Bagging

May 18, 2022

1 Bootstrap Aggregating (Bagging) for classification with Python

Estimated time needed: **45** minutes

1.1 Objectives

After completing this lab you will be able to:

- Understand Bootstrap sampling
- Understand Model Instability
- Apply Bagging
- Understand when to use Bagging

In this notebook, you will learn the process of Bagging (Bootstrap Aggregation) models for classification. Bagging is a method for generating multiple model versions and aggregating the ensemble of models to make a single prediction. For classification, aggregation performs majority vote when predicting a class. The various versions of the model are formed by performing Bootstrap sampling of the training set and using these to train each model in the ensemble .

Table of contents

```
<ol>
    <li><a href="https://#about_dataset">About the dataset</a></li>
    <li><a href="https://#preprocessing">Data pre-processing and selection</a></li>
    <li><a href="https://#modeling">Modeling (Logistic Regression with Scikit-learn)</a></li>
    <li><a href="https://#evaluation">Evaluation</a></li>
    <li><a href="https://#practice">Practice</a></li>
</ol>
```

Let's first import required libraries:

[ ]:
```
# All Libraries required for this lab are listed below. The libraries␣
↪pre-installed on Skills Network Labs are commented.
# !mamba install -qy pandas==1.3.3 numpy==1.21.2 ipywidgets==7.4.2 scipy==7.4.2␣
↪tqdm==4.62.3 matplotlib==3.5.0 seaborn==0.9.0
# Note: If your environment doesn't support "!mamba install", use "!pip␣
↪install"
```

[1]:
```
# Library for reading in data and using dataframes
import pandas as pd
# Using numpy arrays
```

```
import numpy as np
# Data preprocessing functions like LabelEncoder
from sklearn import preprocessing
%matplotlib inline
# Visualizations
import matplotlib.pyplot as plt
# Model accuracy
from sklearn import metrics
# Surpress numpy data type warnings
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
# Give loops a progress bar
from tqdm import tqdm
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-
packages/sklearn/utils/validation.py:37: DeprecationWarning: distutils Version
classes are deprecated. Use packaging.version instead.
  LARGE_SPARSE_SUPPORTED = LooseVersion(scipy_version) >= '0.14.0'
```

This function calculates the train and test accuracy of a model

```
[2]: def get_accuracy(X_train, X_test, y_train, y_test, model):
         return  {"test Accuracy":metrics.accuracy_score(y_test, model.
       ↪predict(X_test)),"train Accuracy": metrics.accuracy_score(y_train, model.
       ↪predict(X_train))}
```

This function creates visualizations of decision trees

```
[3]: # Plot tree helper libraries
     from  io import StringIO
     import pydotplus
     import matplotlib.image as mpimg
     from sklearn import tree


     def plot_tree(model,filename = "tree.png"):
         #global churn_df

         dot_data = StringIO()


         featureNames = [colunm  for colunm in churn_df[['tenure', 'age', 'address',␣
       ↪'income', 'ed', 'employ', 'equip']].columns]
         out=tree.export_graphviz(model,feature_names=featureNames,␣
       ↪out_file=dot_data, class_names= ['left','stay'], filled=True, ␣
       ↪special_characters=True,rotate=False)
         graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
         graph.write_png(filename)
```

```
    img = mpimg.imread(filename)
    plt.figure(figsize=(100, 200))
    plt.imshow(img,interpolation='nearest')
```

This function creates a graph of training accuracy vs how many estimators (Decision Trees) a BaggingClassifier uses

```
[4]: def get_accuracy_bag(X,y,title,times=20,xlabel='Number Estimators'):
         #Iterate through different number of estimators and average out the results␣
      ↪



         N_estimators=[n for n in range(1,70)]
         times=20
         train_acc=np.zeros((times,len(N_estimators)))
         test_acc=np.zeros((times,len(N_estimators)))

         train_time=np.zeros((times,len(N_estimators)))
         test_time=np.zeros((times,len(N_estimators)))
          #average out the results
         for n in tqdm(range(times)):
             X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.
      ↪3)
             for n_estimators in N_estimators:
                 #Iterate through different number of estimators and average out the␣
      ↪results

                 Bag=␣
      ↪BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion="entropy",␣
      ↪max_depth = 10),n_estimators=n_estimators,bootstrap=True,random_state=0)
                 Bag.fit(X_train,y_train)



                 Accuracy=get_accuracy(X_train, X_test, y_train, y_test,  Bag)





                 train_acc[n,n_estimators-1]=Accuracy['train Accuracy']
                 test_acc[n,n_estimators-1]=Accuracy['test Accuracy']




         fig, ax1 = plt.subplots()

         ax2 = ax1.twinx()
```

```
    ax1.plot(train_acc.mean(axis=0))
    ax2.plot(test_acc.mean(axis=0),c='r')
    ax1.set_xlabel(xlabel)
    ax1.set_ylabel('Training accuracy',color='b')
    ax2.set_ylabel('Testing accuracy', color='r')
    plt.title(title)
    plt.show()
```

1.2 Customer churn

A telecommunications company is concerned about the number of customers leaving their land-line business for cable competitors. They need to understand who is leaving. Imagine that you are an analyst at this company and you have to find out why

1.2.1 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

1.2.2 Load Data From CSV File

[5]:
```
churn_df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.
 ↪appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/
 ↪Module%203/data/ChurnData.csv")

churn_df.head()
```

[5]:
```
   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
1    33.0  33.0     12.0    33.0  2.0     0.0    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
4     7.0  35.0     14.0    80.0  2.0    15.0    0.0        1.0        0.0

   longmon  …  pager  internet  callwait  confer  ebill  loglong  logtoll  \
0     4.40  …    1.0       0.0       1.0     1.0    0.0    1.482    3.033
1     9.45  …    0.0       0.0       0.0     0.0    0.0    2.246    3.240
2     6.30  …    0.0       0.0       0.0     1.0    0.0    1.841    3.240
3     6.05  …    1.0       1.0       1.0     1.0    1.0    1.800    3.807
4     7.10  …    0.0       0.0       1.0     1.0    0.0    1.960    3.091

   lninc  custcat  churn
0  4.913      4.0    1.0
1  3.497      1.0    1.0
2  3.401      3.0    0.0
3  4.331      4.0    0.0
4  4.382      3.0    0.0

[5 rows x 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:

[6]:
```python
churn_df = churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ',␣
↪'equip',   'callcard', 'wireless','churn']]
churn_df['churn'] = churn_df['churn'].astype('int')
churn_df.head()
```

[6]:
```
   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
1    33.0  33.0     12.0    33.0  2.0     0.0    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
4     7.0  35.0     14.0    80.0  2.0    15.0    0.0       1.0       0.0

   churn
0      1
1      1
2      0
3      0
4      0
```

1.3 Practice

How many rows and columns are in this dataset in total? What are the names of columns?

[7]:
```
# write your code here
churn_df.shape
```

[7]:
```
(200, 10)
```

Click here for the solution

```
print(churn_df.shape)

print(churn_df.columns)
```

1.4 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

[8]:
```
from sklearn.utils import resample
```

Consider the five rows of data:

[9]:
```
churn_df[0:5]
```

[9]:
```
   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
1    33.0  33.0     12.0    33.0  2.0     0.0    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
4     7.0  35.0     14.0    80.0  2.0    15.0    0.0       1.0       0.0

   churn
0      1
1      1
2      0
3      0
4      0
```

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

[10]:
```
resample(churn_df[0:5])
```

[10]:
```
   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
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
4     7.0  35.0     14.0    80.0  2.0    15.0    0.0       1.0       0.0
```

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

We can repeat the process randomly drawing several other rows

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

```
[11]:    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
      3    38.0  35.0      5.0    76.0  2.0    10.0    1.0       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

         churn
      2      0
      3      0
      4      0
      3      0
      1      1
```

1.5 Train/Test dataset

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

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

      X.head()
```

```
[12]:    tenure   age  address  income   ed  employ  equip
      0    11.0  33.0      7.0   136.0  5.0     5.0    0.0
      1    33.0  33.0     12.0    33.0  2.0     0.0    0.0
      2    23.0  30.0      9.0    30.0  1.0     2.0    0.0
      3    38.0  35.0      5.0    76.0  2.0    10.0    1.0
      4     7.0  35.0     14.0    80.0  2.0    15.0    0.0
```

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

```
[13]: 0    1
      1    1
      2    0
      3    0
      4    0
```

```
Name: churn, dtype: int64
```

1.6 Train/Test dataset

We split our dataset into train and test set:

[14]:
```python
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=0)
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,)
```

Decision Tree Classifier with Scikit-learn

A Decision tree Classifier classifies a sample by learning simple decision rules inferred from the data. One problem with Decision Tree Classifiers is overfitting; they do well with the training data, but they do not Generalize well. Trees have low bias and high variance; as such, they are a prime candidate for Bagging. Instability is another term used to describe models that overfit. Instability is characterized by a slight change in the training set that causes a drastic change in the model. Let's show that Decision tree Classifiers are unstable.

Let's load the DecisionTreeClassifier modle in sklearn

[15]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

We create and train a tree with a max depth of 5

[16]:
```python
max_depth=5
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3,␣
 ↪random_state=10)
Tree = DecisionTreeClassifier(criterion="entropy", max_depth =␣
 ↪max_depth,random_state=10)
Tree
Tree.fit(X_train,y_train)
```

[16]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=10,
            splitter='best')
```

Now we can predict using our test set:

[17]:
```python
yhat = Tree.predict(X_test)
yhat
```

[17]:
```
array([1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0])
```

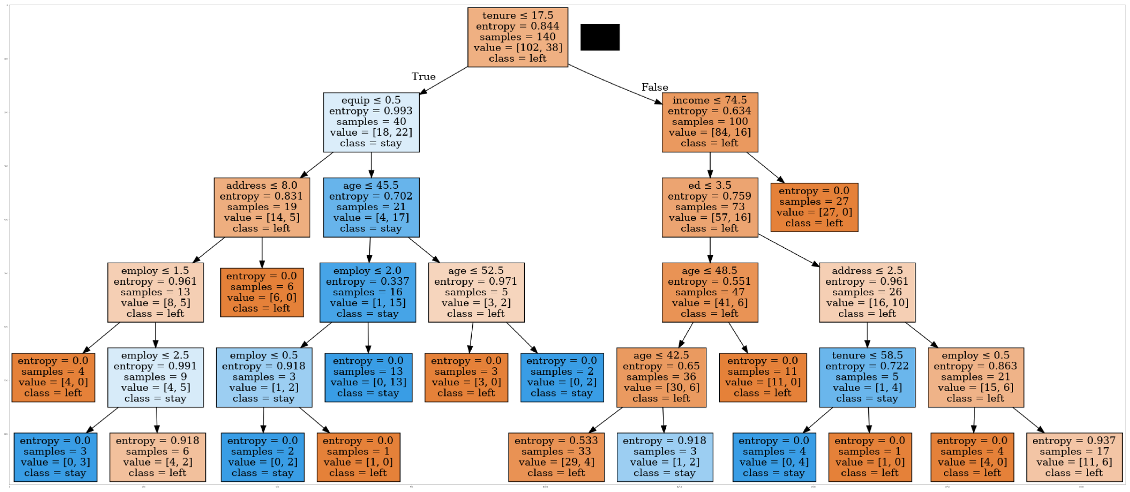
We see the test error is much larger than the training error:

[18]:
```
get_accuracy(X_train, X_test, y_train, y_test,  Tree)
```

[18]:
```
{'test Accuracy': 0.6166666666666667, 'train Accuracy': 0.9071428571428571}
```

We can plot the nodes of the tree:

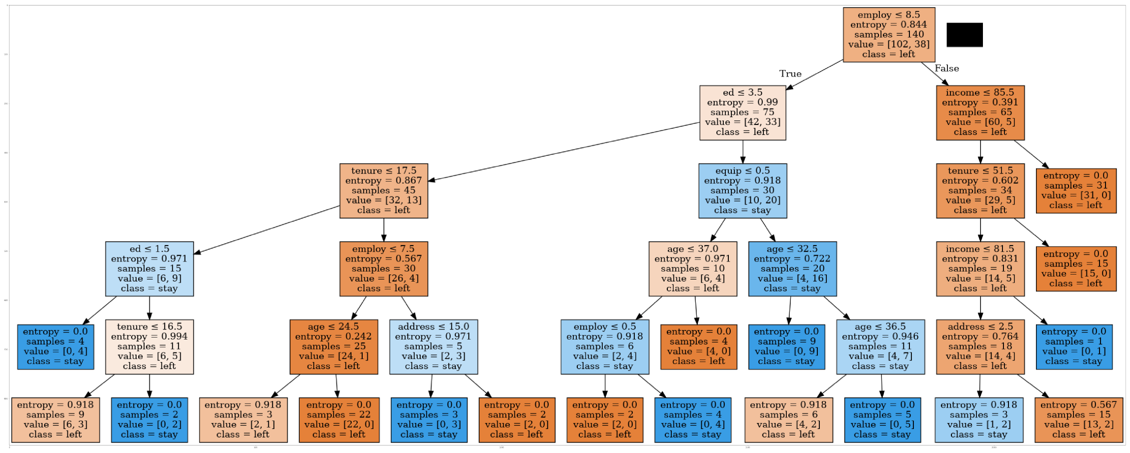
[19]:
```
plot_tree(filename = "tree.png",model=Tree)
```


tenure ≤ 17.5 entropy = 0.844 samples = 140 value = [102, 38] class = left
True
False
equip ≤ 0.5 entropy = 0.993 samples = 40 value = [18, 22] class = stay
income ≤ 74.5 entropy = 0.634 samples = 100 value = [84, 16] class = left
address ≤ 8.0 entropy = 0.831 samples = 19 value = [14, 5] class = left
age ≤ 45.5 entropy = 0.702 samples = 21 value = [4, 17] class = stay
ed ≤ 3.5 entropy = 0.759 samples = 73 value = [57, 16] class = left
entropy = 0.0 samples = 27 value = [27, 0] class = left
employ ≤ 1.5 entropy = 0.961 samples = 13 value = [8, 5] class = left
entropy = 0.0 samples = 6 value = [6, 0] class = left
employ ≤ 2.0 entropy = 0.337 samples = 16 value = [1, 15] class = stay
age ≤ 52.5 entropy = 0.971 samples = 5 value = [3, 2] class = left
age ≤ 48.5 entropy = 0.551 samples = 47 value = [41, 6] class = left
address ≤ 2.5 entropy = 0.961 samples = 26 value = [16, 10] class = left
entropy = 0.0 samples = 4 value = [4, 0] class = left
employ ≤ 2.5 entropy = 0.991 samples = 9 value = [4, 5] class = stay
employ ≤ 0.5 entropy = 0.918 samples = 3 value = [1, 2] class = stay
entropy = 0.0 samples = 13 value = [0, 13] class = stay
entropy = 0.0 samples = 3 value = [3, 0] class = left
entropy = 0.0 samples = 2 value = [0, 2] class = stay
age ≤ 42.5 entropy = 0.65 samples = 36 value = [30, 6] class = left
entropy = 0.0 samples = 11 value = [11, 0] class = left
tenure ≤ 58.5 entropy = 0.722 samples = 5 value = [1, 4] class = stay
employ ≤ 0.5 entropy = 0.863 samples = 21 value = [15, 6] class = left
entropy = 0.0 samples = 3 value = [0, 3] class = stay
entropy = 0.918 samples = 6 value = [4, 2] class = left
entropy = 0.0 samples = 2 value = [0, 2] class = stay
entropy = 0.0 samples = 1 value = [1, 0] class = left
entropy = 0.533 samples = 33 value = [29, 4] class = left
entropy = 0.918 samples = 3 value = [1, 2] class = stay
entropy = 0.0 samples = 4 value = [0, 4] class = stay
entropy = 0.0 samples = 1 value = [1, 0] class = left
entropy = 0.0 samples = 4 value = [4, 0] class = left
entropy = 0.937 samples = 17 value = [11, 6] class = left

We can repeat the process but sampling different data points from the same dataset. We see the tree still suffers from overfitting; in addition, the new tree is entirely different.

[20]:
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.3,␣
 ↪random_state=5)
Tree = DecisionTreeClassifier(criterion="entropy", max_depth =␣
 ↪max_depth,random_state=8)
Tree.fit(X_train,y_train)
print(get_accuracy(X_train, X_test, y_train, y_test,  Tree))
plot_tree(filename = "tree1.png",model=Tree)
```

```
{'test Accuracy': 0.65, 'train Accuracy': 0.9357142857142857}
```


employ ≤ 8.5 entropy = 0.844 samples = 140 value = [102, 38] class = left
True
False
ed ≤ 3.5 entropy = 0.99 samples = 75 value = [42, 33] class = left
income ≤ 85.5 entropy = 0.391 samples = 65 value = [60, 5] class = left
tenure ≤ 17.5 entropy = 0.867 samples = 45 value = [32, 13] class = left
equip ≤ 0.5 entropy = 0.918 samples = 30 value = [10, 20] class = stay
tenure ≤ 51.5 entropy = 0.602 samples = 34 value = [29, 5] class = left
entropy = 0.0 samples = 31 value = [31, 0] class = left
ed ≤ 1.5 entropy = 0.971 samples = 15 value = [6, 9] class = stay
employ ≤ 7.5 entropy = 0.567 samples = 30 value = [26, 4] class = left
age ≤ 37.0 entropy = 0.971 samples = 10 value = [6, 4] class = left
age ≤ 32.5 entropy = 0.722 samples = 20 value = [4, 16] class = stay
income ≤ 81.5 entropy = 0.831 samples = 19 value = [14, 5] class = left
entropy = 0.0 samples = 15 value = [15, 0] class = left
entropy = 0.0 samples = 4 value = [0, 4] class = stay
tenure ≤ 16.5 entropy = 0.994 samples = 11 value = [6, 5] class = left
age ≤ 24.5 entropy = 0.242 samples = 25 value = [24, 1] class = left
address ≤ 15.0 entropy = 0.971 samples = 5 value = [2, 3] class = stay
employ ≤ 0.5 entropy = 0.918 samples = 6 value = [2, 4] class = stay
entropy = 0.0 samples = 4 value = [4, 0] class = left
entropy = 0.0 samples = 9 value = [0, 9] class = stay
age ≤ 36.5 entropy = 0.946 samples = 11 value = [4, 7] class = stay
address ≤ 2.5 entropy = 0.764 samples = 18 value = [14, 4] class = left
entropy = 0.0 samples = 1 value = [0, 1] class = stay
entropy = 0.918 samples = 9 value = [6, 3] class = left
entropy = 0.0 samples = 2 value = [0, 2] class = stay
entropy = 0.918 samples = 3 value = [2, 1] class = left
entropy = 0.0 samples = 22 value = [22, 0] class = left
entropy = 0.0 samples = 3 value = [0, 3] class = stay
entropy = 0.0 samples = 2 value = [2, 0] class = left
entropy = 0.0 samples = 2 value = [2, 0] class = left
entropy = 0.0 samples = 4 value = [0, 4] class = stay
entropy = 0.918 samples = 6 value = [4, 2] class = left
entropy = 0.0 samples = 5 value = [0, 5] class = stay
entropy = 0.918 samples = 3 value = [1, 2] class = stay
entropy = 0.567 samples = 15 value = [13, 2] class = left

Bagging for classification with Scikit-learn

A Bagging classifier is an ensemble model that trains base classifiers on random subsets of the original dataset (Bootstrap Sampling by default), and then aggregate their individual predictions by voting. We import the module:

[21]:
```
from sklearn.ensemble import BaggingClassifier
```

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

[22]:
```
Bag =␣
 ↪BaggingClassifier(base_estimator=DecisionTreeClassifier(criterion="entropy",
                                                           max_depth =␣
 ↪4,random_state=2),
                                                          ␣
 ↪n_estimators=30,random_state=0,bootstrap=True)
```

We fit the model:

[23]:
```
Bag.fit(X_train,y_train)
```

[23]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None,
criterion='entropy', max_depth=4,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=2,
            splitter='best'),
         bootstrap=True, bootstrap_features=False, max_features=1.0,
         max_samples=1.0, n_estimators=30, n_jobs=None, oob_score=False,
         random_state=0, verbose=0, warm_start=False)
```

The method predict aggregates the predictions by voting:

```
[24]: Bag.predict(X_test)
```

```
[24]: array([0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0,
             1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
             1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1])
```

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

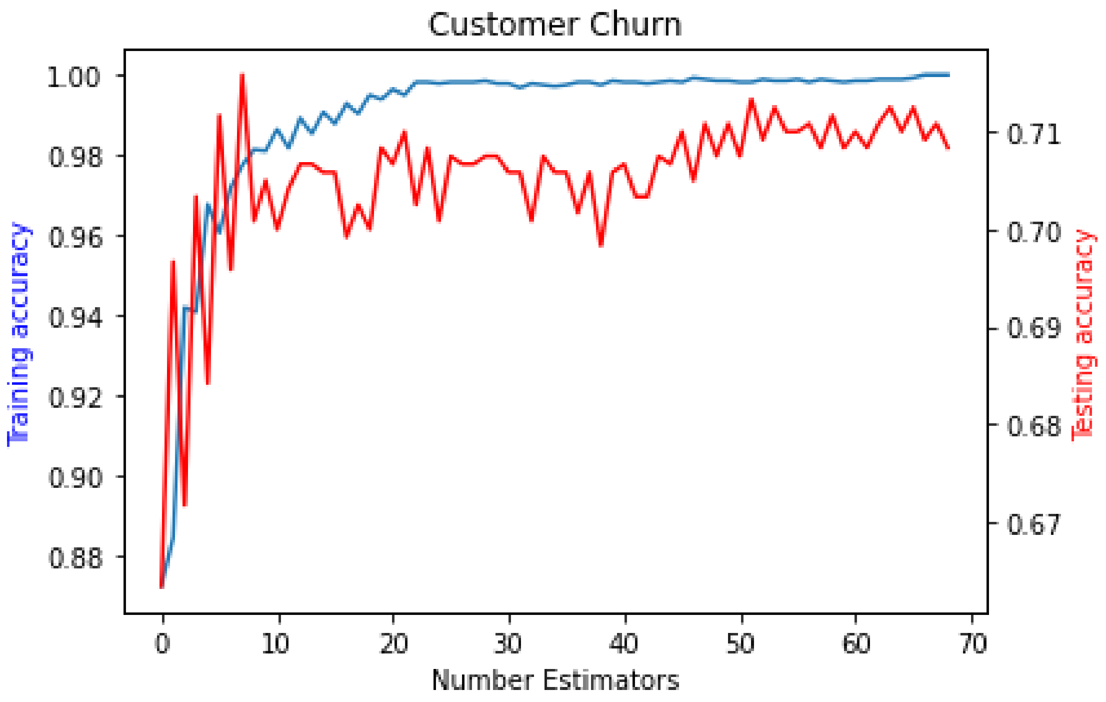
```
[25]: print(get_accuracy(X_train, X_test, y_train, y_test,  Bag))
```

```
{'test Accuracy': 0.6833333333333333, 'train Accuracy': 0.9071428571428571}
```

Here we can see the impact of adding more estimators (Decision Trees) using in Bagging on the testing and training accuracy

```
[26]: get_accuracy_bag(X, y, "Customer Churn")
```

```
100%|      | 20/20 [01:05<00:00,  3.28s/it]
```



1.7 Low Variance Example

Bagging does not improve result if the model has low Variance i.e. does reasonably well on the test and training data. Consider the SVM; the accuracy on the tests data and training data are similar

```
[27]: from sklearn.svm import SVC

      clf=SVC(kernel='linear',gamma='scale')
      clf.fit(X_train, y_train)
      print(get_accuracy(X_train, X_test, y_train, y_test,  clf))
```

```
{'test Accuracy': 0.7166666666666667, 'train Accuracy': 0.7642857142857142}
```

Bagging the SVM does almost nothing:

```
[28]: Bag =␣
        ↪BaggingClassifier(base_estimator=SVC(kernel='linear',gamma='scale'),n_estimators=10,random_
      Bag.fit(X_train,y_train)
      print(get_accuracy(X_train, X_test, y_train, y_test,  Bag))
```

```
{'test Accuracy': 0.6833333333333333, 'train Accuracy': 0.7714285714285715}
```

Practice: Cancer data

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]. 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:

```
[29]: df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
        ↪cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/
        ↪cell_samples.csv")

      df.head()
```

```
[29]:         ID  Clump  UnifSize  UnifShape  MargAdh  SingEpiSize BareNuc  \
      0  1000025      5         1          1        1            2       1
      1  1002945      5         4          4        5            7      10
```

```
2  1015425      3         1          1        1            2        2
3  1016277      6         8          8        1            3        4
4  1017023      4         1          1        3            2        1

   BlandChrom  NormNucl  Mit  Class
0           3         1    1      2
1           3         2    1      2
2           3         1    1      2
3           3         7    1      2
4           3         1    1      2
```

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

```
[30]: df = df[df["BareNuc"] != "?"]
```

Now lets define the X and y for our dataset

```
[31]: X =  df[['Clump', 'UnifSize', 'UnifShape', 'MargAdh', 'SingEpiSize', 'BareNuc',␣
       ↪'BlandChrom', 'NormNucl', 'Mit']]

      X.head()
```

```
[31]:    Clump  UnifSize  UnifShape  MargAdh  SingEpiSize BareNuc  BlandChrom  \
      0      5         1          1        1            2       1           3
      1      5         4          4        5            7      10           3
      2      3         1          1        1            2       2           3
      3      6         8          8        1            3       4           3
      4      4         1          1        3            2       1           3

         NormNucl  Mit
      0         1    1
      1         2    1
      2         1    1
      3         7    1
      4         1    1
```

```
[32]: y = df['Class']

      y.head()
```

```
[32]: 0    2
      1    2
      2    2
      3    2
      4    2
      Name: Class, dtype: int64
```

Now lets split our data into training and testing data with a 80/20 split

[33]:
```
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,)
```

Now to determine the best parameters for `n_estimators` and the `max_depth` of the `base_estimator` we will use `GridSearchCV`

[34]:
```
from sklearn.model_selection import GridSearchCV
```

We can use GridSearch for Exhaustive search over specified parameter values. To alter the base model; we add the double underscore and the attribute value:

Here we are searching odd numbers from 1 to 39 for `n_estimators` and odd numbers from 1 to 20 for `max_depth` in the `base_estimator`

[35]:
```
param_grid = {'n_estimators': [2*n+1 for n in range(20)],
     'base_estimator__max_depth' : [2*n+1 for n in range(10) ] }
```

Create a `BaggingClassifier` object called `Bag` with the `base_estimator` set to a `DecisionTreeClassifier` object where `random_state` = 0 and `bootstrap` = True

[38]:
```
Bag = BaggingClassifier(base_estimator= DecisionTreeClassifier(criterion =␣
 ↪'entropy', random_state=0), bootstrap = True)
```

Click here for the solution

```
Bag = BaggingClassifier(base_estimator = DecisionTreeClassifier(), random_state=0, bootstrap=T
```

Now we create a `GridSearchCV` object and search for the best parameters according to our `parameter_grid`

[39]:
```
search = GridSearchCV(estimator=Bag, param_grid=param_grid, scoring='accuracy',␣
 ↪cv=3)
```

[40]:
```
search.fit(X_train, y_train)
```

[40]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier(class_w
eight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
         …stimators=10, n_jobs=None, oob_score=False,
         random_state=None, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23,
25, 27, 29, 31, 33, 35, 37, 39], 'base_estimator__max_depth': [1, 3, 5, 7, 9,
11, 13, 15, 17, 19]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

We can see the best accuracy score of the searched parameters was ~97%

[42]:
```
search.best_score_
```

[42]:
```
0.9725274725274725
```

We can also see the parameters that resulted in the best score

[43]:
```
search.best_params_
```

[43]:
```
{'base_estimator__max_depth': 5, 'n_estimators': 37}
```

And we can see the testing and training accuracy of the best estimator

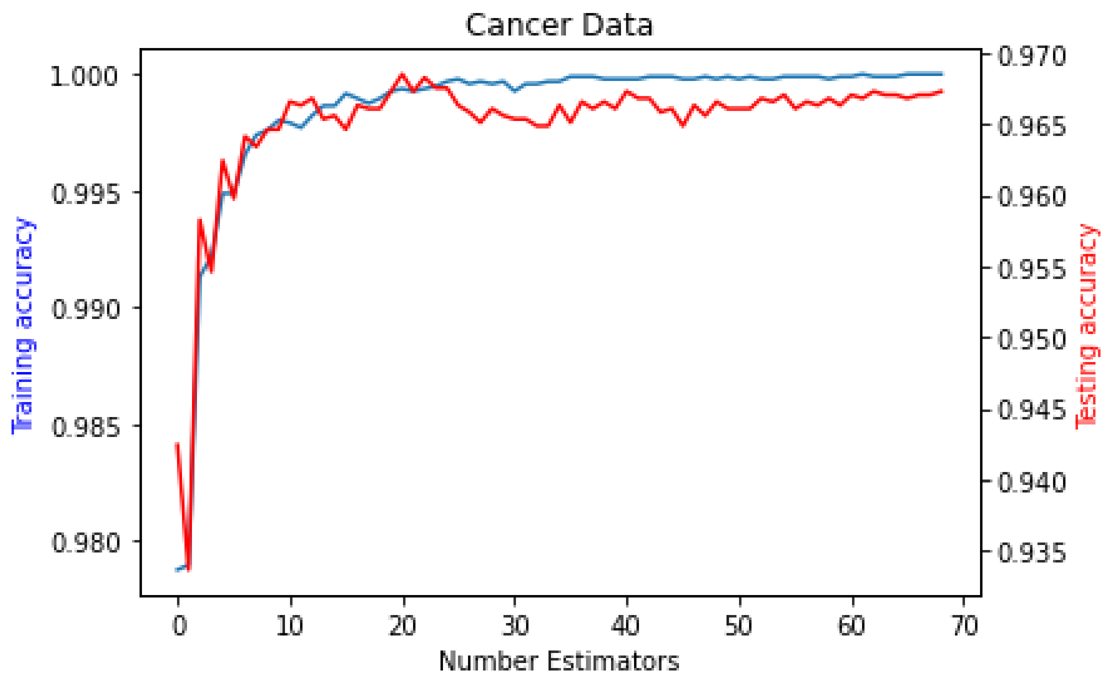
[44]:
```
print(get_accuracy(X_train, X_test, y_train, y_test, search.best_estimator_))
```

```
{'test Accuracy': 0.9708029197080292, 'train Accuracy': 0.9908424908424909}
```

Below we can see a graph of testing and training accuracy holding the `max_depth` of the `base_estimator` at 10 and varying the number of estimators. We can see that it is extremely close to the accuracy of the `best_estimator` we found using `GridSearchCV`

[45]:
```
get_accuracy_bag(X, y, "Cancer Data")
```

```
100%|      | 20/20 [01:28<00:00,  4.45s/it]
```



Practice: During their course of treatment

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.

[46]:
```
df = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.
↪cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%203/data/
↪drug200.csv", delimiter=",")
df.head()
```

[46]:
```
   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
```

Lets create the X and y for our dataset

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

```
[47]: 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)
```

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

```
[48]: 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

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

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

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

```
[52]: X[0:5]
```

```
[52]: 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

```
[53]: 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,)
```

Using the same parameter grid as before

[54]:
```
param_grid = {'n_estimators': [2*n+1 for n in range(20)],
     'base_estimator__max_depth' : [2*n+1 for n in range(10) ]}
```

Create a `BaggingClassifier` object called `Bag` with the `base_estimator` set to a `DecisionTreeClassifier` object where `random_state` = 0 and `bootstrap` = True

[55]:
```
# add your code below
Bag = BaggingClassifier(base_estimator = DecisionTreeClassifier(random_state =␣
 ↪0), bootstrap=True)
```

Click here for the solution

```
Bag = BaggingClassifier(base_estimator=DecisionTreeClassifier(),random_state=0,bootstrap=True)
```

Create `GridSearchCV` object called `search` with the `estimator` set to `Bag`, `param_grid` set to `param_grid`, `scoring` set to `accuracy`, and `cv` set to 3.

[56]:
```
# add your code below
search = GridSearchCV(estimator = Bag, param_grid = param_grid, scoring =␣
 ↪'accuracy', cv = 3)
```

Click here for the solution

```
search = GridSearchCV(estimator=Bag, param_grid=param_grid,scoring='accuracy', cv=3)
```

Fit the `GridSearchCV` object to our `X_train` and `y_train` data

[57]:
```
# add your code below
search.fit(X_train, y_train)
```

[57]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier(class_w
eight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            …stimators=10, n_jobs=None, oob_score=False,
         random_state=None, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23,
25, 27, 29, 31, 33, 35, 37, 39], 'base_estimator__max_depth': [1, 3, 5, 7, 9,
11, 13, 15, 17, 19]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

Click here for the solution

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

After using `fit` we can see the best score and parameters

```
[58]: search.best_score_
```

```
[58]: 1.0
```

```
[59]: search.best_params_
```

```
[59]: {'base_estimator__max_depth': 5, 'n_estimators': 1}
```

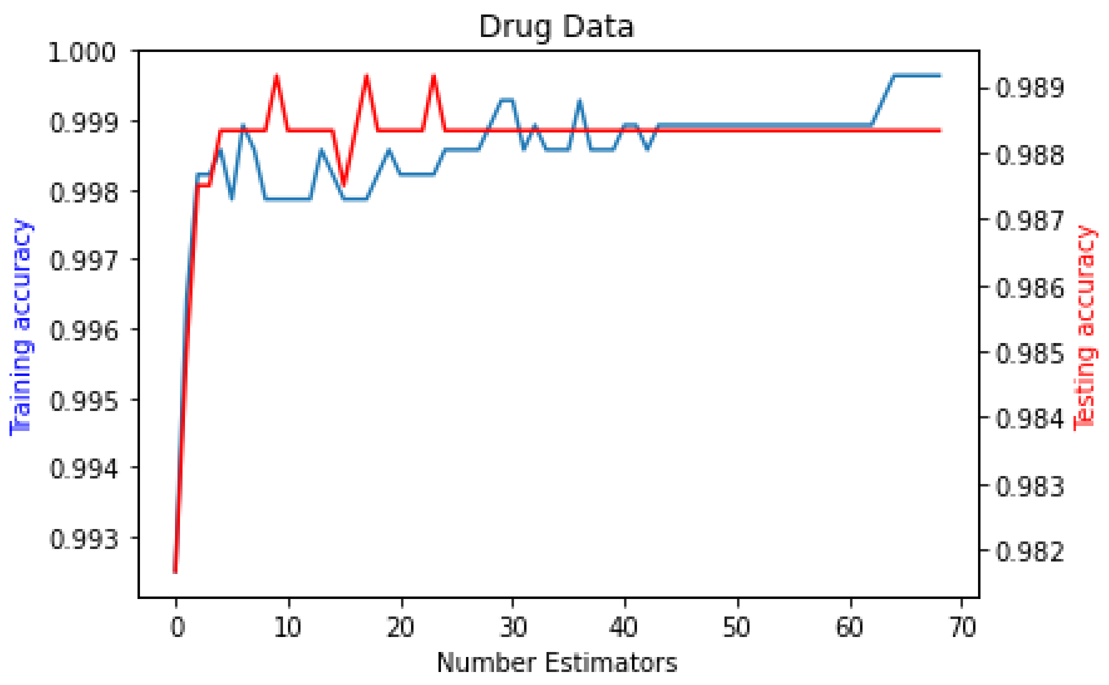
```
[60]: print(get_accuracy(X_train, X_test, y_train, y_test, search.best_estimator_))
```

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

Below we can see a graph of testing and training accuracy holding the max_depth of the base_estimator at 10 and varying the number of estimators. We can see that it is extremely close to the accuracy of the best_estimator we found using GridSearchCV

```
[61]: get_accuracy_bag(X, y, "Drug Data")
```

```
100%|      | 20/20 [01:01<00:00,  3.06s/it]
```



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

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

1.7.1 Thank you for completing this lab!

1.8 Author

Joseph Santarcangelo

1.9 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description

##