# Resumen del capítulo: Potenciación del gradiente

## Ensambles y potenciación

Un **ensamble** es un conjunto de modelos para resolver el mismo problema. La fortaleza de los ensambles es que el error medio de un grupo de modelos es menos significativo que sus errores individuales.

Ya conoces uno de los tipos de modelos de ensamble: el bosque aleatorio. En un bosque aleatorio se promedian los resultados de los alumnos **base** o **débiles** (modelos que componen el ensamble). Los clasificadores base para un bosque aleatorio son árboles de decisión.

Otro enfoque para la construcción de ensambles es la **potenciación**, donde cada modelo posterior considera los errores del anterior y, en la predicción final, los pronósticos de los alumnos básicos. Echa un vistazo:

$$a_N(x) = \sum_{k=1}^{N} \gamma_k b_k(x)$$

donde *aN(x)* es la predicción del ensamble, *N* es el número de alumnos base, *bk(x)* es la predicción del alumno base y γn es la ponderación del modelo.

Por ejemplo, estamos tratando con una tarea de regresión. Tenemos *n* observaciones con características *x* y respuestas correctas *y*. Nuestra tarea es minimizar la función de pérdida *ECM*:

$$\text{ECM}(y, a) = \frac{1}{n}\sum_{i=1}^{n}(a(x_i) - y_i)^2 \longrightarrow \min_{a(x)}$$

Por conveniencia, iguala los pesos del modelo a la unidad:

$$\gamma_k = 1, \text{for all } k = 1, \ldots, N$$

Obtenemos:

$$a_N(x) = \sum_{k=1}^{N} b_k(x)$$

Ahora creamos un conjunto de modelos secuenciales.

Primero, construye al alumno base $b_1$ resolviendo la tarea de minimización:

$$b_1 = \arg\min_{b} \frac{1}{n}\sum_{i=1}^{n}(b(x_i) - y_i)^2$$

El resultado es este ensamble:

$$a_1(x) = b_1(x)$$

Indica el **residuo**. Es la diferencia entre la predicción en el primer paso y las respuestas correctas:

$$e_{1,i} = y_i - b_1(x_i)$$

En el segundo paso, construimos el modelo de esta manera:

$$b_2 = \arg\min_b \frac{1}{n} \sum_{i=1}^{n} (b(x_i) - e_{1,i})^2$$

El ensamble tomará la siguiente forma:

$$a_2(x) = \sum_{k=1}^{2} b_k(x) = b_1(x) + b_2(x)$$

El ensamble tomará la siguiente forma:

$$e_{2,i} = y_i - a_2(x_i) = y_i - \sum_{k=1}^{2} b_k(x_i)$$

En cada paso siguiente, el algoritmo minimiza el error de ensamble del paso anterior.

Vamos a resumir las fórmulas. En el paso *N-1*, el residuo se calcula de la siguiente manera:

$$e_{N-1,i} = y_i - a_{N-1}(x_i)$$

El ensamble en sí se representa como la suma de las predicciones de todos los alumnos base combinados hasta este paso:

$$a_{N-1}(x) = \sum_{k=1}^{N-1} b_k(x)$$

Entonces, en el paso N, el algoritmo elegirá el modelo con el error de ensamble en el paso N-1:

$$b_N(x) = \arg\min_b \frac{1}{n} \sum_{i=1}^{n} (b(x_i) - e_{N-1,i})^2$$

## Potenciación del gradiente

En la potenciación, cada alumno base intenta "empujar" las predicciones del paso anterior hacia las respuestas correctas. Así es como se minimiza la función de pérdida. La potenciación del gradiente hace que el proceso sea aún más eficiente.

Como antes, nuestro objetivo es minimizar la función de pérdida, ¡pero ahora las respuestas del modelo son el argumento a lo largo del cual se realizará el descenso! Y así obtenemos la **potenciación del gradiente**.

Por ejemplo, nuestra función de pérdida es *L(y, a)* y tiene una derivada. Recordemos la fórmula del ensamble:

$$a_N(x) = a_{N-1}(x) + \gamma_N b_N(x)$$

En cada paso, selecciona las respuestas que minimizarán la función:

$$L(y, a(x)) \longrightarrow \min_a$$

Minimiza la función con descenso de gradiente. Para ello, en cada paso, calcula el gradiente negativo de la función de pérdida para la predicción *gN*:

$$g_N(x) = \nabla L(y, a_{N-1}(x) + a)$$

Para impulsar las predicciones hacia las respuestas correctas, el alumno base aprende a predecir *gN*:

$$b_N(x) = \arg\min_b \frac{1}{n} \sum_{i=1}^{N} (b(x_i) - g_N(x_i))^2$$

Obtén el peso para *bN* de la tarea de minimización, iterando varios números:

$$\gamma N = \arg\min_\gamma L(y, a_{N-1}(x) - \gamma b_N(x))$$

Es el coeficiente para el alumno base lo que ayuda a ajustar el ensamble para hacer predicciones lo más precisas posible.

La potenciación de gradiente es adecuada para diferentes funciones de pérdida que tienen derivadas, por ejemplo, el cuadrado medio en una tarea de regresión o logarítmica en una tarea de clasificación binaria.

## Regularización de potenciación del gradiente

La regularización se puede utilizar para reducir el sobreajuste durante la potenciación del gradiente. Si se han reducido los pesos en una regresión lineal, entonces la regularización de potenciación del gradiente es:

1. reducción del tamaño del paso;
2. ajuste de los parámetros del árbol;
3. aleatorización de submuestras para estudiantes base $b_i$

Reduce el tamaño del paso. Revisa la fórmula para calcular predicciones en el paso $N$:

$$a_N(x) = a_{N-1}(x) + \gamma_N b_N(x)$$

Cuando un algoritmo toma pasos que son demasiado grandes, este recuerda rápidamente el conjunto de entrenamiento. Esto da como resultado un sobreajuste del modelo.

Introduce el coeficiente $\eta$,* que controla la **tasa de aprendizaje** y puede usarse para reducir el tamaño del paso:

$$a_N(x) = a_{N-1}(x) + \eta \times \gamma_N b_N(x)$$

El valor de este coeficiente se elige iterando diferentes valores en el rango de 0 a 1. Un valor más pequeño significa un paso más pequeño hacia el gradiente negativo y una mayor exactitud del ensamble. Pero si la tasa de aprendizaje es demasiado baja, el proceso de entrenamiento llevará demasiado tiempo.

La segunda forma de regularizar la potenciación del gradiente es ajustar los parámetros del árbol. Podemos limitar la profundidad del árbol o la cantidad de elementos en cada nodo, probar diferentes valores y ver cómo afecta el resultado. Por ejemplo, limitemos la profundidad de cada árbol a 2. En cada paso de la potenciación, no tendremos que hacer un árbol complejo que se ajuste a todos los residuos y el modelo no estará demasiado sobreajustado.

El tercer método de regularización es trabajar con submuestras. El algoritmo funciona con submuestras en lugar del conjunto completo. Esta versión del algoritmo es similar al DGE y se llama **potenciación del gradiente estocástico.**

## Librerías para potenciación del gradiente

1. XGBoost (potenciación del gradiente extrema) es una librería popular de potenciación del gradiente en Kaggle. Código abierto. Lanzada en 2014.
2. LightGBM (máquina ligera de potenciación del gradiente). Desarrollada por Microsoft. Entrenamiento de potenciación del gradiente rápido y preciso. Funciona directamente con características categóricas. Lanzada en 2017. Comparación con XGBoost: https://lightgbm.readthedocs.io/en/latest/Experiments.html (materiales en inglés).
3. CatBoost (potenciación categórica). Desarrollada por Yandex. Superior a otros algoritmos en términos de métricas de evaluación. Aplica varias técnicas de codificación para características categóricas (LabelEncoding, One-Hot Encoding). Lanzada en 2017. Comparación con XGBoost y LightGBM: benchmark https://catboost.ai/#benchmark (materiales en inglés).

Echemos un vistazo a la librería CatBoost.

Importa `CatBoostClassifier` de la librería y crea un modelo. Ya que tenemos un problema de clasificación, especifica la función de pérdida logística. Haz 10 iteraciones para que no tengamos que esperar demasiado.

```
from catboost import CatBoostClassifier

model = CatBoostClassifier(loss_function="Logloss", iterations=10)
```

Entrena el modelo con el método `fit()`. Además del objetivo y las características, pasa las características categóricas al modelo:

```
# cat_features - características categóricas

model.fit(features_train, target_train, cat_features=cat_features)
```

Cuando tienes muchas iteraciones y no quieres generar información para cada una, usa el argumento `verbose`:

```
model = CatBoostClassifier(loss_function="Logloss", iterations=50)
model.fit(features_train, target_train, cat_features=cat_features, verbose=10)
```

Calcula la predicción con `predict()`:

```
pred_valid = model.predict(features_valid)
```