Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Exe22 - Bagging and Boosting Exercise

Name: Phua Yan Han

IC Number: 050824070059

Date :

Introduction :

Conclusion :

Bagging and Boosting Exercise

Reference: (https://www.datacamp.com/community/tutorials/ensemble-learning-python (https://www.datacamp.com/community/tutorials/ensemble-learning-python))

Bagging Method

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
from sklearn.preprocessing import MinMaxScaler
```

In [2]:
```
import pandas as pd
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/b
                   header=None)
data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size', 'U
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nucle
                'Normal Nucleoli', 'Mitoses','Class']

data = data.drop(['Sample code'],axis=1)
print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))
data.head()
```

```
Number of instances = 699
Number of attributes = 10
```

Out[2]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses
0	5	1	1	1	2	1	3	1	1
1	5	4	4	5	7	10	3	2	1
2	3	1	1	1	2	2	3	1	1
3	6	8	8	1	3	4	3	7	1
4	4	1	1	3	2	1	3	1	1

In [3]:
```
data.describe()
```

Out[3]:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bland Chromatin	Normal Nucleoli	M
count	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.000000	699.(
mean	4.417740	3.134478	3.207439	2.806867	3.216023	3.437768	2.866953	1.5
std	2.815741	3.051459	2.971913	2.855379	2.214300	2.438364	3.053634	1.7
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.(
25%	2.000000	1.000000	1.000000	1.000000	2.000000	2.000000	1.000000	1.(
50%	4.000000	1.000000	1.000000	1.000000	2.000000	3.000000	1.000000	1.(
75%	6.000000	5.000000	5.000000	4.000000	4.000000	5.000000	4.000000	1.(
max	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.(

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
 #   Column                       Non-Null Count  Dtype 
---  ------                       --------------  ----- 
 0   Clump Thickness              699 non-null    int64 
 1   Uniformity of Cell Size      699 non-null    int64 
 2   Uniformity of Cell Shape     699 non-null    int64 
 3   Marginal Adhesion            699 non-null    int64 
 4   Single Epithelial Cell Size  699 non-null    int64 
 5   Bare Nuclei                  699 non-null    object
 6   Bland Chromatin              699 non-null    int64 
 7   Normal Nucleoli              699 non-null    int64 
 8   Mitoses                      699 non-null    int64 
 9   Class                        699 non-null    int64 
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
```

In [5]:
```
data['Bare Nuclei']
```

Out[5]:
```
0       1
1      10
2       2
3       4
4       1
       ..
694     2
695     1
696     3
697     4
698     5
Name: Bare Nuclei, Length: 699, dtype: object
```

In [6]:
```
data.replace('?',0, inplace=True)
data['Bare Nuclei']
```

Out[6]:
```
0       1
1      10
2       2
3       4
4       1
       ..
694     2
695     1
696     3
697     4
698     5
Name: Bare Nuclei, Length: 699, dtype: object
```

In [7]:
```
# Convert the DataFrame object into NumPy array otherwise you will not be able
values = data.values

# Now impute it

imputedData = imputer.fit_transform(values)
```

In [8]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
normalizedData = scaler.fit_transform(imputedData)
```

In [9]:
```
# Bagged Decision Trees for Classification - necessary dependencies
from sklearn.datasets import make_classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
```

In [10]:
```
# Segregate the features from the labels
X = data.drop('Class',axis=1)
y = data['Class']
```

In [11]:
```
model = BaggingClassifier()
# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, e
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.955 (0.024)
```

In [ ]:
```

```

Boosting Method

In [12]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn import model_selection
seed = 7
num_trees = 70
kfold = model_selection.KFold(n_splits=10, random_state=seed,shuffle=True)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = model_selection.cross_val_score(model, X, y, cv=kfold)
print(results.mean())
```

```
0.9599378881987578
```

Exercise 1 Perform classification using the Titanic dataset using the classifiers that you already know (Dtree and RF)

In [13]:
```
#Preprocessing the entire Titanic dataset
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

t_dataset = pd.read_csv('../Data files/titanic.csv')
t_dataset
```

Out[13]:

	Survived	Pclass	Name	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	Mr. Owen Harris Braund	male	22.0	1	0	7.2500
1	1	1	Mrs. John Bradley (Florence Briggs Thayer) Cum...	female	38.0	1	0	71.2833
2	1	3	Miss. Laina Heikkinen	female	26.0	0	0	7.9250
3	1	1	Mrs. Jacques Heath (Lily May Peel) Futrelle	female	35.0	1	0	53.1000
4	0	3	Mr. William Henry Allen	male	35.0	0	0	8.0500
...	...	...	...	...	...	...	...	...
882	0	2	Rev. Juozas Montvila	male	27.0	0	0	13.0000
883	1	1	Miss. Margaret Edith Graham	female	19.0	0	0	30.0000
884	0	3	Miss. Catherine Helen Johnston	female	7.0	1	2	23.4500
885	1	1	Mr. Karl Howell Behr	male	26.0	0	0	30.0000
886	0	3	Mr. Patrick Dooley	male	32.0	0	0	7.7500

887 rows × 8 columns

In [14]:
```
#drop name column
t_dataset=t_dataset.drop('Name',axis=1)
t_dataset
```

Out[14]:

	Survived	Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	male	22.0	1	0	7.2500
1	1	1	female	38.0	1	0	71.2833
2	1	3	female	26.0	0	0	7.9250
3	1	1	female	35.0	1	0	53.1000
4	0	3	male	35.0	0	0	8.0500
...	...	...	...	...	...	...	...
882	0	2	male	27.0	0	0	13.0000
883	1	1	female	19.0	0	0	30.0000
884	0	3	female	7.0	1	2	23.4500
885	1	1	male	26.0	0	0	30.0000
886	0	3	male	32.0	0	0	7.7500

887 rows × 7 columns

In [15]:
```
#encode categorical data into numerical value
from sklearn import preprocessing
t_dataset['Sex'] = t_dataset['Sex'].replace({'male': 1, 'female': 0})
t_dataset
```

Out[15]:

	Survived	Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	1	22.0	1	0	7.2500
1	1	1	0	38.0	1	0	71.2833
2	1	3	0	26.0	0	0	7.9250
3	1	1	0	35.0	1	0	53.1000
4	0	3	1	35.0	0	0	8.0500
...	...	...	...	...	...	...	...
882	0	2	1	27.0	0	0	13.0000
883	1	1	0	19.0	0	0	30.0000
884	0	3	0	7.0	1	2	23.4500
885	1	1	1	26.0	0	0	30.0000
886	0	3	1	32.0	0	0	7.7500

887 rows × 7 columns

In [16]:
```
#create a copy of the cleaned dataset
t_dataset_copy = t_dataset.copy()
t_dataset_copy
```

Out[16]:

	Survived	Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare
0	0	3	1	22.0	1	0	7.2500
1	1	1	0	38.0	1	0	71.2833
2	1	3	0	26.0	0	0	7.9250
3	1	1	0	35.0	1	0	53.1000
4	0	3	1	35.0	0	0	8.0500
...	...	...	...	...	...	...	...
882	0	2	1	27.0	0	0	13.0000
883	1	1	0	19.0	0	0	30.0000
884	0	3	0	7.0	1	2	23.4500
885	1	1	1	26.0	0	0	30.0000
886	0	3	1	32.0	0	0	7.7500

887 rows × 7 columns

In [17]:
```
#define dependent variable and independent variable
X = t_dataset_copy[['Pclass','Sex','Siblings/Spouses Aboard','Parents/Children
y = t_dataset_copy['Survived'].values
print(X)
print(y)
```

```
[[ 3.      1.      1.      0.      7.25  ]
 [ 1.      0.      1.      0.     71.2833]
 [ 3.      0.      0.      0.      7.925 ]
 ...
 [ 3.      0.      1.      2.     23.45  ]
 [ 1.      1.      0.      0.     30.    ]
 [ 3.      1.      0.      0.      7.75  ]]
[0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 0 1 0 0 1 1 0 0 0 1
 0 0 1 0 0 1 1 0 0 1 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1
 0 0 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0
 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0
 1 1 0 0 1 0 1 1 1 1 0 0 1 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 0
 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 1 1
 1 0 0 0 0 0 0 1 1 1 0 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 0 1 1 1 1 0 0 0 0 0 0
 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 1 0 1 0 1 1 1 1 0 0 0 1
 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0
 0 0 0 1 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 1 0
 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 1 1
 1 1 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1
 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1 0 1 0 1
 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 0 1 1 0 1 0
 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 0 1 1 1 0 0 0 1 0 1 0 0 0 1 0 0 0
 0 1 0 0 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0 1 0 0 1 0
 1 0 0 1 0 0 0 0 0 1 0 1 1 1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0
 1 1 0 0 1 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0
 1 1 1 1 1 0 1 0 0 0 1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0
 1 0 0 1 0 0 1 1 0 0 1 1 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1
 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0
 1 1 1 1 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 1 0 1 0]
```

In [18]:
```
#Split the dataset into the Training and the Test set. Set the test set to 0.3
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
```

In [ ]:
```
```

In [19]:
```
#Decision Tree object

from sklearn import metrics,tree
clf = DecisionTreeClassifier(max_depth =3)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct
metrics.accuracy_score(y_test,y_pred)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[19], line 4
      1 #Decision Tree object
      3 from sklearn import metrics,tree
----> 4 clf = DecisionTreeClassifier(max_depth =3)
      6 # Train Decision Tree Classifer
      7 clf = clf.fit(X_train,y_train)

NameError: name 'DecisionTreeClassifier' is not defined
```

In [ ]:
```
#Random forest
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, max_depth=3)

# Train the Random Forest classifier
clf = clf.fit(X_train, y_train)

# Predict the response for the test dataset
y_pred = clf.predict(X_test)
print(y_pred)
y_pred_flatten = clf.predict_proba(X_test).flatten()
print(y_pred_flatten)
# Model Accuracy, how often the classifier is correct
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

Exercise 2 Perform classification using the Titanic dataset using the classifiers that you already know and with feature selection and dimension reduction. Which gives you the best result?

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
# Step 2: Feature Scaling with StandardScaler
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
# Step 3: Apply PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
# Step 4: Create and train Decision Tree Classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_pca, y_train)
# Step 5: Predict the response for test dataset
y_pred = clf.predict(X_test_pca)
explained_variance_ratio = pca.explained_variance_ratio_
print(explained_variance_ratio)
print(y_pred)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
# Step 2: Feature Scaling with StandardScaler
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
# Step 3: Apply PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
# Step 4: Create and train Decision Tree Classifier
clf = DecisionTreeClassifier()
clf.fit(X_train_pca, y_train)
# Step 5: Predict the response for test dataset
y_pred = clf.predict(X_test_pca)
explained_variance_ratio = pca.explained_variance_ratio_
print(explained_variance_ratio)
print(y_pred)
accuracy = accuracy_score(y_test, y_pred)
print(accuracy)
```

In [ ]:
```
#rebuild analytical dataset & create a copy of the cleaned dataset
t_dataset_copy2 = t_dataset.copy()

#define dependent variable and independent variable
X = t_dataset_copy2[['Pclass','Sex','Siblings/Spouses Aboard','Parents/Children
y = t_dataset_copy2['Survived'].values

#Split the dataset into the Training and the Test set. Set the test set to 0.3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random
```

In [ ]:
```
#RF Feature Selector
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score

# Create a random forest classifier (10000 trees)
clf = RandomForestClassifier(n_estimators=10000, max_depth=3)

# Train the classifier
clf = clf.fit(X_train, y_train)

# Create a selector object that will use the random forest classifier to identi
# features that have an importance of more than 0.15
selector = SelectFromModel(clf, threshold=0.15)

# Train the selector
X_train_selected = selector.transform(X_train)
X_test_selected = selector.transform(X_test)


# Create a new random forest classifier for the most important features
clf_most_important = RandomForestClassifier(n_estimators=1000, max_depth=3)

# Train the new classifier on the new dataset containing the most important fea
clf_most_important.fit(X_train_selected, y_train)

# Apply The Limited Classifier To The Test Data
y_pred_most_important = clf_most_important.predict(X_test_selected)

# View The Accuracy Of Our Limited Feature (2 Features) Model
accuracy_most_important = accuracy_score(y_test, y_pred_most_important)
print("Accuracy with most important features on test data:", accuracy_most_impo
# 0.7790262172284644
```

Exercise 3 Perform classification using the Titanic dataset using bagging and boosting (choose 1 bagging and 1 boosting algo)

In [ ]:
```
!pip install xgboost
```

In [ ]:
```
#create a copy of the cleaned dataset
import xgboost as xgb
t_dataset_copy3 = t_dataset.copy()
#define dependent variable and independent variable
X = t_dataset_copy3[['Pclass','Sex','Siblings/Spouses Aboard','Parents/Children
y = t_dataset_copy3['Survived'].values
#Split the dataset into the Training and the Test set. Set the test set to 0.3
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random
# Apply Xgboost
clf_xgb = xgb.XGBClassifier()
#fit model
clf_xgb.fit(X_train, y_train)
# make predictions for test data
y_pred_xgb = clf_xgb.predict(X_test)
# evaluate predictions
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print("Accuracy with XGBoost on test data:", accuracy_xgb)
```

Out of all 3 approaches, which gives you the best result?

In [ ]:
```
print('random forest classifier')
```