Skills Network

(https://skills.network/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_ic
SkillsNetwork-Channel-SkillsNetworkCoursesIBMML241ENSkillsNetwork820-2023-01-01)

Random Forests (RF) for classification with Python

Estimated time needed: **45** minutes

Objectives

After completing this lab you will be able to:

- Understand the difference between Bagging and Random Forest
- Understand that Random Forests have less Correlation between predictors in their ensemble, improving accuracy
- Apply Random Forest
- Understand Hyperparameters selection in Random Forest

In this notebook, you will learn Random Forests (RF) for classification and Regression. Random Forest is similar to Bagging using multiple model versions and aggregating the ensemble of models to make a single prediction. RF uses an ensemble of tree's and introduces randomness into each tree by randomly selecting a subset of the features for each node to split on. This makes the predictions of each tree uncorrelated, improving results when the models are aggregated. In this lab we will illustrate the sampling process of RF to Bagging, then demonstrate how each predictor for random forest are not correlated. Finally, we will apply Random Forests to several datasets using Grid-Search to find the optimum Hyperparameters.

Table of contents

Let's first import the required libraries:

In [146]:

```
# All Libraries required for this lab are listed below. The libraries pre-installed on Si
# !mamba install -qy pandas==1.3.3 numpy==1.21.2 ipywidgets==7.4.2 scipy==7.4.2 tqdm==4.
# Note: If your environment doesn't support "!mamba install", use "!pip install"
```

In [147]:

```
import pandas as pd
import pylab as plt
import numpy as np
import scipy.optimize as opt
from sklearn import preprocessing
%matplotlib inline
import matplotlib.pyplot as plt
from sklearn import metrics
from tqdm import tqdm
```

Ignore error warnings

In [148]:

```
import warnings
warnings.filterwarnings('ignore')
```

This function will calculate the accuracy of the training and testing data given a model.

In [149]:

```
def get_accuracy(X_train, X_test, y_train, y_test, model):
    return  {"test Accuracy":metrics.accuracy_score(y_test, model.predict(X_test)),"tria
```

This function calculates the average correlation between predictors and displays the pairwise correlation between predictors.

In [150]:

```

def get_correlation(X_test, y_test,models):
    #This function calculates the average correlation between predictors
    n_estimators=len(models.estimators_)
    prediction=np.zeros((y_test.shape[0],n_estimators))
    predictions=pd.DataFrame({'estimator '+str(n+1):[] for n in range(n_estimators)})

    for key,model in zip(predictions.keys(),models.estimators_):
        predictions[key]=model.predict(X_test.to_numpy())

    corr=predictions.corr()
    print("Average correlation between predictors: ", corr.mean().mean()-1/n_estimators)
    return corr
```

What's the difference between RF and Bagging

RF is similar to Bagging in that it uses model ensembles to make predictions. Like Bagging it if you add more models, RF does not suffer from Overfitting. In this section, we go over some of the differences between RF and Bagging, using the dataset:

About the dataset

We will use a telecommunications dataset for predicting customer churn. This is a historical customer dataset where each row represents one customer. The data is relatively easy to understand, and you may uncover insights you can use immediately. Typically, it is less expensive to keep customers than acquire new ones, so the focus of this analysis is to predict the customers who will stay with the company.

This data set provides information to help you predict what behavior will help you to retain customers. You can analyze all relevant customer data and develop focused customer retention programs.

The dataset includes information about:

- Customers who left within the last month – the column is called Churn
- Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies
- Customer account information – how long they had been a customer, contract, payment method, paperless billing, monthly charges, and total charges
- Demographic info about customers – gender, age range, and if they have partners and dependents

Load Data From CSV File

In [151]:

```
churn_df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.clo

churn_df.head()
```

Out[151]:

```
   tenure   age  address  income   ed  employ  equip  callcard  wireless  longmon  ...  page
0    11.0  33.0      7.0   136.0  5.0     5.0    0.0       1.0       1.0     4.40  ...    1.
1    33.0  33.0     12.0    33.0  2.0     0.0    0.0       0.0       0.0     9.45  ...    0.
2    23.0  30.0      9.0    30.0  1.0     2.0    0.0       0.0       0.0     6.30  ...    0.
3    38.0  35.0      5.0    76.0  2.0    10.0    1.0       1.0       1.0     6.05  ...    1.
4     7.0  35.0     14.0    80.0  2.0    15.0    0.0       1.0       0.0     7.10  ...    0.

5 rows × 28 columns
```

Data pre-processing and selection

Let's select some features for the modeling. Also, we change the target data type to be an integer, as it is a requirement by the skitlearn algorithm:

In [152]:

```
churn_df = churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip', 'cal
churn_df['churn'] = churn_df['churn'].astype('int')
churn_df.head()
```

Out[152]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	1
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	1
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	0

Bootstrap Sampling

Bootstrap Sampling is a method that involves drawing of sample data repeatedly with replacement from a data source to estimate a model parameter. Scikit-learn has methods for Bagging but its helpful to understand Bootstrap sampling. We will import `resample`

In [153]:

```
from sklearn.utils import resample
```

Consider the five rows of data:

In [154]:

```
churn_df[0:5]
```

Out[154]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	1
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	1
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	0

We can perform a bootstrap sample using the function `resample` ; we see the dataset is the same size, but some rows are repeated:

In [155]:

```
for n in range(5):

    print(resample(churn_df[0:5]))
```

```
   tenure   age  address  income   ed  employ  equip  callcard  wireless  \
2    23.0  30.0      9.0    30.0  1.0     2.0    0.0       0.0       0.0
2    23.0  30.0      9.0    30.0  1.0     2.0    0.0       0.0       0.0
3    38.0  35.0      5.0    76.0  2.0    10.0    1.0       1.0       1.0
3    38.0  35.0      5.0    76.0  2.0    10.0    1.0       1.0       1.0
1    33.0  33.0     12.0    33.0  2.0     0.0    0.0       0.0       0.0

   churn
2      0
2      0
3      0
3      0
1      1
   tenure   age  address  income   ed  employ  equip  callcard  wireless  \
3    38.0  35.0      5.0    76.0  2.0    10.0    1.0       1.0       1.0
3    38.0  35.0      5.0    76.0  2.0    10.0    1.0       1.0       1.0
2    23.0  30.0      9.0    30.0  1.0     2.0    0.0       0.0       0.0
1    33.0  33.0     12.0    33.0  2.0     0.0    0.0       0.0       0.0
1    33.0  33.0     12.0    33.0  2.0     0.0    0.0       0.0       0.0

   churn
3      0
3      0
2      0
1      1
1      1
   tenure   age  address  income   ed  employ  equip  callcard  wireless  \
3    38.0  35.0      5.0    76.0  2.0    10.0    1.0       1.0       1.0
3    38.0  35.0      5.0    76.0  2.0    10.0    1.0       1.0       1.0
3    38.0  35.0      5.0    76.0  2.0    10.0    1.0       1.0       1.0
2    23.0  30.0      9.0    30.0  1.0     2.0    0.0       0.0       0.0
2    23.0  30.0      9.0    30.0  1.0     2.0    0.0       0.0       0.0

   churn
3      0
3      0
3      0
2      0
2      0
   tenure   age  address  income   ed  employ  equip  callcard  wireless  \
0    11.0  33.0      7.0   136.0  5.0     5.0    0.0       1.0       1.0
4     7.0  35.0     14.0    80.0  2.0    15.0    0.0       1.0       0.0
3    38.0  35.0      5.0    76.0  2.0    10.0    1.0       1.0       1.0
1    33.0  33.0     12.0    33.0  2.0     0.0    0.0       0.0       0.0
0    11.0  33.0      7.0   136.0  5.0     5.0    0.0       1.0       1.0

   churn
0      1
4      0
3      0
1      1
0      1
   tenure   age  address  income   ed  employ  equip  callcard  wireless  \
1    33.0  33.0     12.0    33.0  2.0     0.0    0.0       0.0       0.0
4     7.0  35.0     14.0    80.0  2.0    15.0    0.0       1.0       0.0
4     7.0  35.0     14.0    80.0  2.0    15.0    0.0       1.0       0.0
```

```
2    23.0  30.0      9.0    30.0  1.0    2.0   0.0      0.0      0.0
2    23.0  30.0      9.0    30.0  1.0    2.0   0.0      0.0      0.0

   churn
1      1
4      0
4      0
2      0
2      0
```

Select Variables at Random

Like Bagging, RF uses an independent bootstrap sample from the training data. In addition, we select m variables at random out of all M possible variables. Let's do an example.

In [156]:

```
X=churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip']]
```

there are 7 features

In [157]:

```
M=X.shape[1]
M
```

Out[157]:

```
7
```

Let us select $m = 3$, and randomly sample features from the 5 Bootstrap Samples from above.

In [158]:

```
m=3
```

We list out the index of the features

In [159]:

```
feature_index= range(M)
feature_index
```

Out[159]:

```
range(0, 7)
```

We can use the function to sample to randomly select indexes

In [160]:

```
import random
random.sample(feature_index,m)
```

Out[160]:

```
[6, 2, 3]
```

We now randomly select features from the bootstrap samples, in randomly selecting a subset of the features for each node to split on.

In [161]:

```
for n in range(5):

    print("sample {}".format(n))
    print(resample(X[0:5]).iloc[:,random.sample(feature_index,m)])
```

```
sample 0
   address  income  equip
2      9.0    30.0    0.0
4     14.0    80.0    0.0
4     14.0    80.0    0.0
4     14.0    80.0    0.0
3      5.0    76.0    1.0
sample 1
   employ   age  equip
1     0.0  33.0    0.0
1     0.0  33.0    0.0
1     0.0  33.0    0.0
2     2.0  30.0    0.0
4    15.0  35.0    0.0
sample 2
   income  address  tenure
0   136.0      7.0    11.0
1    33.0     12.0    33.0
4    80.0     14.0     7.0
1    33.0     12.0    33.0
1    33.0     12.0    33.0
sample 3
   equip  employ  tenure
0    0.0     5.0    11.0
2    0.0     2.0    23.0
0    0.0     5.0    11.0
2    0.0     2.0    23.0
2    0.0     2.0    23.0
sample 4
    age  employ  income
2  30.0     2.0    30.0
4  35.0    15.0    80.0
0  33.0     5.0   136.0
2  30.0     2.0    30.0
1  33.0     0.0    33.0
```

In Random Forest, we would use these data subsets to train each node of a tree.

Train/Test dataset

Let's define X, and y for our dataset:

In [162]:

```
y = churn_df['churn']
y.head()
```

Out[162]:

```
0    1
1    1
2    0
3    0
4    0
Name: churn, dtype: int32
```

Train/Test dataset

We split our dataset into train and test set:

In [163]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3, random_state=1
print ('Train set', X_train.shape,  y_train.shape)
print ('Test set', X_test.shape,  y_test.shape)
```

```
Train set (140, 7) (140,)
Test set (60, 7) (60,)
```

Bagging Review

In [164]:

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
```

Bagging improves models that suffer from overfitting; they do well on the training data, but they do not Generalize well. Decision Trees are a prime candidate for this reason, in addition, they are fast to train; We create a `BaggingClassifier` object, with a Decision Tree as the `base_estimator` .

In [165]:

```
n_estimators=20
Bag= BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion="entropy", max_de
```

We fit the model:

In [166]:

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

Out[166]:

```
▸           BaggingClassifier
▸ base_estimator: DecisionTreeClassifier
        ▸ DecisionTreeClassifier
```

The method `predict` aggregates the predictions by voting:

In [167]:

```
Bag.predict(X_test).shape
```

Out[167]:

```
(60,)
```

We see the training accuracy is slightly better but the test accuracy improves dramatically:

In [168]:

```
print(get_accuracy(X_train, X_test, y_train, y_test,  Bag))
```

```
{'test Accuracy': 0.7333333333333333, 'trian Accuracy': 0.907142857142857
1}
```

Each tree is similar; we can see this by plotting the correlation between each tree and the average correlation.

In [169]:

```
get_correlation(X_test, y_test,Bag).style.background_gradient(cmap='coolwarm')
```

```
Average correlation between predictors:  0.25400671537472364
```

Out[169]:

	estimator 1	estimator 2	estimator 3	estimator 4	estimator 5	estimator 6	estimator 7	estimat
estimator 1	1.000000	-0.057709	0.152641	0.132379	0.068323	0.195047	0.209679	0.2561
estimator 2	-0.057709	1.000000	-0.002979	0.335171	0.349647	0.121829	-0.078409	0.0135
estimator 3	0.152641	-0.002979	1.000000	0.395985	-0.010903	0.342381	0.455239	0.6743
estimator 4	0.132379	0.335171	0.395985	1.000000	0.456572	0.242393	0.436809	0.4276
estimator 5	0.068323	0.349647	-0.010903	0.456572	1.000000	0.362231	-0.011036	0.0908
estimator 6	0.195047	0.121829	0.342381	0.242393	0.362231	1.000000	0.198030	0.3706
estimator 7	0.209679	-0.078409	0.455239	0.436809	-0.011036	0.198030	1.000000	0.4746
estimator 8	0.256111	0.013546	0.674356	0.427623	0.090878	0.370625	0.474619	1.0000
estimator 9	0.177811	0.180022	0.442603	0.417131	0.002915	0.183073	0.564524	0.5466
estimator 10	0.318511	0.223814	0.359425	0.494783	0.409514	0.163299	0.404226	0.4640
estimator 11	-0.024845	0.451486	-0.092675	0.051331	0.347826	0.195047	-0.121393	-0.0743
estimator 12	0.318511	-0.074605	0.519170	0.415618	-0.045502	0.244949	0.619813	0.6254
estimator 13	0.209679	-0.078409	0.552099	0.340807	0.099322	0.198030	0.738562	0.4746
estimator 14	0.112611	0.404443	0.296511	0.405843	0.434355	0.505181	0.323942	0.2567
estimator 15	0.294475	0.246580	0.324850	0.224442	0.294475	0.605983	0.148803	0.2838
estimator 16	-0.035245	0.481571	0.216541	0.199294	0.387699	0.158114	0.062622	0.1406
estimator 17	0.161491	0.044130	0.561502	0.375523	0.068323	0.529414	0.540752	0.6691
estimator 18	0.161491	0.044130	0.479730	0.294475	0.161491	0.195047	0.430394	0.5039
estimator 19	0.236433	0.215365	0.415029	0.445634	0.315244	0.494975	0.140028	0.4542
estimator 20	0.015456	-0.059131	0.006783	0.194960	-0.100465	-0.069338	0.247156	0.0205

It can be shown that this correlation reduces performance. Random forest minimizes the correlation between trees, improving results.

Random Forest

Random forests are a combination of trees such that each tree depends on a random subset of the features and data. As a result, each tree in the forest is different and usually performs better than Bagging. The most important parameters are the number of trees and the number of features to sample. First, we import `RandomForestClassifier`.

In [170]:

```
from sklearn.ensemble import RandomForestClassifier
```

Like Bagging, increasing the number of trees improves results and does not lead to overfitting in most cases; but the improvements plateau as you add more trees. For this exxample, the number of trees in the forest (default=100):

In [171]:

```
n_estimators=20
```

`max_features` m the number of features to consider when looking for the best split. If we have M features denoted by:

In [172]:

```
M_features=X.shape[1]
```

If we have M features, a popular method to determine m is to use the square root of M

$$m = floor(\sqrt{M})$$

In [173]:

```
max_features=round(np.sqrt(M_features))-1
max_features
```

Out[173]:

```
2
```

In [174]:

```
y_test
```

```
Out[174]:
```

```
58     1
40     0
34     0
102    0
184    0
198    1
95     1
4      0
29     0
168    0
171    1
18     0
11     0
89     0
110    0
118    0
159    1
39     0
136    1
59     0
51     0
16     0
44     0
94     0
31     0
162    0
38     0
28     0
193    0
27     0
47     0
165    0
194    0
177    0
176    0
97     1
174    1
73     0
69     0
172    1
108    0
107    1
189    0
14     0
56     0
19     1
114    0
39     0
185    0
124    0
98     0
123    1
119    0
53     0
33     1
179    0
181    0
106    0
199    0
```

We use floor to make sure m is an integer:

We create the RF object :

In [1750]:

```
model = RandomForestClassifier( max_features=max_features,n_estimators=n_estimators, ran
```

We train the model

In [197]:

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

Out[197]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

We obtain the training and testing accuracy; we see that RF does better than Bagging:

In [198]:

```
print(get_accuracy(X_train, X_test, y_train, y_test, model))
```

```
{'test Accuracy': 0.9708029197080292, 'trian Accuracy': 1.0}
```

We see that each tree in RF is less correlated than Bagging:

In [199]:

```
get_correlation(X_test, y_test,model).style.background_gradient(cmap='coolwarm')
```

```
Average correlation between predictors:  0.8856492760073719
```

Out[199]:

	estimator 1	estimator 2	estimator 3	estimator 4	estimator 5	estimator 6	estimator 7	estimator 8	estimator 9
estimator 1	1.000000	0.921711	0.873383	0.892130	0.825489	0.937849	0.841533	0.860999	0.889364
estimator 2	0.921711	1.000000	0.858702	0.907470	0.874795	0.923256	0.827337	0.907470	0.906602
estimator 3	0.873383	0.858702	1.000000	0.830301	0.790647	0.838539	0.773745	0.830301	0.855103
estimator 4	0.892130	0.907470	0.830301	1.000000	0.846626	0.863341	0.863341	0.938455	0.909765
estimator 5	0.825489	0.874795	0.790647	0.846626	1.000000	0.821066	0.886181	0.909765	0.902837

```
138    1
Name: churn, dtype: int32
```

Cancer Data Example

The example is based on a dataset that is publicly available from the UCI Machine Learning Repository (Asuncion and Newman, 2007)[http://mlearn.ics.uci.edu/MLRepository.html (http://mlearn.ics.uci.edu/MLRepository.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_ic SkillsNetwork-Channel-SkillsNetworkCoursesIBMML241ENSkillsNetwork31576874-2022-01-01)]. The dataset consists of several hundred human cell sample records, each of which contains the values of a set of cell characteristics. The fields in each record are:

Field name	Description
ID	Clump thickness
Clump	Clump thickness
UnifSize	Uniformity of cell size
UnifShape	Uniformity of cell shape
MargAdh	Marginal adhesion
SingEpiSize	Single epithelial cell size
BareNuc	Bare nuclei
BlandChrom	Bland chromatin
NormNucl	Normal nucleoli
Mit	Mitoses
Class	Benign or malignant

Let's load the dataset:

In [200]:

```
df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM

df.head()
```

Out[200]:

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	Nor
0	1000025	5	1	1	1	2	1	3	
1	1002945	5	4	4	5	7	10	3	
2	1015425	3	1	1	1	2	2	3	
3	1016277	6	8	8	1	3	4	3	
4	1017023	4	1	1	3	2	1	3	

Now lets remove rows that have a ? in the `BareNuc` column:

In [201]:

```
df= df[pd.to_numeric(df['BareNuc'], errors='coerce').notnull()]
```

We obtain the features:

In [202]:

```
X =  df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize', 'BareNuc', 'BlandCh

X.head()
```

Out[202]:

	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mi
0	5	1	1	1	2	1	3	1	
1	5	4	4	5	7	10	3	2	
2	3	1	1	1	2	2	3	1	
3	6	8	8	1	3	4	3	7	
4	4	1	1	3	2	1	3	1	

We obtain the class labels:

In [203]:

```
y=df['Class']
y.head()
```

Out[203]:

```
0    2
1    2
2    2
3    2
4    2
Name: Class, dtype: int64
```

We split the data into training and testing sets.

In [204]:

```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (546, 9) (546,)
Test set: (137, 9) (137,)
```

We use `GridSearchCV` to search over specified parameter values of the model.

In [205]:

```python
from sklearn.model_selection import GridSearchCV
```

We create a `RandomForestClassifier` object and list the parameters using the method `get_params()` :

In [206]:

```python
model = RandomForestClassifier()
model.get_params().keys()
```

Out[206]:

```
dict_keys(['bootstrap', 'ccp_alpha', 'class_weight', 'criterion', 'max_dep
th', 'max_features', 'max_leaf_nodes', 'max_samples', 'min_impurity_decrea
se', 'min_samples_leaf', 'min_samples_split', 'min_weight_fraction_leaf',
'n_estimators', 'n_jobs', 'oob_score', 'random_state', 'verbose', 'warm_st
art'])
```

We can use GridSearch for Exhaustive search over specified parameter values. We see many of the parameters are similar to Classification trees; let's try a different parameter for `max_depth` , `max_features` and `n_estimators` .

In [207]:

```python
param_grid = {'n_estimators': [2*n+1 for n in range(20)],
             'max_depth' : [2*n+1 for n in range(10) ],
             'max_features':["auto", "sqrt", "log2"]}
```

We create the Grid Search object and fit it:

In [208]:

```python
search = GridSearchCV(estimator=model, param_grid=param_grid,scoring='accuracy')
search.fit(X_train, y_train)
```

Out[208]:

```
▸            GridSearchCV
▸ estimator: RandomForestClassifier
       ▸ RandomForestClassifier
```

We can see the best accuracy score of the searched parameters was ~77%.

In [209]:

```python
search.best_score_
```

Out[209]:

```
0.9780817347789826
```

The best parameter values are:

In [210]:

```
search.best_params_
```

Out[210]:

```
{'max_depth': 3, 'max_features': 'sqrt', 'n_estimators': 15}
```

We can calculate accuracy on the test data using the test data:

In [211]:

```

print(get_accuracy(X_train, X_test, y_train, y_test, search.best_estimator_))
```

```
{'test Accuracy': 0.9708029197080292, 'trian Accuracy': 0.979853479853479
8}
```

Practice

Imagine that you are a medical researcher compiling data for a study. You have collected data about a set of patients, all of whom suffered from the same illness. During their course of treatment, each patient responded to one of 5 medications, Drug A, Drug B, Drug c, Drug x and y.

Part of your job is to build a model to find out which drug might be appropriate for a future patient with the same illness. The features of this dataset are Age, Sex, Blood Pressure, and the Cholesterol of the patients, and the target is the drug that each patient responded to.

It is a sample of multiclass classifier, and you can use the training part of the dataset to build a decision tree, and then use it to predict the class of a unknown patient, or to prescribe a drug to a new patient.

In [212]:

```
df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM
df.head()
```

Out[212]:

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	F	HIGH	HIGH	25.355	drugY
1	47	M	LOW	HIGH	13.093	drugC
2	47	M	LOW	HIGH	10.114	drugC
3	28	F	NORMAL	HIGH	7.798	drugX
4	61	F	LOW	HIGH	18.043	drugY

Let's create the X and y for our dataset:

In [213]:

```
X = df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na_to_K']].values
X[0:5]
```

Out[213]:

```
array([[23, 'F', 'HIGH', 'HIGH', 25.355],
       [47, 'M', 'LOW', 'HIGH', 13.093],
       [47, 'M', 'LOW', 'HIGH', 10.114],
       [28, 'F', 'NORMAL', 'HIGH', 7.798],
       [61, 'F', 'LOW', 'HIGH', 18.043]], dtype=object)
```

In [214]:

```
y = df["Drug"]
y[0:5]
```

Out[214]:

```
0    drugY
1    drugC
2    drugC
3    drugX
4    drugY
Name: Drug, dtype: object
```

Now lets use a `LabelEncoder` to turn categorical features into numerical:

In [215]:

```
from sklearn import preprocessing
le_sex = preprocessing.LabelEncoder()
le_sex.fit(['F','M'])
X[:,1] = le_sex.transform(X[:,1])


le_BP = preprocessing.LabelEncoder()
le_BP.fit([ 'LOW', 'NORMAL', 'HIGH'])
X[:,2] = le_BP.transform(X[:,2])


le_Chol = preprocessing.LabelEncoder()
le_Chol.fit([ 'NORMAL', 'HIGH'])
X[:,3] = le_Chol.transform(X[:,3])

X[0:5]
```

Out[215]:

```
array([[23, 0, 0, 0, 25.355],
       [47, 1, 1, 0, 13.093],
       [47, 1, 1, 0, 10.114],
       [28, 0, 2, 0, 7.798],
       [61, 0, 1, 0, 18.043]], dtype=object)
```

Split the data into training and testing data with a 80/20 split

In [216]:

```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=4
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (160, 5) (160,)
Test set: (40, 5) (40,)
```

We can use GridSearch for Exhaustive search over specified parameter values.

In [217]:

```
param_grid = {'n_estimators': [2*n+1 for n in range(20)],
             'max_depth' : [2*n+1 for n in range(10) ],
             'max_features':["auto", "sqrt", "log2"]}
```

Create a `RandomForestClassifier` object called model :

In [218]:

```
model = RandomForestClassifier()
```

Create `GridSearchCV` object called `search` with the `estimator` set to `model` , `param_grid` set to `param_grid` , `scoring` set to `accuracy` , and `cv` set to 3 and Fit the `GridSearchCV` object to our `X_train` and `y_train` data

In [141]:

```
search = GridSearchCV(estimator=model, param_grid=param_grid,scoring='accuracy', cv=3)
search.fit(X_train, y_train)
```

Out[141]:

```
▸          GridSearchCV
▸ estimator: RandomForestClassifier
      ▸ RandomForestClassifier
```

Click here for the solution

```
search = GridSearchCV(estimator=model, param_grid=param_grid,scoring='accurac
y', cv=3)
search.fit(X_train, y_train)
```

We can find the accuracy of the best model.

In [142]:

```
search.best_score_
```

Out[142]:

```
1.0
```

We can find the best parameter values:

In [143]:

```
search.best_params_
```

Out[143]:

```
{'max_depth': 7, 'max_features': 'auto', 'n_estimators': 17}
```

In [144]:

```
print(get_accuracy(X_train, X_test, y_train, y_test, search.best_estimator_))
```

```
{'test Accuracy': 0.925, 'trian Accuracy': 1.0}
```

We can find the accuracy test data:

Click here for the solution

```
    print(get_accuracy(X_train, X_test, y_train, y_test, search.best_estimator_))
```

Want to learn more?

IBM SPSS Modeler is a comprehensive analytics platform that has many machine learning algorithms. It has been designed to bring predictive intelligence to decisions made by individuals, by groups, by systems – by your enterprise as a whole. A free trial is available through this course, available here: SPSS Modeler (https://www.ibm.com/analytics/spss-statistics-software?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_ic SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

Also, you can use Watson Studio to run these notebooks faster with bigger datasets. Watson Studio is IBM's leading cloud solution for data scientists, built by data scientists. With Jupyter notebooks, RStudio, Apache Spark and popular libraries pre-packaged in the cloud, Watson Studio enables data scientists to collaborate on their projects without having to install anything. Join the fast-growing community of Watson Studio users today with a free account at Watson Studio (https://www.ibm.com/cloud/watson-studio?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_ic SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

Thank you for completing this lab!

Author

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_ic
SkillsNetwork-Channel-
SkillsNetworkCoursesIBMDeveloperSkillsNetworkML0101ENSkillsNetwork20718538-2021-01-01)

Other Contributors

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-02-09	0.1	Joseph Santarcangelo	Created Lab Template
2022-05-03	0.2	Richard Ye	QA pass