3. SVD como motor algebraico de PCA.
4. Varianza explicada y elección de n_components (entero, ratio, "mle").
5. IncrementalPCA para datasets que no entran en RAM (partial_fit).
6. Randomized PCA (svd_solver="randomized") para acelerar en alta dimensión.
7. KernelPCA con kernels RBF, polinómico y sigmoide; tuning de gamma.
8. Pipeline completo: StandardScaler → PCA → modelo.

Definiciones y características

- PCA (Principal Component Analysis): técnica lineal que proyecta los datos sobre los ejes ortogonales que capturan máxima varianza.
- Componentes principales: vectores ortonormales (filas de components_) que definen el nuevo sistema de coordenadas; el primero apunta a la dirección de máxima varianza, el segundo a la siguiente ortogonal, etc.
- SVD (Singular Value Decomposition): factorización $X = U \Sigma V^T$; las columnas de V son los componentes principales y $\Sigma^2/(n-1)$ da la varianza explicada.
- explained_variance_ratio_: array con la fracción de varianza total capturada por cada componente; su suma acumulada guía la elección de n_components.
- IncrementalPCA: versión que procesa mini-batches con partial_fit; útil cuando X no entra en memoria. Costo: ligeramente más lenta y aproximada.
- Randomized PCA (svd_solver="randomized"): algoritmo estocástico que encuentra los primeros d componentes en $O(m \cdot d^2) + O(d^3)$ en vez de $O(m \cdot n^2) + O(n^3)$. Default cuando n_components << min(m, n).
- KernelPCA: aplica PCA en el espacio de features inducido por un kernel (RBF, poly, sigmoid); permite capturar estructura no-lineal (ej. swiss roll).
- gamma: hiperparámetro del kernel RBF que controla el ancho de la gaussiana; valor chico = kernel suave, valor alto = kernel angosto. Se tunea con GridSearchCV.
- Scaling previo: PCA es sensible a la escala — siempre estandarizar (StandardScaler) antes, salvo que todas las variables ya estén en la misma unidad.

Dataset / recursos

- MNIST (fetch_openml("mnist_784")) — 70.000 × 784, ideal para mostrar reducción a ~150 componentes preservando 95% de varianza.
- Swiss roll (sklearn.datasets.make_swiss_roll) — para contrastar PCA lineal vs KernelPCA RBF.
- Géron, Hands-On ML, cap. 8 — Dimensionality Reduction, sección "PCA".

Ejercicios

1. Cargá MNIST, aplicá StandardScaler + PCA(n_components=0.95) y reportá cuántos componentes quedaron.
2. Graficá la curva de varianza acumulada (cumsum(explained_variance_ratio_)) y marcá el codo.
3. Compará tiempos de PCA(svd_solver="full") vs "randomized" sobre MNIST con %timeit.
4. Usá IncrementalPCA con n_batches=100 y verificá que el resultado se aproxima al PCA full (cosine similarity entre componentes > 0.99).
5. Sobre make_swiss_roll(n_samples=1000), aplicá KernelPCA(kernel="rbf", gamma=0.04, n_components=2) y comparalo con PCA lineal en un scatter 2D.

Homework verificable

Construí un pipeline StandardScaler → PCA → LogisticRegression sobre MNIST (subset 10k) y:

1. Reportá accuracy con n_components=50, 100, 154 (154 ≈ 95% varianza).
2. Mostrá la matriz pipe.named_steps["pca"].components_.shape y el explained_variance_ratio_.sum().
3. Entregá notebook + un párrafo explicando el trade-off accuracy vs nº de componentes.

Criterio de aceptación: accuracy ≥ 0.90 con n_components=100 y curva de varianza acumulada graficada.

Errores comunes

1. No escalar antes de PCA — variables con varianza grande (por unidad, no por información) dominan los componentes. Siempre StandardScaler primero.
2. Usar PCA para reducir dimensionalidad antes de un modelo basado en árboles (RF, XGBoost) — no aporta, los árboles son invariantes a rotaciones de features.
3. Interpretar los componentes como "features originales seleccionadas" — son combinaciones lineales, no subconjuntos.
4. Aplicar PCA().fit(X_train_and_test) — fuga de datos. Hacer fit solo sobre train y transform sobre test.
5. Olvidar que KernelPCA no tiene inverse_transform por default — hay que pasar fit_inverse_transform=True.

Preguntas frecuentes

1. ¿Cuántos componentes elegir? Regla práctica: el menor d tal que cumsum(explained_variance_ratio_) ≥ 0.95. También se puede usar n_components="mle" (Minka) o ver el codo del scree plot.
2. ¿PCA mejora siempre la accuracy? No. Reduce ruido y acelera el entrenamiento, pero puede perder información discriminativa. Validá con CV.
3. ¿Cuándo usar Incremental vs Randomized? Incremental si X no entra en RAM; Randomized si entra pero querés velocidad y n_components es chico vs dimensión original.
4. ¿KernelPCA reemplaza a t-SNE/UMAP? No. KernelPCA preserva varianza global en el espacio kernel; t-SNE/UMAP preservan estructura local. Son herramientas distintas (ver clases 083-084).
5. ¿Por qué PCA es sensible a outliers? Porque maximiza varianza y los outliers tienen varianza desproporcionada. Considerá RobustScaler o TruncatedSVD si hay sparse + outliers.

Referencias

- Géron, A. Hands-On Machine Learning, 3ª ed., cap. 8 — Dimensionality Reduction, secciones "PCA", "Randomized PCA", "Incremental PCA", "Kernel PCA".
- scikit-learn user guide: <https://scikit-learn.org/stable/modules/decomposition.html#pca>
- Halko, Martinsson & Tropp (2011), Finding structure with randomness — paper original de Randomized SVD.
- Schölkopf, Smola & Müller (1998), Nonlinear component analysis as a kernel eigenvalue problem.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 093 — Clase 093 — LLE (Locally Linear Embedding)

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 8. Duración estimada: 45 min.*

Objetivo

Entender LLE como técnica no lineal de reducción de dimensionalidad basada en manifold learning: preservar las relaciones lineales locales entre cada punto y sus vecinos para "desenrollar" estructuras curvas (Swiss roll, S-curve) donde PCA falla.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar la intuición de LLE: cada punto se reconstruye como combinación lineal de sus k vecinos, y esa relación se preserva en baja dimensión.
- Aplicar sklearn.manifold.LocallyLinearEmbedding sobre un dataset no lineal (Swiss roll) y visualizar el resultado en 2D.
- Elegir el hiperparámetro n_neighbors y discutir su impacto (sub/sobre-ajuste local).
- Comparar LLE contra PCA y otras técnicas no lineales (Isomap, t-SNE) en términos de qué preservan.
- Identificar cuándo LLE es apropiado y cuándo conviene usar la variante Modified LLE.

Temas

- Motivación: limitaciones de PCA en manifolds curvos.
- Algoritmo LLE en dos pasos: (1) pesos de reconstrucción local, (2) embedding que preserva esos pesos.
- Hiperparámetros: n_neighbors, n_components, method (standard, modified, hessian, ltsa).
- Variantes: Modified LLE (MLLE), Hessian LLE, LTSA.
- Visualización del Swiss roll antes y después.
- Limitaciones: costo computacional $O(m \log(m) \cdot n \cdot k^3 + m \cdot n \cdot k^2)$ y sensibilidad al ruido.

Definiciones y características

- LLE (Locally Linear Embedding): técnica no lineal de reducción de dimensionalidad. No usa proyecciones; descubre cómo cada instancia se relaciona linealmente con sus vecinos más cercanos y busca una representación de baja dimensión donde esas relaciones locales se preserven al máximo.
- Manifold learning: familia de técnicas que asume que los datos de alta dimensión yacen sobre una variedad (manifold) de menor dimensión embebida en el espacio original. LLE, Isomap, t-SNE y UMAP son ejemplos.
- k vecinos (n_neighbors): cantidad de vecinos más cercanos que se usan para reconstruir cada punto. Valor bajo → ruido / desconexiones; valor alto → pierde la estructura local y se parece a PCA.
- Reconstruction weights (W): matriz de pesos que minimiza $\Sigma \, ||x - \Sigma_j \, w_{ij} \, x^j||^2$ con $\Sigma_j \, w_{ij} = 1$ y $w_{ij} = 0$ si j no es vecino de i. Captura la geometría local.
- Modified LLE (MLLE): variante que usa múltiples vectores de pesos por vecindario para evitar el problema de regularización del LLE estándar cuando n_neighbors > n_components. Más estable y suele dar mejores embeddings.
- Hessian LLE / LTSA: otras variantes que reemplazan el criterio de reconstrucción por restricciones geométricas más fuertes (curvatura local, tangentes locales).

Dataset / recursos

- sklearn.datasets.make_swiss_roll(n_samples=1000, noise=0.2) — dataset clásico para visualizar reducción no lineal.
- sklearn.datasets.make_s_curve(n_samples=1000) — manifold alternativo en forma de S.
- sklearn.manifold.LocallyLinearEmbedding.
- Géron, cap. 8 — sección "LLE" y figuras del Swiss roll.

Ejercicios

1. Swiss roll básico: generá un Swiss roll de 1000 puntos, aplicá LocallyLinearEmbedding(n_neighbors=10, n_components=2) y graficá el resultado coloreando por la coordenada original t. Verificá que el rollo quede "desenrollado".
2. Comparación con PCA: sobre el mismo Swiss roll, aplicá PCA(n_components=2) y compará visualmente. Discutí por qué PCA aplasta el rollo en lugar de desenrollarlo.
3. Barrido de n_neighbors: probá n_neighbors ∈ {5, 10, 30, 100} y graficá los 4 embeddings en una grilla 2×2. Describí qué pasa en cada extremo.
4. Modified LLE: repetí el ejercicio 1 con method='modified' y n_neighbors=12. Compará con el LLE estándar — ¿qué embedding se ve más limpio?
5. LLE sobre datos reales: cargá load_digits() (64 dimensiones) y proyectá a 2D con LLE. Coloreá por dígito. ¿Se separan las clases?

Homework verificable

Generá un Swiss roll con n_samples=1500 y random_state=42. Aplicá LLE estándar y MLLE (ambos con n_neighbors=12, n_components=2). Guardá los dos arrays resultantes en embeddings.npz con keys lle y mlle.

Criterio de aceptación: ambos arrays tienen shape (1500, 2), ningún NaN, y la correlación de Spearman entre la primera coordenada del embedding de MLLE y la variable t del Swiss roll original es $|\rho| > 0.95$ (el embedding preservó el orden a lo largo del rollo).

Errores comunes

- No escalar los datos cuando las features tienen magnitudes muy distintas — los "vecinos" pasan a estar dominados por una feature.
- Elegir n_neighbors muy bajo (ej. 3-4): el grafo de vecindad queda desconectado y LLE devuelve embeddings rotos o con componentes colapsados.
- Elegir n_neighbors muy alto: se pierde el carácter local y el resultado tiende a parecerse a PCA, perdiendo la ventaja no lineal.
- Esperar que LLE preserve distancias globales: no lo hace. Para distancias geodésicas usar Isomap; para clusters bien separados, t-SNE o UMAP.
- Usar LLE estándar con n_neighbors > n_components sin regularización: la solución se vuelve degenerada. Para eso existe MLLE.

Preguntas frecuentes

- ¿PCA o LLE? PCA si los datos viven en un subespacio lineal o si querés interpretar componentes / hacer compresión rápida. LLE (y otras técnicas de manifold) si la estructura es curva y querés visualizar. En la práctica, primero PCA para reducir ruido a ~50 dimensiones y después LLE/t-SNE a 2D.
- ¿LLE sirve para preprocesar antes de un clasificador? Rara vez. No es eficiente sobre datos nuevos (no tiene una función transform natural y rápida) y t-SNE/UMAP suelen visualizar mejor. Para preprocesar, PCA o autoencoders.
- ¿Qué diferencia hay con Isomap? Isomap preserva distancias geodésicas globales sobre el grafo de vecinos; LLE preserva sólo relaciones lineales locales. Isomap suele dar embeddings más fieles a la geometría global; LLE es más barato.
- ¿Por qué a veces LLE colapsa todo en una línea? Suele ser un n_neighbors mal elegido o falta de regularización. Probá MLLE o ajustá reg (parámetro de regularización del solver).
- ¿Es determinístico? No del todo: depende del solver de eigendecomposición y de random_state. Fijá random_state y eigen_solver='dense' para reproducibilidad estricta.

Referencias

- Géron, A. Hands-On Machine Learning, 3ra ed., cap. 8 — Dimensionality Reduction, sección "LLE".
- Roweis, S. & Saul, L. (2000). Nonlinear Dimensionality Reduction by Locally Linear Embedding. Science, 290(5500).
- scikit-learn docs: LocallyLinearEmbedding y manifold learning comparison.
- Zhang, Z. & Wang, J. (2007). MLLE: Modified Locally Linear Embedding Using Multiple Weights.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 094 — Clase 094 — MDS, Isomap, t-SNE, UMAP, LDA

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 8 + UMAP docs. Duración estimada: 80 min.*

Objetivo

Conocer y aplicar técnicas de reducción de dimensionalidad más allá de PCA: MDS, Isomap, t-SNE, UMAP y LDA. Entender qué preserva cada una (distancias, geodésicas, vecindarios locales, estructura global, separación entre clases) y cuándo elegir cada método según el problema (visualización 2D, preprocesamiento para clasificación, datos en variedades no lineales).

Resultados de aprendizaje

1. Distinguís entre métodos lineales (PCA, LDA) y no lineales (Isomap, t-SNE, UMAP) y sabés cuándo usar cada uno.
2. Aplicás MDS, Isomap, TSNE, umap.UMAP y LinearDiscriminantAnalysis de scikit-learn / umap-learn sobre datasets reales.
3. Configurás los hiperparámetros clave: perplexity (t-SNE), n_neighbors y min_dist (UMAP), n_neighbors (Isomap).
4. Interpretás correctamente embeddings 2D: qué significan los clusters, qué NO significan las distancias entre clusters en t-SNE.
5. Justificás por qué UMAP suele ser preferible a t-SNE: más rápido, preserva mejor la estructura global y es determinista con random_state.

Temas

Método	Tipo	Preserva	Uso típico	Hiperparámetros clave
MDS	No lineal	Distancias pairwise	Visualización de matrices d	n_components, metric
Isomap	No lineal	Distancias geodésicas (sob	Datos en manifold curvo (S	n_neighbors
t-SNE	No lineal	Vecindarios locales (proba	Visualización 2D/3D de clu	perplexity, learning_rate
UMAP	No lineal	Estructura local y global	Visualización + preprocesa	n_neighbors, min_dist
LDA	Lineal supervisado	Separación entre clases	Reducción previa a clasific	n_components (≤ n_clases−1)

Definiciones y características

- MDS (Multidimensional Scaling): proyecta a baja dimensión intentando que las distancias entre puntos

en el espacio reducido sean lo más parecidas posible a las distancias originales. No asume linealidad pero es costoso: O(n²) en memoria.
- Isomap: variante de MDS que reemplaza la distancia euclídea por la distancia geodésica (camino más corto sobre el grafo de k vecinos más cercanos). Ideal cuando los datos viven en una variedad curva (ej: Swiss roll).
- t-SNE (t-distributed Stochastic Neighbor Embedding): convierte similitudes en distribuciones de probabilidad y minimiza la divergencia KL entre el espacio original y el reducido. Excelente para visualizar clusters; no sirve como preprocesamiento general porque es estocástico y no preserva distancias globales.
- Perplexity (t-SNE): controla cuántos vecinos efectivos considera cada punto. Valores típicos: 5–50. Datasets grandes → perplexity mayor.
- UMAP (Uniform Manifold Approximation and Projection): alternativa moderna a t-SNE basada en topología algebraica. Más rápido, escalable, preserva mejor la estructura global y soporta transform sobre datos nuevos.
- n_neighbors (UMAP): balance entre estructura local (valores bajos, ~5–15) y global (valores altos, ~50–200). min_dist controla qué tan "apretados" se ven los clusters.
- LDA (Linear Discriminant Analysis): método supervisado que proyecta los datos maximizando la separación entre clases y minimizando la varianza dentro de cada clase. Limitado a n_clases − 1 dimensiones.
- Supervisado vs no supervisado: LDA usa las etiquetas y; PCA, MDS, Isomap, t-SNE y UMAP no. Esto hace que LDA sea óptimo como preprocesamiento de clasificación cuando hay etiquetas.

Dataset / recursos

- sklearn.datasets.make_swiss_roll — clásico para Isomap vs PCA.
- sklearn.datasets.load_digits — 8x8 = 64 dims, ideal para visualizar con t-SNE/UMAP.
- sklearn.datasets.fetch_openml('mnist_784') — 70k × 784, para comparar tiempos t-SNE vs UMAP.
- Librerías: scikit-learn (MDS, Isomap, TSNE, LDA) + umap-learn (pip install umap-learn).

Ejercicios

1. Generá un Swiss roll con make_swiss_roll(n_samples=1500) y reducí a 2D con PCA, MDS e Isomap. Graficá los tres y comentá cuál "desenrolla" la variedad.
2. Cargá load_digits y aplicá t-SNE con perplexity $\in \{5, 30, 50, 100\}$. Mostrá los 4 plots y explicá el efecto.
3. Sobre load_digits, compará t-SNE vs UMAP: medí tiempo de ejecución con time.perf_counter() y reportá ambos embeddings 2D.
4. Aplicá LDA a load_digits reduciendo a 2D y a 9D. Entrená un LogisticRegression sobre cada versión y compará accuracy con el original (64 dims).
5. Con UMAP sobre load_digits, probá n_neighbors $\in \{2, 15, 100\}$ con min_dist=0.1. Graficá los tres y explicá el trade-off local vs global.

Homework verificable

Tomá load_digits y producí un script compare_dimred.py que:

1. Aplique PCA(2), Isomap(2), t-SNE(2) y UMAP(2).
2. Para cada embedding, entrene un KNeighborsClassifier(n_neighbors=5) con cross_val_score (cv=5) sobre el embedding 2D.
3. Imprima una tabla con método, tiempo de ajuste y accuracy media.

Criterio: UMAP debe terminar al menos 3× más rápido que t-SNE sobre los 1797 dígitos, y la accuracy 2D de

UMAP debe superar 0.90.

Errores comunes

1. Interpretar las distancias entre clusters en t-SNE como reales. t-SNE preserva vecindarios locales, no distancias globales: dos clusters cercanos en el plot no son necesariamente similares.
2. Usar t-SNE como preprocesamiento de un modelo. No tiene transform confiable sobre datos nuevos; usá UMAP o PCA para eso.
3. No escalar antes de Isomap o t-SNE. Estos métodos dependen de distancias; sin StandardScaler las features con escala grande dominan.
4. Pedirle a LDA más componentes que n_clases − 1. scikit-learn lanza error; LDA está topeado por la cantidad de clases.
5. Olvidar random_state en t-SNE/UMAP. Son estocásticos: sin semilla el embedding cambia en cada corrida y los plots no son reproducibles.

Preguntas frecuentes

1. ¿t-SNE o UMAP? UMAP en casi todos los casos: más rápido, preserva estructura global, tiene transform, escala a millones de puntos. t-SNE sigue siendo válido para visualización pequeña cuando ya tenés pipelines hechos.
2. ¿Cuándo usar Isomap en vez de PCA? Cuando sospechás que los datos viven en una variedad curva (ej: imágenes de un objeto rotando). PCA proyecta linealmente y aplana mal la curvatura.
3. ¿LDA o PCA antes de clasificar? Si tenés etiquetas, LDA suele dar mejor separación con menos dimensiones. PCA es agnóstico al target y puede tirar info útil.
4. ¿Por qué MDS es tan lento? Calcula la matriz de distancias $O(n^2)$ y resuelve un problema de optimización. Para n > 5000 es impráctico; usá Isomap o UMAP.
5. ¿Puedo usar UMAP como preprocesamiento supervisado? Sí: UMAP acepta y en fit y aprende un embedding que respeta las etiquetas (semi-supervisado).

Referencias

- Géron, Hands-On ML, cap. 8 — "Other Dimensionality Reduction Techniques".
- McInnes, Healy & Melville (2018). UMAP: Uniform Manifold Approximation and Projection. arXiv:1802.03426.
- van der Maaten & Hinton (2008). Visualizing Data using t-SNE. JMLR.
- Documentación: <https://umap-learn.readthedocs.io/> · <https://scikit-learn.org/stable/modules/manifold.html>

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 095 — Clase 095 — Clustering K-Means: selección de K, MiniBatch

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 9. Duración estimada: 70 min.*

Objetivo

Aplicar K-Means para segmentar datos no etiquetados, elegir el número de clusters K con criterios reproducibles (elbow, silhouette) y escalar el algoritmo con MiniBatchKMeans cuando el dataset no entra

cómodo en memoria.

Resultados de aprendizaje

Al finalizar, vas a poder:

- Explicar el algoritmo de Lloyd y por qué K-Means++ mejora la inicialización aleatoria.
- Ajustar KMeans de scikit-learn fijando n_init y random_state para resultados estables.
- Elegir K combinando elbow method (inercia vs K) y silhouette score.
- Reemplazar KMeans por MiniBatchKMeans y discutir el trade-off velocidad / calidad.
- Diagnosticar por qué K-Means falla sin escalado o frente a clusters no esféricos.

Temas

1. Clustering no supervisado: planteo del problema.
2. Algoritmo de Lloyd: asignación + actualización de centroides.
3. Inicialización: random vs K-Means++; rol de n_init.
4. Inercia (within-cluster sum of squares) como objetivo.
5. Selección de K: elbow method, silhouette score, gap statistic (mención).
6. MiniBatchKMeans para datasets grandes / streaming.
7. Limitaciones: clusters no convexos, densidades distintas, sensibilidad al escalado.

Definiciones y características

- K-Means — algoritmo iterativo que parte el espacio de features en K regiones de Voronoi minimizando la suma de distancias cuadradas a los centroides.
- K-Means++ — esquema de inicialización que elige centroides iniciales separados entre sí; baja la varianza entre corridas y suele converger en menos iteraciones.
- Inertia — suma de distancias cuadradas de cada punto a su centroide asignado (model.inertia_). Monótona decreciente con K; sirve para el elbow.
- Elbow method — graficar inertia_ vs K y elegir el K donde la curva "se quiebra" (la mejora marginal se aplana).
- Silhouette score — métrica en [-1, 1] que combina cohesión intra-cluster y separación inter-cluster. Más alto = mejor; útil para comparar K.
- MiniBatchKMeans — variante que actualiza centroides con mini-lotes en vez de pasar por todo X en cada iteración. ~5-10× más rápido, inercia levemente peor.
- Centroides — vectores (K, n_features) que representan el "centro" de cada cluster; en KMeans es la media de los puntos asignados.
- n_init — cantidad de corridas con distintas semillas; scikit-learn se queda con la de menor inercia. Default 10 (sklearn ≥1.4: "auto").

Dataset / recursos

- sklearn.datasets.make_blobs(n_samples=2000, centers=5, cluster_std=0.8, random_state=42) para los ejercicios de exploración.
- sklearn.datasets.load_digits() (1797 × 64) para el ejercicio de MiniBatchKMeans.
- Opcional: dataset Mall_Customers.csv o cualquier CSV tabular numérico ya escalado.

Ejercicios

1. Generá blobs con 5 centros, ajustá KMeans(n_clusters=5, n_init=10, random_state=42) y graficá los puntos coloreados por etiqueta junto con cluster_centers_.
2. Para K en range(2, 11), calculá inertia_ y silhouette_score. Graficá ambas curvas y justificá el K elegido.

3. Repetí el ajuste sin escalar un dataset donde una feature tenga escala 100× mayor que la otra. Compará con StandardScaler previo y comentá el cambio en las etiquetas.
4. Sobre load_digits(), ajustá KMeans(n_clusters=10) y MiniBatchKMeans(n_clusters=10, batch_size=256). Cronometrá ambos con %timeit y compará inercias.
5. Probá K-Means sobre make_moons(noise=0.05). Mostrá visualmente por qué falla y proponé qué algoritmo usarías en su lugar (te lo vamos a contestar en la clase 086).

Homework verificable

Entregá un script homework_085.py que:

1. Cargue make_blobs con random_state=42, 4 centros reales.
2. Recorra K = 2..8 y guarde inercia + silhouette en un DataFrame.
3. Imprima el K óptimo según silhouette y genere elbow.png + silhouette.png.

Criterio de aceptación: el script corre sin errores con python homework_085.py, imprime K óptimo = 4, y produce los dos PNG con ejes y título legibles.

Errores comunes

1. No escalar features — K-Means usa distancia euclídea; una feature en miles domina a otra en décimas. Aplicá StandardScaler salvo que tengas razón explícita para no hacerlo.
2. No fijar n_init (o dejarlo en 1) — una sola corrida puede caer en un mínimo local malo. Mantené n_init >= 10 y random_state fijo para reproducibilidad.
3. Elegir K mirando solo la inercia — la inercia siempre baja al subir K. Sin elbow visible ni silhouette, el criterio queda arbitrario.
4. Asumir clusters esféricos — K-Means no sirve para "lunas", anillos o densidades muy distintas; ahí necesitás DBSCAN o GMM.
5. Usar MiniBatchKMeans con batch_size muy chico — la varianza entre updates explota y los centroides oscilan. Mantenelo en 256–1024 salvo benchmarks específicos.

Preguntas frecuentes

1. ¿Elbow o silhouette? Silhouette es más objetivo cuando no hay codo claro; usá ambos y, si discrepan, priorizá silhouette + criterio de negocio.
2. ¿Cuántas iteraciones máximas? El default max_iter=300 alcanza casi siempre. Si convergés antes, sklearn corta solo.
3. ¿K-Means es determinístico? No: depende de la inicialización. Fijá random_state y n_init para resultados reproducibles.
4. ¿Cuándo usar MiniBatchKMeans? Cuando n_samples > 10^5 o el dataset llega en streaming. Para datasets chicos, el K-Means clásico es más preciso y suficientemente rápido.
5. ¿Sirve para detectar outliers? No directamente; los outliers distorsionan los centroides. Para outliers usá DBSCAN o IsolationForest.

Referencias

- Géron, A. Hands-On Machine Learning (3ra ed.), cap. 9 — sección "K-Means".
- scikit-learn user guide: Clustering — K-Means.
- Arthur, D. & Vassilvitskii, S. (2007). k-means++: The Advantages of Careful Seeding.
- Sculley, D. (2010). Web-Scale K-Means Clustering (paper original de MiniBatchKMeans).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.

- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 096 — Clase 096 — DBSCAN

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 9. Duración estimada: 55 min.*

Objetivo

Que el alumno aplique DBSCAN (Density-Based Spatial Clustering of Applications with Noise) para descubrir clusters de forma arbitraria e identificar outliers nativamente, sin tener que predefinir k como en K-Means. Que sepa elegir eps con un k-distance plot y entienda cuándo conviene escalar a HDBSCAN.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Ejecutar DBSCAN con sklearn.cluster.DBSCAN y tunear eps y min_samples para un dataset 2D.
2. Elegir eps mirando el codo del k-distance plot (no a ojo).
3. Identificar outliers vía la etiqueta -1 que DBSCAN asigna a los puntos ruido.
4. Distinguir core / border / noise points y entender la noción de density-reachable.
5. Comparar DBSCAN vs K-Means y saber cuándo usar HDBSCAN (eps variable, clusters de densidad mixta).

Temas

#	Tema	Por qué importa
1	Intuición de densidad vs centroides	DBSCAN encuentra "regiones densas" — no ne
2	Hiperparámetros eps y min_samples	Son los dos botones del modelo. Mal puesto
3	Core / border / noise points	Vocabulario base para leer la salida y dep
4	k-distance plot para elegir eps	Método estándar para no ir a ciegas.
5	Etiqueta -1 y detección de outliers	DBSCAN es de los pocos clusterers que dete
6	Limitaciones: densidad uniforme, curse of	Por qué falla en datasets reales con clust
7	HDBSCAN como evolución	eps jerárquico: maneja densidad variable y

Definiciones y características

DBSCAN

: Algoritmo de clustering basado en densidad. Agrupa puntos que están densamente conectados y marca como ruido los que quedan en regiones de baja densidad. No requiere k. Complejidad ~O(n log n) con índice espacial.

eps (epsilon)

: Radio de la vecindad alrededor de cada punto. Define qué se considera "cerca". Es el hiperparámetro más sensible — un cambio chico colapsa o explota los clusters.

min_samples

: Cantidad mínima de puntos (incluyendo el propio) dentro de eps para que un punto se considere core. Regla práctica: min_samples ≥ dim + 1, típicamente 4–10 para 2D.

Core point

: Punto con al menos min_samples vecinos dentro de eps. Es el "núcleo" desde el cual crece el cluster.

Border point

: Punto que está dentro del eps de un core, pero él mismo no tiene min_samples vecinos. Pertenece al cluster pero no propaga.

Noise point (outlier)

: Punto que no es core ni border. DBSCAN lo etiqueta como -1. Sale gratis del modelo, sin entrenar un detector aparte.

Density-reachable

: Relación que conecta dos puntos vía una cadena de core points. Es la definición formal de "pertenecer al mismo cluster" en DBSCAN.

HDBSCAN (Hierarchical DBSCAN)

: Evolución de DBSCAN que reemplaza eps por una jerarquía. Maneja clusters de densidades distintas, expone un parámetro más intuitivo (min_cluster_size) y devuelve probabilidades de pertenencia. Es el default moderno para clustering no supervisado.

Dataset / recursos

- make_moons(n_samples=1000, noise=0.05) de scikit-learn — dos lunas entrelazadas, el caso canónico donde K-Means falla y DBSCAN brilla.
- make_blobs con densidades distintas para mostrar la limitación de DBSCAN y motivar HDBSCAN.

Ejercicios

1. DBSCAN sobre moons. Entrená DBSCAN(eps=0.2, min_samples=5) sobre make_moons. Graficá los clusters y contá cuántos puntos quedaron como -1.

2. K-distance plot. Calculá la distancia al k-ésimo vecino más cercano (k=min_samples) para todos los puntos, ordenala y graficala. Identificá el codo y usalo como eps.

3. Sensibilidad a eps. Probá eps {0.05, 0.1, 0.2, 0.5} y reportá número de clusters y % de ruido en cada caso. Mostrá cómo eps chico = todo ruido y eps grande = un cluster.

4. DBSCAN vs K-Means. Sobre el mismo make_moons, corré ambos con k=2. Mostrá visualmente que K-Means parte las lunas por la mitad y DBSCAN las separa bien.

5. HDBSCAN sobre densidades mixtas. Generá blobs con cluster_std distinto por blob. Mostrá que un eps único en DBSCAN no puede capturar ambos, y que HDBSCAN(min_cluster_size=20) sí.

Homework verificable

Notebook que sobre un dataset 2D sintético (moons + outliers inyectados a mano) haga: (a) k-distance plot y elección razonada de eps; (b) DBSCAN con esos hiperparámetros; (c) reporte de % de outliers detectados vs inyectados; (d) comparación visual con K-Means; (e) corrida con HDBSCAN sobre blobs de densidad mixta.

Criterio de aceptación: eps justificado con el codo del k-distance plot (no a ojo). DBSCAN recupera al menos el 80% de los outliers inyectados. La comparación con K-Means muestra explícitamente la falla de los centroides en datos no convexos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Elegir eps a ojo y obtener un solo cluster	El parámetro es altamente sensible y depen
DBSCAN sobre features sin escalar	eps es una distancia euclidiana; si una fe
min_samples=1 o min_samples=2	Todo termina siendo core point, no hay rui
Aplicar DBSCAN en alta dimensión (>10)	Curse of dimensionality: las distancias se
Pedirle predict() a DBSCAN	DBSCAN no tiene predict — no hay forma nat

Preguntas frecuentes

¿K-Means o DBSCAN?

K-Means si los clusters son aproximadamente esféricos, de tamaño parecido, y sabés (o podés estimar) k. DBSCAN si los clusters tienen forma arbitraria, no sabés cuántos hay, o necesitás detectar outliers en la misma pasada. Para producción moderna: HDBSCAN suele ganar a ambos cuando no tenés intuición previa.

¿Por qué DBSCAN devuelve -1?

Es la etiqueta convencional para noise points — puntos que no pertenecen a ningún cluster. No es un cluster más, es la salida natural del algoritmo para outliers.

¿Cómo elijo min_samples?

Regla práctica: min_samples = 2 * dim. Para 2D, usá 4. Para 3D, 6. Más grande = clusters más robustos pero más ruido. En la práctica, min_samples mueve menos la aguja que eps.

¿DBSCAN escala a millones de puntos?

Con algorithm='ball_tree' o 'kd_tree' queda en ~O(n log n). Para >1M puntos en alta dimensión, considerá HDBSCAN o aproximaciones tipo sklearn.cluster.OPTICS.

¿Cuándo HDBSCAN vence a DBSCAN?

Siempre que los clusters tengan densidades distintas entre sí. DBSCAN usa un eps global; HDBSCAN construye una jerarquía y elige el corte óptimo por cluster. Además expone min_cluster_size que es más intuitivo de tunear que eps.

Referencias

- Géron, cap. 9 § "DBSCAN".
- scikit-learn DBSCAN user guide
- HDBSCAN docs
- Ester et al. (1996), A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise — paper original.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 097 — Clase 097 — Agglomerative, BIRCH, Mean Shift, Affinity Propagation, Spectral

Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 9 § Other Clustering Algorithms. Duración estimada: 70 min.

Objetivo

Que el alumno conozca el zoológico de algoritmos de clustering más allá de K-Means y DBSCAN — Agglomerative (jerárquico, lee dendrogramas), BIRCH (escalable a millones de filas), Mean Shift (denso, sin especificar k), Affinity Propagation (elige exemplars por message-passing) y Spectral Clustering (clustering vía autovectores del grafo de similitud) — y sepa cuándo elegir cada uno según tamaño del dataset, forma de los clusters y necesidad de jerarquía.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Construir un dendrograma con scipy.cluster.hierarchy.linkage y cortarlo a una altura dada para obtener clusters.
2. Elegir un linkage (ward, complete, average, single) según la forma esperada de los clusters y conocer el efecto del chaining en single.
3. Usar BIRCH para datasets que no entran en memoria, ajustando threshold y branching_factor.
4. Aplicar Mean Shift ajustando bandwidth (con estimate_bandwidth) y entender por qué descubre el número de clusters automáticamente.
5. Decidir entre Affinity Propagation y Spectral Clustering según escalabilidad (AP es O(n²) en memoria) y geometría (Spectral funciona en clusters no convexos).

Temas

#	Algoritmo	Hiperparámetro clav	k a priori	Complejidad	Cuándo usarlo
1	Agglomerative	linkage, n_clusters o d	Sí (o threshold)	O(n³) / O(n² log n)	Querés jerarquía + dendrograma; n ≤
2	BIRCH	threshold, branching_f	Opcional	O(n)	Datasets grandes (millones) que no e
3	Mean Shift	bandwidth	No	O(n²) por iter	Modos de densidad; clusters de tama
4	Affinity Propagation	damping, preference	No	O(n²·T)	Pocos puntos, querés exemplars rea
5	Spectral	n_clusters, affinity (rbf,	Sí	O(n³) (eigen)	Clusters no convexos, manifolds, gra

Definiciones y características

Clustering aglomerativo (jerárquico)
: Algoritmo bottom-up: arranca con cada punto como su propio cluster y en cada paso fusiona los dos clusters más cercanos hasta llegar a uno solo. Produce una jerarquía completa que se visualiza como dendrograma. En sklearn: AgglomerativeClustering(n_clusters=k) o distance_threshold=d (entonces n_clusters=None).

Linkage
: Criterio para medir la distancia entre dos clusters durante la fusión. Las cuatro opciones que importan:

- ward (default sklearn): minimiza la varianza intra-cluster al fusionar. Tiende a producir clusters de tamaño parecido. Solo con distancia euclidiana.
- complete (max-linkage): distancia entre los dos puntos más lejanos. Produce clusters compactos.
- average: promedio de todas las distancias entre puntos de ambos clusters. Compromiso razonable.
- single (min-linkage): distancia entre los dos puntos más cercanos. Sufre chaining — un puente fino de puntos une dos clusters que deberían quedar separados.

Dendrograma

: Diagrama de árbol invertido que muestra el orden y la altura (= distancia entre clusters fusionados) de cada merge. Cortar con una línea horizontal a altura h da los clusters resultantes. Se grafica con scipy.cluster.hierarchy.dendrogram(Z) donde Z = linkage(X, method='ward').

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)

: Algoritmo de dos pasos para datasets enormes. (1) Recorre los datos una sola vez construyendo un CF-tree (Clustering Feature tree) que comprime puntos en sub-clusters compactos. (2) Aplica un clustering tradicional sobre los sub-clusters. Streaming-friendly (online). Hiperparámetros: threshold (radio máximo de un sub-cluster — más chico, más sub-clusters) y branching_factor (hijos máximos por nodo).

Mean Shift

: Algoritmo basado en densidad. Cada punto "trepa" iterativamente hacia el modo (máximo local) de la densidad estimada con un kernel gaussiano de ancho bandwidth. Puntos que convergen al mismo modo forman un cluster. No requiere k. Sensible al bandwidth: chico → muchos clusters chiquitos; grande → todo un cluster. sklearn.cluster.estimate_bandwidth(X) da un valor de arranque razonable.

Bandwidth

: Ancho del kernel en Mean Shift. Análogo conceptual al eps de DBSCAN — define qué tan "lejos" mira cada punto para estimar densidad.

Affinity Propagation

: Algoritmo de message passing entre puntos. Cada punto manda dos tipos de mensajes (responsibility y availability) a los demás hasta que emerge un subconjunto de exemplars (puntos representativos reales del dataset) y cada otro punto se asigna al exemplar más afín. No requiere k. Caro: $O(n^2)$ en memoria y $O(n^2 \cdot T)$ en tiempo — no escala más allá de unos miles de puntos. Hiperparámetros: damping (entre 0.5 y 1, para evitar oscilaciones) y preference (controla cuántos exemplars emergen).

Spectral Clustering

: Construye una matriz de similitud S (típicamente con kernel RBF o k-NN), calcula el grafo Laplaciano L = D − S, toma los k autovectores de menor autovalor, los apila como nuevas features y corre K-Means en ese embedding. Funciona donde K-Means falla: clusters no convexos, dos lunas entrelazadas, círculos concéntricos. Cuello de botella: eigendecomposición $O(n^3)$.

Similarity matrix

: Matriz n×n donde S[i,j] mide qué tan parecidos son los puntos i y j (alta similitud = cerca). Es la entrada de Spectral y Affinity Propagation. Para Spectral con kernel RBF: $S[i,j] = \exp(-\gamma \cdot ||x_i - x_j||^2)$.

Dataset / recursos

- make_blobs (sklearn) — para Agglomerative y BIRCH (clusters convexos).
- make_moons y make_circles — clusters no convexos, donde Spectral brilla y K-Means/Agglomerative-ward fallan.
- Dataset opcional escalable: generá 1M de puntos sintéticos para probar BIRCH vs K-Means en tiempo y memoria.

Ejercicios

1. Dendrograma sobre make_blobs. Generá 50 puntos en 4 blobs. Calculá Z = linkage(X, method='ward') y graficá el dendrograma. Cortá a una altura que dé 4 clusters. Comparalo con AgglomerativeClustering(n_clusters=4).

2. Linkage comparison. Sobre make_moons(n_samples=300, noise=0.05), ajustá AgglomerativeClustering(n_clusters=2) con linkage='ward', 'complete', 'average', 'single'. Ploteá los 4

resultados lado a lado. ¿Cuál recupera las dos lunas? ¿Por qué?

3. BIRCH escalable. Generá 100k puntos con make_blobs. Medí tiempo y memoria de BIRCH(n_clusters=5) vs KMeans(n_clusters=5). Variá threshold {0.1, 0.5, 1.0} y mostrá cómo cambia el número de sub-clusters internos del CF-tree.

4. Mean Shift sin saber k. Sobre 3 blobs claros, corré MeanShift(bandwidth=estimate_bandwidth(X, quantile=0.2)). Verificá que recupera 3 clusters automáticamente. Después poné quantile=0.5 y observá cómo colapsa a menos clusters.

5. Spectral vs K-Means en make_circles. Generá dos círculos concéntricos con make_circles(n_samples=500, factor=0.5, noise=0.05). Ajustá KMeans(n_clusters=2) y SpectralClustering(n_clusters=2, affinity='nearest_neighbors'). Graficá ambos. Documentá por qué Spectral funciona y K-Means no.

Homework verificable

Notebook con: (a) dendrograma de load_iris().data usando ward linkage, cortado para obtener 3 clusters; (b) tabla comparativa de Adjusted Rand Index entre las labels verdaderas y las predichas por Agglomerative, BIRCH, Mean Shift, Affinity Propagation y Spectral — corriendo los 5 sobre el mismo Iris; (c) gráfico 2D (con las 2 primeras PCs) coloreado por las labels de cada método; (d) párrafo justificando qué método ganó y por qué.

Criterio de aceptación: el dendrograma se visualiza correctamente con eje y = distancia. La tabla muestra ARI [-1, 1] para los 5 métodos. Spectral y Agglomerative-ward deberían superar 0.7 en Iris.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
AgglomerativeClustering con linkage='ward'	ward solo soporta distancia euclidiana. Fi
Dendrograma con miles de hojas, ilegible	Demasiados puntos. Fix: pasale truncate_mo
MeanShift corre eterno o devuelve 1 solo c	bandwidth mal calibrado (default = estimat
AffinityPropagation no converge — warning	damping muy bajo (oscilaciones). Fix: subí
SpectralClustering con affinity='rbf' da c	gamma del RBF mal escalado. Fix: usá affin

Preguntas frecuentes

¿Cuándo Agglomerative en lugar de K-Means?

Cuando querés (a) explorar la estructura jerárquica del dataset con un dendrograma antes de decidir k, (b) no asumir clusters esféricos (con average o complete linkage), o (c) reproducibilidad determinística — Agglomerative no depende de inicialización aleatoria. Costo: $O(n^3)$, inviable para n > ~10k.

¿BIRCH reemplaza a MiniBatch K-Means?

Casi. Ambos son escalables. BIRCH gana cuando los datos llegan en streaming o no entran ni siquiera en lotes (es one-pass). MiniBatch K-Means es más simple y suele ser suficiente para datasets que entran en disco. Si dudás, empezá por MiniBatch K-Means.

¿Cómo elijo el bandwidth de Mean Shift sin probar a mano?

Usá sklearn.cluster.estimate_bandwidth(X, quantile=0.2, n_samples=500). El quantile controla el ancho del kernel basado en distancias entre pares de puntos muestreados. Empezá con 0.2 y bajá si querés clusters más finos.

¿Affinity Propagation sirve para algo en la práctica?

Para datasets chicos (< 1000 puntos) donde necesitás exemplars reales — por ejemplo, elegir 50 reviews representativas de 800. Para clustering general, los demás algoritmos lo superan en velocidad y robustez.

¿Spectral Clustering es lo mismo que clustering sobre PCA?

No. PCA proyecta sobre direcciones de máxima varianza (lineal). Spectral usa autovectores del grafo Laplaciano de la matriz de similitud — captura estructura no lineal (manifolds). Por eso recupera dos lunas entrelazadas, donde PCA + K-Means falla.

Referencias

- Géron, Hands-On ML, cap. 9 § Other Clustering Algorithms.
- scikit-learn — Clustering overview (tabla comparativa)
- scipy linkage + dendrogram
- Frey & Dueck (2007) — Affinity Propagation paper
- von Luxburg (2007) — A Tutorial on Spectral Clustering

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 098 — Clase 098 — Gaussian Mixture Models

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 9. Duración estimada: 70 min.*

Objetivo

Entender los Gaussian Mixture Models (GMM) como modelo probabilístico de soft clustering, ajustarlos con el algoritmo EM en scikit-learn, elegir el número de componentes con BIC/AIC, y conocer las variantes (covariance_type, BayesianGaussianMixture) para aplicarlas en clustering, densidad y detección de anomalías.

Resultados de aprendizaje

Al finalizar la clase vas a poder:

1. Explicar qué es un GMM y cómo se diferencia de K-Means (asignación dura vs. probabilística).
2. Ajustar un GaussianMixture con scikit-learn y obtener predict, predict_proba y score_samples.
3. Seleccionar el número óptimo de componentes comparando BIC y AIC en una grilla.
4. Elegir el covariance_type apropiado (full, tied, diag, spherical) según supuestos y tamaño del dataset.
5. Usar BayesianGaussianMixture para que el modelo descarte componentes innecesarios automáticamente.

Temas

- Modelos de mezcla: intuición y fórmula.
- Algoritmo Expectation-Maximization (EM): paso E (responsabilidades) + paso M (actualizar medias, covarianzas, pesos).
- API de sklearn.mixture.GaussianMixture: n_components, covariance_type, n_init, tol.

- Métodos: predict_proba (soft), score_samples (log-densidad), sample (generar datos).
- Selección de modelo con BIC y AIC.
- Variantes de covarianza y su impacto en parámetros / sesgo / varianza.
- Bayesian GMM con prior de Dirichlet: aprende cuántos componentes hacen falta.
- Detección de anomalías por umbral sobre densidad.

Definiciones y características

- Gaussian Mixture Model (GMM): modelo generativo que asume que los datos provienen de una mezcla ponderada de K distribuciones gaussianas multivariadas. Cada punto tiene probabilidad de pertenecer a cada componente.
- Mixture (mezcla): combinación convexa $p(x) = \Sigma\ \pi k \cdot N(x \mid \mu k, \Sigma k)$ con pesos πk que suman 1.
- EM algorithm (Expectation-Maximization): procedimiento iterativo que alterna entre estimar las responsabilidades de cada componente sobre cada punto (E) y reestimar parámetros maximizando la verosimilitud esperada (M). Converge a un óptimo local.
- Soft clustering: asignación probabilística — cada punto recibe un vector de probabilidades de pertenencia, no una etiqueta única. Útil cuando los clusters se solapan.
- covariance_type: controla la forma de las matrices de covarianza.
- full: cada componente tiene su propia matriz completa (más flexible, más parámetros).
- tied: todos comparten la misma matriz.
- diag: matrices diagonales (ejes alineados).
- spherical: una sola varianza escalar por componente (clusters esféricos).
- BIC (Bayesian Information Criterion): $-2 \cdot \log L + k \cdot \log n$. Penaliza fuerte la complejidad; preferí el modelo con BIC mínimo. Tiende a elegir modelos más parsimoniosos.
- AIC (Akaike Information Criterion): $-2 \cdot \log L + 2 \cdot k$. Penaliza menos que BIC; suele elegir modelos algo más complejos.
- BayesianGaussianMixture: variante con prior de Dirichlet sobre los pesos. Si fijás n_components alto, el modelo "apaga" los componentes sobrantes (pesos ≈ 0), evitando elegir K manualmente.

Dataset / recursos

- sklearn.datasets.make_blobs con clusters de varianzas distintas (para ver el aporte vs. K-Means).
- sklearn.datasets.load_iris para validar agrupamiento contra etiquetas reales.
- Opcional: dataset Old Faithful (geyser) — clásico ejemplo de mezcla bimodal.

Ejercicios

1. Ajuste básico: generá make_blobs con 3 centros y cluster_std variable. Ajustá GaussianMixture(n_components=3) y compará predict con las etiquetas reales (ARI).
2. Soft vs. hard: sobre el mismo dataset, mostrá predict_proba de 5 puntos cerca de la frontera. Compará con la asignación dura de K-Means.
3. Selección de K con BIC/AIC: ajustá GMMs con n_components de 1 a 10. Graficá BIC y AIC vs. K e identificá el mínimo.
4. covariance_type: repetí el ajuste con los 4 tipos sobre un dataset con clusters elípticos rotados. Compará BIC y visualizá las elipses de covarianza.
5. Bayesian GMM: ajustá BayesianGaussianMixture(n_components=10, weight_concentration_prior=0.01) sobre datos con 3 clusters reales y mostrá que los pesos efectivos son ≈ 3.

Homework verificable

Sobre load_iris (sin usar la etiqueta para entrenar):

1. Estandarizá las features.
2. Ajustá GMMs con n_components de 1 a 8 y covariance_type en ['full', 'tied', 'diag', 'spherical'].
3. Elegí el (n_components, covariance_type) con BIC mínimo.
4. Calculá el Adjusted Rand Index entre predict del mejor modelo y y real.

Criterio: ARI ≥ 0.85 y el mejor modelo elegido por BIC tiene n_components ∈ {2, 3}.

Errores comunes

1. No estandarizar las features. GMM con covariance_type='spherical' o 'diag' es muy sensible a la escala.
2. Usar n_init=1 (default). EM converge a óptimos locales; subí a n_init=10 para resultados estables.
3. Elegir K mirando solo la log-verosimilitud. Siempre aumenta con K; usá BIC/AIC que penalizan complejidad.
4. covariance_type='full' con pocos datos y alta dimensión: explota los parámetros ($K \cdot d \cdot (d+1)/2$) y sobreajusta. Bajá a diag o tied.
5. Asumir que predict_proba da incertidumbre calibrada. Da responsabilidades dentro del modelo; si el modelo está mal especificado, las probas pueden ser engañosas.

Preguntas frecuentes

1. ¿GMM o K-Means? K-Means es más rápido y simple, pero asume clusters esféricos de igual tamaño y asigna duro. GMM permite clusters elípticos, tamaños distintos, solapamiento y da probabilidades. Si tus clusters son claramente esféricos y no se solapan, K-Means alcanza.
2. ¿BIC o AIC? Si querés el modelo más parsimonioso y tenés muchos datos, usá BIC. Si priorizás capacidad predictiva y no te molesta un modelo algo más complejo, usá AIC. En la práctica, mostrá los dos y mirá si coinciden.
3. ¿Cómo detecto anomalías con GMM? Calculá score_samples(X) (log-densidad) y marcá como anomalía los puntos con score bajo un percentil (ej. 4%).
4. ¿Sirve para datos de alta dimensión? Con covariance_type='full' no escala bien (muchos parámetros). Reducí dimensión con PCA o usá diag/tied.
5. ¿Qué hace BayesianGaussianMixture que no haga GMM? Aprende cuántos componentes son necesarios: si fijás n_components alto, los sobrantes quedan con peso ≈ 0. Evita la búsqueda en grilla de K.

Referencias

- Géron, Hands-On ML (3ª ed.), cap. 9 § "Gaussian Mixtures".
- scikit-learn — Gaussian Mixture Models.
- scikit-learn — GaussianMixture y BayesianGaussianMixture.
- Bishop, Pattern Recognition and Machine Learning, cap. 9 (EM y mixtures).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 099 — Clase 099 — Detección de anomalías: Isolation Forest, LOF, One-Class SVM

Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 9 + sklearn outlier detection. Duración estimada: 70 min.

Objetivo

Que el alumno detecte puntos anómalos (fraude, fallas, outliers) en datos sin etiquetas, eligiendo entre Isolation Forest, LOF, One-Class SVM y Elliptic Envelope según la geometría del problema, y entendiendo la diferencia entre outlier detection (entrenar con datos sucios) y novelty detection (entrenar limpio, predecir sobre nuevos).

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Distinguir outlier detection vs novelty detection y elegir el algoritmo acorde.
2. Entrenar IsolationForest y ajustar el hiperparámetro contamination.
3. Usar LocalOutlierFactor en modo novelty=False (fit_predict) y novelty=True (predict).
4. Aplicar OneClassSVM y EllipticEnvelope, reconociendo sus supuestos (kernel, gaussianidad).
5. Evaluar detectores de anomalías con score_samples, ROC-AUC y reglas de negocio (top-k).

Temas

#	Tema	Por qué importa
1	Outlier vs novelty detection	Define cómo se entrena y qué se predice.
2	Isolation Forest	Default sólido en alta dimensión, escala b
3	Local Outlier Factor (LOF)	Detecta anomalías locales por densidad.
4	One-Class SVM	Frontera no lineal con kernel RBF; sensibl
5	Elliptic Envelope	Asume distribución gaussiana; útil en dato
6	score_samples y umbrales	Salida continua > etiqueta binaria.
7	Evaluación sin labels	Top-k, inspección manual, ROC si hay groun

Definiciones y características

Anomaly / outlier detection

: Tarea no supervisada de identificar instancias que difieren significativamente de la mayoría. Entrena sobre un dataset contaminado (con algunas anomalías) y las marca dentro del mismo dataset. Salida sklearn: +1 (inlier) / -1 (outlier).

Novelty detection

: Variante donde el entrenamiento se hace sobre datos limpios (solo inliers), y luego en inferencia se predice si nuevos puntos son normales o novedosos. LocalOutlierFactor(novelty=True) y OneClassSVM operan en este modo.

Isolation Forest

: Ensemble de árboles que aíslan puntos eligiendo features y splits al azar. Las anomalías quedan aisladas con menos splits (path corto). Escala bien a alta dimensión y N grande. Default recomendado.

contamination

: Hiperparámetro que indica la fracción esperada de anomalías en los datos (entre 0 y 0.5, o 'auto'). Define el umbral del score. Si te equivocás mucho, calibrá con score_samples y elegí el umbral a mano.

Local Outlier Factor (LOF)

: Compara la densidad local de un punto con la de sus k vecinos. Si la densidad es mucho menor que la de sus vecinos, es outlier. Bueno para anomalías locales (un punto raro dentro de un cluster). No escala bien a N muy grande.

One-Class SVM

: Aprende una frontera (típicamente con kernel RBF) que envuelve la región "normal" del espacio. Sensible a escala (estandarizar siempre) y al hiperparámetro nu (cota superior de la fracción de outliers). Lento en N grande; preferir SGDOneClassSVM para datasets grandes.

Elliptic Envelope

: Ajusta una gaussiana robusta a los datos y marca como outliers los puntos fuera de un elipsoide de confianza. Asume distribución unimodal aproximadamente gaussiana — si los datos son multimodales o tienen estructura compleja, falla.

score_samples(X)

: Devuelve un score continuo de "normalidad" por instancia (más alto = más normal). Útil para rankear top-k anomalías o elegir el umbral a mano en lugar de confiar en contamination.

Dataset / recursos

Sintético con make_blobs + outliers uniformes inyectados, o sklearn.datasets.fetch_kddcup99 (detección de intrusiones, real y con labels para evaluar).

Ejercicios

1. Isolation Forest baseline. Generá 2 blobs + 5% de outliers uniformes. Ajustá IsolationForest(contamination=0.05). Plot 2D con inliers vs outliers detectados.

2. LOF local. Usá el mismo dataset pero metiendo un outlier dentro de uno de los blobs (anomalía local). Comparar Isolation Forest vs LocalOutlierFactor(n_neighbors=20): ¿cuál lo agarra?

3. One-Class SVM con escalado. Entrená OneClassSVM(kernel='rbf', nu=0.05) con y sin StandardScaler. Comparar fronteras de decisión.

4. Top-k con score_samples. Usá score_samples de Isolation Forest, ordená ascendente, y devolvé los 10 puntos más anómalos. Inspeccionálos visualmente.

5. Evaluación con labels. Sobre fetch_kddcup99 (subset), entrená Isolation Forest sin usar labels. Después calculá ROC-AUC contra las labels reales (normal. vs ataque). Reportá AUC.

Homework verificable

Notebook con dataset de transacciones sintéticas (montos, hora, comercio): (a) inyectar 2% de transacciones anómalas (montos extremos, horarios raros); (b) entrenar Isolation Forest, LOF y One-Class SVM; (c) generar tabla comparativa con precision@k=20, recall y tiempo de entrenamiento; (d) elegir el ganador y justificar.

Criterio de aceptación: los tres modelos están entrenados sobre los mismos datos escalados, la tabla compara las 3 métricas, y la justificación menciona supuestos (densidad local vs global, escalabilidad).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
LocalOutlierFactor no tiene predict	Por default novelty=False y solo expone fi
One-Class SVM marca todo como outlier (o t	Datos sin escalar — el kernel RBF es ultra

IsolationForest con contamination muy off	Si la fracción real difiere mucho del valo
Elliptic Envelope falla con datos multimod	Asume una sola gaussiana. Fix: cambiá a Is
Evaluar con accuracy sobre clases desbalan	El modelo "todo normal" da 99% accuracy y

Preguntas frecuentes

¿Isolation Forest, LOF o One-Class SVM?

Isolation Forest primero, casi siempre: escala bien, pocos hiperparámetros, robusto en alta dimensión. LOF si sospechás anomalías locales (raros dentro de un cluster denso). One-Class SVM si el dataset es chico/mediano y la frontera "normal" es claramente no lineal — pero estandarizá sí o sí. Elliptic Envelope solo si los datos son unimodales y aproximadamente gaussianos.

¿Cómo elijo contamination?

Si conocés la prevalencia esperada (ej. 1% de fraude), poné ese valor. Si no, dejá 'auto' y después calibrá mirando la distribución de score_samples — buscá un quiebre natural en el histograma.

¿Necesito escalar las features?

Para One-Class SVM y LOF, sí (ambos usan distancias). Para Isolation Forest, no es estrictamente necesario porque parte features con splits — pero no hace daño.

¿Sirve para series de tiempo?

No directamente — estos modelos asumen i.i.d. Para series, mirá descomposición + residuos, ARIMA, Prophet, o modelos específicos como PyOD con ventanas deslizantes.

¿Cómo evalúo si no tengo labels?

Inspección manual del top-k con score_samples. Si hay labels parciales o un sample anotado, ROC-AUC y precision@k. Sin nada de eso, sanity check: ¿los outliers detectados tienen sentido para el negocio?

Referencias

- Géron, cap. 9 § Anomaly Detection.
- sklearn — Novelty and Outlier Detection
- sklearn — IsolationForest
- sklearn — LocalOutlierFactor
- Liu, Ting & Zhou (2008), Isolation Forest, ICDM.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Cierre de la parte

Fin del bundle consolidado de Parte 1 — Machine Learning Clásico · 50 clases.