Aprende Machine Learning

antes de que sea demasiado tarde

GENERAL, PRÁCTICA

Random Forest, el poder del Ensamble

junio 17 by Na8

Si ya leíste el algoritmo de árbol de Decisión con Aprendizaje Automático, tu próximo paso es el de estudiar Random Forest. **Comprende qué és y cómo funciona** con un ejemplo práctico en Python. Podrás descargar el código de ejemplo en una Jupyter Notebook -como siempre-.

Random Forest es un tipo de Ensamble en Machine Learning en donde combinaremos diversos árboles -ya veremos cómo y con qué características- y la salida de cada uno se contará como «*un voto*» y la opción más votada será la respuesta del <<Bosque Aleatorio>>.

Random Forest, al igual que el árbol e decisión, es un modelo de aprendizaje supervisado para clasificación (aunque también puede usarse para problemas de regresión).

¿Cómo surge Random Forest?

Uno de los problemas que aparecía con la creación de un árbol de decisión es que si le damos la profundidad suficiente, el árbol tiende a «memorizar» las soluciones en vez de generalizar el aprendizaje. Es decir, a padecer de overfitting. La solución para evitar esto es la de crear muchos árboles y que trabajen en conjunto. Veamos cómo.

Cómo funciona Random Forest?

Random Forest funciona así:

- Seleccionamos **k** *features* (columnas) de las **m** totales (siendo **k** menor a **m**) y creamos un árbol de decisión con esas **k** características.
- Creamos **n** árboles variando siempre la cantidad de *k features* y también podríamos variar la cantidad de muestras que pasamos a esos árboles (esto es conocido como «*bootstrap sample*»)
- Tomamos **cada uno** de los **n** árboles y le pedimos que hagan una misma clasificación. Guardamos el resultado de cada árbol obteniendo **n** salidas.
- Calculamos los votos obtenidos para cada «clase» seleccionada y consideraremos a la más votada como la clasificación final de nuestro «bosque».

¿Por qué es aleatorio?

Contamos con una <<doble aleatoriedad>>: tanto en la selección del valor **k** de características para cada árbol como en la cantidad de muestras que usaremos para entrenar cada árbol creado.

Es curioso que para este algoritmo la aleatoriedad sea tan importante y de hecho es lo que lo «hace bueno», pues le brinda flexibilidad suficiente como para poder obtener gran variedad de árboles y de muestras que en su conjunto aparentemente caótico, producen una salida concreta. Darwin estaría orgulloso 😉

Ventajas y Desventajas del uso de Random Forest

Vemos algunas de sus ventajas son:

- funciona bien -aún- sin ajuste de hiperparámetros
- funciona bien para problemas de clasificación y también de regresión.
- al utilizar múltiples árboles **se reduce considerablemente el riesgo de overfiting**
- se mantiene estable con nuevas muestras puesto que al utilizar cientos de árboles sigue prevaleciendo el promedio de sus votaciones.

Y sus desjeventajas:

- en algunos datos de entrada «particulares» random forest también puede caer en overfitting
- es mucho más «costo» de crear y ejecutar que «un sólo árbol» de decisión.
- Puede requerir muchísimo tiempo de entrenamiento
- OJO! Random Forest no funciona bien con datasets pequeños.
- Es muy difícil poder interpretar los ¿cientos? de árboles creados en el bosque, si quisiéramos comprender y explicar a un cliente su comportamiento.

Vamos al Código Python

Continuaremos con el ejercicio propuesto en el artículo «desbalanceo de datos» en donde utilizamos el dataset de Kaggle con información de fraude en tarjetas de crédito. Cuenta con 284807 filas y 31 columnas de características. Nuestra salida será 0 si es un cliente «normal» o 1 si hizo uso fraudulento.

¿Llegas a ver la mínima linea roja que representa los casos de Fraude? son apenas 492 frente a más de 250.000 casos de uso normal.

Retomaremos el mejor caso que obtuvimos en el ejercicio anterior utilizando Regresión Logística y logrando un 98% de aciertos, pero recuerda también las métricas de F1, precisión y recall que eran las que realmente nos ayudaban a validar el modelo.

Requerimientos para hacer el ejercicio Random Forest

Necesitaremos tener instalado Python 3.6 en el sistema y como lo haremos en una Notebook Jupyter, recomiendo tener instalada la suite de Anaconda que simplificará todo.

> ¿Cómo instalar el ambiente de desarrollo Python con Anaconda?

Pues vamos con nuestro Bosque!

Creamos el modelo y lo entrenamos

Utilizaremos el modelo RandomForrestClassifier de SkLearn.

```
from sklearn.ensemble import RandomForestClassifier

# Crear el modelo con 100 arboles
model = RandomForestClassifier(n_estimators=100,
                               bootstrap = True, verbose=2,
                               max_features = 'sqrt')
# a entrenar!
model.fit(X_train, y_train)
```

Luego de unos minutos obtendremos el modelo entrenado (en mi caso 1 minuto 30 segundos)

Los Hiperparámetros más importantes

Al momento de ajustar el modelo, debemos tener en cuenta los siguientes hiperparámetros. Estos nos ayudarán a que el bosque de mejores resultados para cada ejercicio. Recuerda que esto no se trata de «copiar y pegar»!

- **n_estimators**: será la cantidad de árboles que generaremos.
- **max_features**: la manera de seleccionar la cantidad máxima de features para cada árbol.
- **min_sample_leaf**: número mínimo de elementos en las hojas para permitir un nuevo split (división) del nodo.
- **oob_score**: es un método que emula el cross-validation en árboles y permite mejorar la precisión y evitar overfitting.
- **boostrap**: para utilizar diversos tamaños de muestras para entrenar. Si se pone en falso, utilizará siempre el dataset completo.
- **n_jobs**: si tienes multiples cores en tu CPU, puedes indicar cuantos puede usar el modelo al entrenar para acelerar el entrenamiento.

Evaluamos resultados

Veamos la matriz de confusión y las métricas sobre el conjunto de test!!! (no confundir con el de training!!!)

Vemos muy buenos resultados, clasificando con error apenas 11 + 28 muestras.

Aquí podemos destacar que para la clase «minoritaria», **es decir la que detecta los casos de fraude** tenemos un buen valor de recall (de 0.80) lo cual es un buen indicador! y el F1-score macro avg es de 0.93. Logramos construir un modelo de Bosque aleatorio que a pesar de tener un conjunto de datos de entrada muy desigual, logra buenos resultados.

Comparamos con el Baseline

Si comparamos estos resultados con los del algoritmo de Regresión Logística, vemos que el Random Forest nos dio mejores clasificaciones, menos falsos positivos y mejores métricas en general.

Conclusiones

Avanzando en nuestro aprendizaje sobre diversos modelos que podemos aplicar a las problemáticas que nos enfrentamos, hoy sumamos a nuestro kit de herramientas el Random Forest, vemos que es un modelo sencillo, bastante rápido y si bien perdemos la interpretabilidad maravillosa que nos brindaba 1 sólo árbol de decisión, es el precio a pagar para evitar el overfitting y para ganar un clasificador más robusto.

Los algoritmos Tree-Based -en inglés- son muchos, todos parten de la idea principal de árbol de decisión y la mejoran con diferentes tipos de ensambles y técnicas. Tenemos que destacar a 2 modelos que según el caso logran superar a las mismísimas redes neuronales! son XGboost y LightGBM. Si te parecen interesantes puede que en el futuro escribamos sobre ellos.

Suscribete al blog

Recibe los próximos artículos sobre Machine Learning, estrategias, teoría y código Python en tu casilla de correo!

Email: Your email address here

ENVIAR

NOTA: algunos usuarios reportaron que el email de confirmación y/o posteriores a la suscripción entraron en su carpeta de SPAM. Te sugiero que revises y que agregues nuestro remitente a tus contactos para evitar problemas. Gracias!

Recursos y Adicionales

Puedes descargar la notebook para este ejercicio desde mi cuenta de GitHub:

- Código en jupyter Notebook en GitHub
- Dataset de Kaggle

Otros artículos sobre Random Forest en inglés:

- Random Forest Simple Explanation
- An Implementation of Random Forest in Python

Comparte el artículo:

Relacionado

12 Consejos útiles para aplicar Machine Learning

Principales Algoritmos usados en Machine Learning

Arbol de Decisión en Python: Clasificación y predicción.

Algoritmos | Aprendizaje Automático | ejemplo | Ejercicio | Modelos

CLASIFICACIÓN CON DATOS DESBALANCEADOS

TU PROPIO SERVICIO DE MACHINE LEARNING

Deja un comentario

Introduce aquí tu comentario...

Visita nuestra Guía de Aprendizaje

Buscar

Search ...

Contacto

Suscripción

Recibe los artículos de Aprende Machine Learning en tu casilla de correo. Cada 15 días y sin Spam!

Email: Your email address here

ENVIAR