Clase 088 — Boosting: AdaBoost y Gradient Boosting

Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 7. Duración estimada: 70 min.

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.

Siguiente clase

Clase 089 — XGBoost, LightGBM y CatBoost

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb