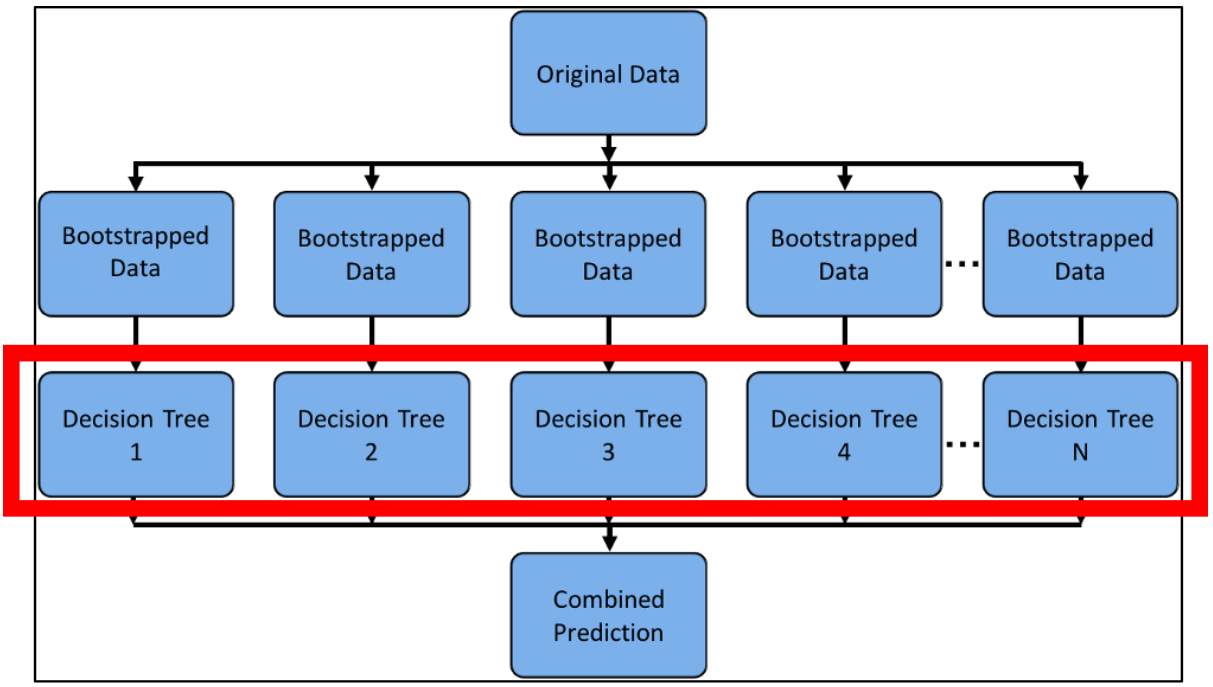
Bosques aleatorios

Objetivos de aprendizaje
Al final de este módulo, serán capaces de:

- Describir qué son los bosques aleatorios y cómo funcionan



The area enclosed in the red rectangle is where the random forest algorithm differs from the bagged tree algorithm.

Los bosques aleatorios ofrecen una variación leve en los bagged trees. La diferencia es que cuando se crea cada árbol, cada vez que una división se considera, se elige una muestra aleatoria de características m como candidatos de división del conjunto completo de las características p. Se le permite a la división usar una de aquellas características m. Esto sucede para cada árbol en cada división.

¿Por qué bosques aleatorios en vez de bagged trees?

Un problema con bagged trees es que puede haber una característica muy fuerte en el conjunto de datos que es muy predictivo el mismo. Cuando se usen bagged trees, incluso con datos de bootstrapping, es probable que tendrán a la mayoría de sus árboles utilizando esa única característica como la división superior (nodo raíz), dando como resultado en un conjunto de árboles que se correlacionan. Si intentan y promedian las cantidades altamente correlacionadas, no reducirá la varianza, el cuál es un objetivo fundamental de bagged trees. Si no utilizaron datos de bootstrapping en bagged trees, tendrán árboles

extremadamente correlacionados y a menudo árboles idénticos, lo que frustraría el propósito de usar bagged trees. Para los bosques aleatorios, cada vez que seleccionan aleatoriamente un subconjunto de características para cada división, están tratando de correlacionar los árboles y así reducir la varianza del modelo.

Lecturas opcionales

A fondo: árboles de decisión y bosques aleatorios: Esta lectura del manual de ciencia de datos en Python es útil si quieren lectura complementaria.
Cuidado con las importaciones de bosques aleatorios por defecto: Encontrar qué características (columnas) de datos del conjunto de datos contribuyen más a una predicción suele denominar importancia de la característica. El problema es que no conocer cómo se calculan las cosas puede causar algunos problemas. Si alguna vez mencionan los bosques aleatorios en entrevistas, una buena idea es conocer cómo se pueden calcular las importancias de las características.

Bosques aleatorios para la regresión en Python

Objetivos de aprendizaje

Al final de este módulo, serán capaces de:

- Crear un modelo de bosque aleatorio utilizando Python
- Afinar n_estimators (# de los árboles de decisión)

Tarea

La tarea en esta clase será predecir los precios de las casas usando un modelo de bosque aleatorio.
Importen las bibliotecas

```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
# Importen un regresor de bosque aleatorio
from sklearn.ensemble import RandomForestRegressor
```

Carguen el conjunto de datos
Utilizaremos el conjunto de datos de casas en California que se usaron cuando aprendimos sobre los árboles de decisión.

```
df = pd.read_csv('YOUR PATH')
df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Ordenen los datos en una matriz de características y un vector objetivo y un train-test-split

```
# Ordenen los datos en matriz de características y vector objetivoy = df['MedHouseVal']
X = df.drop(columns = 'MedHouseVal')
# Dividan los datos para la validación
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

Paso 1: importar el modelo que quieran usar
En sklearn, todos los modelos de aprendizaje automático se implementan como clases de Python

```
# Esto ya se importó antes, así que lo comentamos
from sklearn.ensemble import RandomForestRegressor
```

Paso 2: hacer una instancia del modelo
Aquí es donde podemos afinar los hiperparámetros del modelo. Por ahora utilizaremos los parámetros por defecto. Pueden observar que esto será Max_depth = None y n_estimators = 100. Estos son solo algunos de los parámetros importantes a explorar.

```
rf = RandomForestRegressor(random_state = 42)
```

```
# Looking at some hyperparameters that seem tunable
rf.get_params()
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

Paso 3: entrenar el modelo en los datos, almacenando la información aprendida de los datos.
El modelo está aprendiendo la relación entre X e y. ¡Esto puede tomar un tiempo para ejecutar!

```
rf.fit(X_train, y_train)
```

Paso 4: predecir los valores para y (recuerden que este paso permite ver las predicciones reales, pero no es necesario para evaluar o afinar el modelo).

```
rf.predict(X_test)
```

Paso 5: evaluar el rendimiento del modelo

```
rf_train_score = rf.score(X_train, y_train)
rf_test_score = rf.score(X_test, y_test)
print(rf_train_score)
print(rf_test_score)
```

```
0.972618186878301
0.8080212525635748
```

Lo primero que debemos observar es cuánto el modelo de bosque aleatorio mejoró el valor R2 comparado con nuestro sencillo árbol de decisión. En nuestra clase anterior, el mejor valor R2 que pudimos alcanzar en nuestro conjunto de prueba fue de solo 0,691 como se muestra en la salida final la clase “Árboles de regresión en Python”.

```
0.7961670169616584
0.6912777375475382
```

Paso 6: afinar los modelos

Ajustar max_depth

Como lo hicimos anteriormente, podemos afinar max_depth que permitimos para cada árbol en nuestro bosque aleatorio. Establezcamos max_depth a 9 y evaluemos nuestros resultados.

```
rf_9 = RandomForestRegressor(max_depth = 9, random_state = 42)
rf_9.fit(X_train, y_train)
rf_9_train_score = rf_9.score(X_train, y_train)
rf_9_test_score = rf_9.score(X_test, y_test)
print(rf_9_train_score)
print(rf_9_test_score)
```

```
0.8423182891263877
0.76602214456465
```

Tengan en cuenta que si bien nuestros resultados con un max_depth de 9 fueron óptimos para cada árbol, este NO es el caso para el bosque aleatorio. (Nuestra puntuación de prueba disminuyó)

Para ver cuál era la profundidad de cada árbol en el bosque aleatorio cuando el max_depth era ilimitado, pueden usar el siguiente código:

```
[estimator.get_depth() for estimator in rf.estimators_]
```

Para ahorrar espacio, la salida no se muestra, sin embargo, observen que la profundidad de cada árbol varía.
Pueden intentar diferentes valores para max_depth u otros parámetros para ver si pueden hacer mejores en el modelo por defecto.

Afinar n_estimators (# de los árboles de decisión)

Otro ajuste de parámetro es n_estimators, el cual representa el número de los árboles que deben crecer. Puede que se demore el código de abajo en ejecutar. La razón es que cuando entrenan un conjunto, están entrenando más de un modelo (en este caso, un árbol). Observemos si podemos mejorar la puntuación al doblar la cantidad de árboles de 100 a 200.

```
# Intenten 200 árboles
rf_200 = RandomForestRegressor(n_estimators = 200, random_state = 42)
# Ajusten el modelo
rf_200.fit(X_train, y_train)
# Obtengan las puntuaciones
rf_200_train_score = rf_200.score(X_train, y_train)
rf_200_test_score = rf_200.score(X_test, y_test)
print(rf_200_train_score)
print(rf_200_test_score)
```

```
0.973629451624732
0.8091204656172442
```

En este caso, apenas teníamos una mejora perceptible con 200 árboles comparado con los 100 árboles por defecto. Esto dependerá de sus datos.