Clase13 IMA539

Alejandro Ferreira Vergara

May 5, 2023

1 XGBOOST

1.1 ¿Qué aprenderemos hoy?

- Gradient Boosting
- Implementación XGBoost como Modelo de Clasificación

1.2 Gradient Boosting

El término **Gradient Boosting** proviene de la idea de "impulsar" o mejorar un solo modelo débil, combinándolo con una serie de otros modelos débiles para generar un modelo fuerte colectivo. Gradient Boosting es una extensión del método Boosting, donde el proceso de generación aditiva de modelos débiles se formaliza como un algoritmo de descenso de gradiente sobre una función objetivo.

Los algoritmos de Gradient Boosting son de Aprendizaje Supervisado.

Se pueden usar muchos tipos diferentes de modelos (débiles) para en un algoritmo de Gradient Boosting, pero en la práctica casi siempre se usan **Árboles de Decisión**.

Ahora, supongamos que para un modelo de árbol de decisión, calculamos el error de la siguiente manera:

$$r_i = y_i - \widehat{y_i}$$

Para mejorar el modelo anterior, podemos construir otro árbol de decisión, pero esta vez intentando predecir los residuos r_i en lugar de las etiquetas originales. Esto se puede considerar como construir otro modelo para corregir el error en el modelo actual. Luego, agregamos el nuevo árbol al modelo, hacemos nuevas predicciones y luego volvemos a calcular los residuos.

Para hacer predicciones con múltiples árboles, simplemente pasamos la instancia dada (datos) a través de cada árbol y sumamos las predicciones de cada árbol.

Si miramos la **Suma de Errores al Cuadrado (SSE)**, podríamos observar que el error del modelo disminuye a medida que agregamos un árbol (no por siempre). Para explicar este aumento del rendimiento del modelo, tomemos el gradiente de la función SSE:

$$SSE(y_i, \widehat{y_i}) = \frac{1}{2}\sum_i (y_i - \widehat{y_i})^2$$

$$\Rightarrow \frac{\partial SSE(y_i, \widehat{y}_i)}{\partial \widehat{y}_i} = -(y_i - \widehat{y}_i)$$

Por lo tanto, el residuo r_i es el gradiente negativo de la función de pérdida. Es decir, al construir modelos que ajuntan las etiquetas en la dirección de estos residuos, esto es en realidad un algoritmo de **Descenso de gradiente en la función de pérdida de error cuadrático**.

Con esto, se minimiza la función de pérdida para las instancias de entrenamiento hasta que finalmente se alcanza un mínimo local para los datos de entrenamiento.

2 ¿Qué es XGBoost?

XGBoost significa Extreme Gradient Boosting. Es una librería de aprendizaje automático de árbol de decisión impulsado por gradiente (GBDT), distribuida y escalable. Proporciona refuerzo de árboles paralelos y es la librería de aprendizaje automático líder para problemas de regresión y clasificación.

Un GBDT es un algoritmo de aprendizaje en conjunto, similar a Random Forest. Ambos modelos constan de múltiples árboles de descisión. La diferencia está en cómo se combinan.

Recordando, Random Forest utiliza una técnica llamada bagging para construir árboles de decisión completos en paralelo a partir de muestras aleatorias de arranque del conjunto de datos. La predicción final es un promedio de todas las predicciones del árbol de decisión.

Los modelos GBDT entrenan iterativamente un conjunto de árboles de decisión poco profundos, y cada iteración usa los residuos de error del modelo anterior para ajustar el siguiente modelo. La predicción final es una suma ponderada de todas las predicciones del árbol.

XGBoost es una implementación escalable y muy precisa de Gradient Boosting que supera los límites de la potencia informática para los algoritmos de árbol potenciado, y se crea en gran medida para potenciar el rendimiento del modelo de aprendizaje automático y la velocidad computacional. Con XGBoost, los árboles se construyen en paralelo, en lugar de secuencialmente como GBDT. Sigue una estrategia por niveles, escaneando valores de gradiente y usando estas sumas parciales para evaluar la calidad de las divisiones en cada división posible en el conjunto de entrenamiento.

2.1 Algunas características de XGBoost

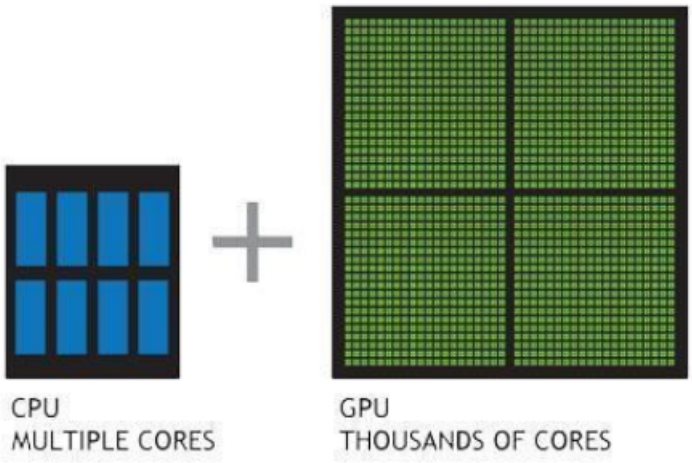
- Inicialmente se construyeron las implementaciones para Python y R. Debido a su popularidad, hoy tiene implementaciones en Java, Scala, Julia, entre otros lenguaje de programación.
- Se ha integrado con una amplia variedad de otras heeramientas y paquetes, como Scikit-learn (Python) y Caret (R). Además está integrado con marcos de trabajo de procesamiento distribuido como Apache Spark y Dask.
- En 2019, XGBoost fue incluido entre los codiciados ganadores del premio Tecnología del año de InfoWorld.
- Los modelos XGBoost tienen la mejor combinación de rendimiento de predicción y tiempo de procesamiento en comparación con otros algoritmos.
- Actualmente se ejecuta en plataformas OS X, Windows y Linux.

- Al utilizar GPU se mejora el rendimiento de un modelo XGBoost.

[1]:
```
from IPython.display import Image

Image(filename=r'clase13/13_1.png', width=300)
```

[1]:



El algoritmo XGBoost acelerado por GPU utiliza operaciones rápidas de suma de prefijos paralelos para escanear todas las divisiones posibles, así como la clasificación de radix paralela para repartir datos. Construye un árbol de decisiones para una iteración de impulso dada, un nivel a la vez, procesando todo el conjunto de datos simultáneamente en la GPU.

El lanzamiento inicial de NVIDIA spark-xgboost permitió el entrenamiento y la inferencia de modelos de aprendizaje automático XGBoost en los nodos de Apache Spark. Esto ha ayudado a convertirlo en un mecanismo líder para el aprendizaje automático distribuido de clase empresarial.

Spark XGBoost acelerado por GPU acelera el preprocesamiento de volúmenes masivos de datos, permite tamaños de datos más grandes en la memoria de GPU y mejora el tiempo de entrenamiento y ajuste de XGBoost.

3 Implementación XGBoost con Python

- Documentación: https://xgboost.readthedocs.io/en/latest/

[ ]:
```
import xgboost as xgb
import numpy as np
from sklearn.datasets import fetch_covtype
from sklearn.model_selection import train_test_split
import time

cov = fetch_covtype()
```

```
X = cov.data
y = cov.target
```

[ ]:
```
X.shape
```

[ ]:
```
np.unique(y)
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪25,train_size=0.75,random_state=42)

dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

num_round = 300
```

[ ]:
```
param = {'objective': 'multi:softmax',
         'num_class': 8,
         'tree_method': 'hist'}

tmp = time.time()
cpu_res = {}
xgb.train(param, dtrain, num_round, evals=[(dtest, 'test')],␣
 ↪evals_result=cpu_res)
print("Tiempo de entrenamiento en CPU: %s segundos" % (str(time.time() - tmp)))
```

[ ]:
```
num_round = 300

param = {'objective': 'multi:softmax',
         'num_class': 8,
         'tree_method': 'gpu_hist'}

gpu_res = {}
tmp = time.time()
xgb.train(param, dtrain, num_round, evals=[(dtest, 'test')],␣
 ↪evals_result=gpu_res)
print("Tiempo de entrenamiento en GPU: %s segundos" % (str(time.time() - tmp)))
```

3.1 XGBoost como modelo de Clasificación

[ ]:
```
from sklearn import datasets
from sklearn import metrics
from sklearn.model_selection import train_test_split
import xgboost as xgb

dataset = datasets.load_wine()

X = dataset.data; y = dataset.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

[ ]:
```
np.unique(y)
```

[ ]:
```
model = xgb.XGBClassifier(n_estimators=2, max_depth=2, learning_rate=1,␣
↪objective='multi:softprob')
model.fit(X_train, y_train)
print(model)
```

[ ]:
```
y_pred =  model.predict(X_test)

print(metrics.classification_report(y_test, y_pred))
print(metrics.confusion_matrix(y_test, y_pred))
```

[ ]:
```

```