Ensemble classification methods

Training multiple machine learning algorithms to improve predictive performance compared to the base model.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

Data preprocessing

The *NY & SF real estate* dataset contains 7 feature variables to classify if a property is from New York City or San Fransisco. This data was taken from R2D4.

In [2]:
```
#load data
df = pd.read_csv("https://raw.githubusercontent.com/dswede43/ML-methods/main/data/r
df.head()
```

Out[2]:

	in_sf	beds	bath	price	year_built	sqft	price_per_sqft	elevation
0	0	2.0	1.0	999000	1960	1000	999	10
1	0	2.0	2.0	2750000	2006	1418	1939	0
2	0	2.0	2.0	1350000	1900	2150	628	9
3	0	1.0	1.0	629000	1903	500	1258	9
4	0	0.0	1.0	439000	1930	500	878	10

Feature variables (7)

1. beds (continuous) Number of beds in the property

2. bath (continuous): Number of baths in the property

3. price (continuous): Price value of the property

4. year_built (continuous): Year the property was built

5. sqft (continuous): Square-foot size of the property

6. price_per_sqft (continuous): Price per square-foot of the property

7. elevation (continuous): Elevation of the property

In [3]:
```
#check for null values in the data
df.isna().sum()
```

Out[3]:
```
in_sf            0
beds             0
bath             0
price            0
year_built       0
sqft             0
price_per_sqft   0
elevation        0
dtype: int64
```

In [4]:
```
#define the feature and target variable data
X = df.drop(['in_sf'], axis = 1)
y = df['in_sf']
```

In [5]:
```
#split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_s
```

Single tree classifier

Supervised non-parametric machine learning methods to classify a target variable.

In [6]:
```
#fit the single tree classifier
single_tree = DecisionTreeClassifier(random_state = 42)
single_tree.fit(X_train, y_train)
```

Out[6]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

In [7]:
```
#calculate the accuracy of the model
y_pred = single_tree.predict(X_test)
single_tree_accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of single decision tree classifier on test set is {single_tree_acc
```

```
Accuracy of single decision tree classifier on test set is 0.8889
```

Bagging

Construct multiple tree classifiers using multiple bootstrapped training sets and average the resulting predictions to reduce variance.

In [8]:
```
#define a range of tree numbers (forest size)
num_trees = list(range(1, 201))
```

In [9]:
```
#create the random forest classifier
bagging_classifier = BaggingClassifier(n_jobs = -1)

bagging_results = []
#for each forest size
for i in num_trees:
    #set the number of trees
    bagging_classifier.set_params(n_estimators = i)

    #fit the model
    bagging_classifier.fit(X_train, y_train)

    #calculate the model accuracy
    y_pred = bagging_classifier.predict(X_test)
    bagging_accuracy = accuracy_score(y_test, y_pred)

    #store the results
    bagging_results.append([i, bagging_accuracy])

bagging_results = pd.DataFrame(bagging_results)
bagging_results.columns = ['forest_size', 'accuracy']
bagging_results.head()
```

Out[9]:

	forest_size	accuracy
0	1	0.848485
1	2	0.868687
2	3	0.868687
3	4	0.888889
4	5	0.878788

Out-of-bag (OOB) Bagging

Bagging where predictions are made using the out-of-bag (OOB) sample observations. Similar to leave-one-out cross-validation (LOOCV).

In [10]:
```
#create the random forest classifier
oob_bagging_classifier = BaggingClassifier(oob_score = True, n_jobs = -1)

oob_bagging_results = []
#for each forest size
for i in num_trees[16:]:
    #set the number of trees
    oob_bagging_classifier.set_params(n_estimators = i)

    #fit the model
```

```
    oob_bagging_classifier.fit(X_train, y_train)

    #calculate the model accuracy
    oob_bagging_accuracy = oob_bagging_classifier.oob_score_

    #store the results
    oob_bagging_results.append([i, oob_bagging_accuracy])

oob_bagging_results = pd.DataFrame(oob_bagging_results)
oob_bagging_results.columns = ['forest_size', 'accuracy']
oob_bagging_results.head()
```

Out[10]:

	forest_size	accuracy
0	17	0.900763
1	18	0.913486
2	19	0.893130
3	20	0.895674
4	21	0.900763

Random forests

Bagging but each tree is constructed using a random sample of feature variables to decorrelate and diversify each tree.

In [11]:
```
#create the random forest classifier
rf_classifier = RandomForestClassifier(n_jobs = -1)

rf_results = []
#for each forest size
for i in num_trees:
    #set the number of trees
    rf_classifier.set_params(n_estimators = i)

    #fit the model
    rf_classifier.fit(X_train, y_train)

    #calculate the model accuracy
    y_pred = rf_classifier.predict(X_test)
    rf_accuracy = accuracy_score(y_test, y_pred)

    #store the results
    rf_results.append([i, rf_accuracy])

rf_results = pd.DataFrame(rf_results)
rf_results.columns = ['forest_size', 'accuracy']
rf_results.head()
```

Out[11]:

	forest_size	accuracy
0	1	0.747475
1	2	0.868687
2	3	0.898990
3	4	0.888889
4	5	0.919192

Out-of-bag (OOB) Random forest

Random forest with out-of-bag (OOB) sample error estimation.

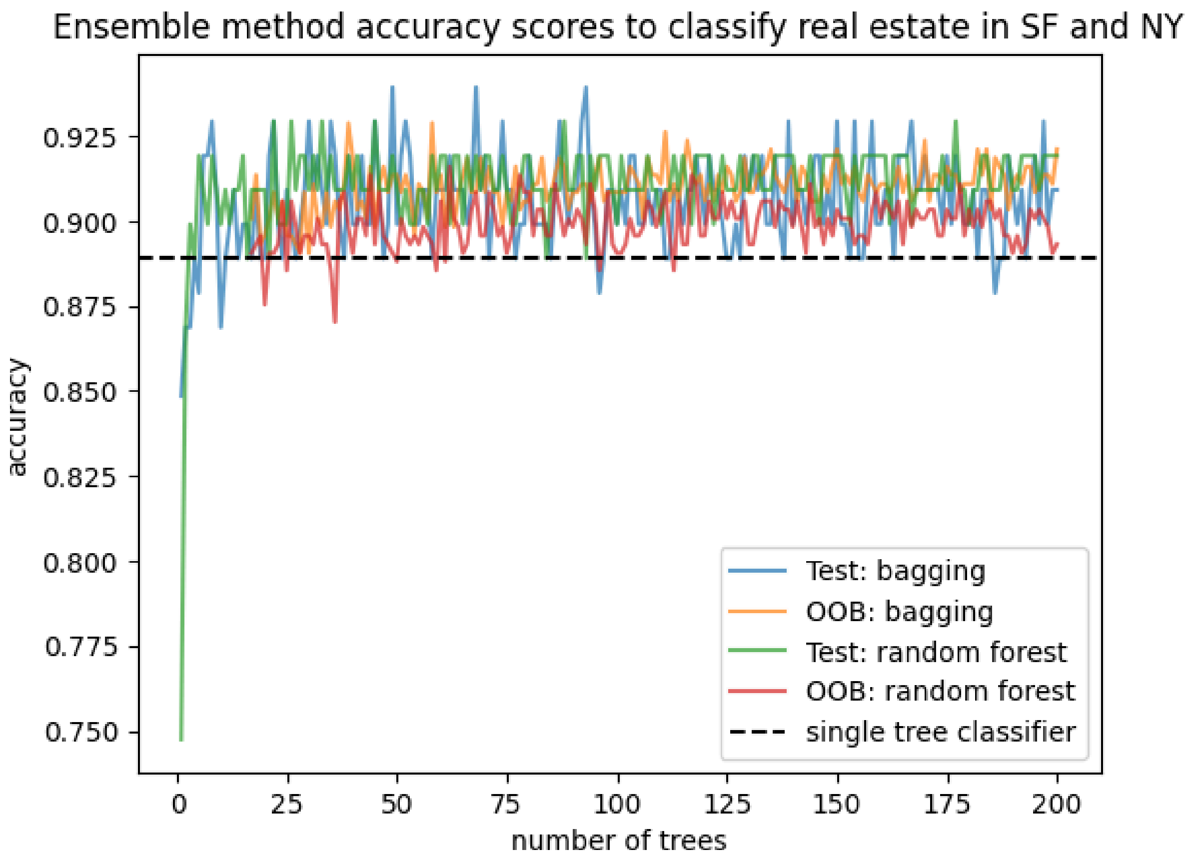
In [14]:
```
#create the random forest classifier
oob_rf_classifier = RandomForestClassifier(oob_score = True, n_jobs = -1)

oob_rf_results = []
#for each forest size
for i in num_trees[16:]:
    #set the number of trees
    oob_rf_classifier.set_params(n_estimators = i)

    #fit the model
    oob_rf_classifier.fit(X_train, y_train)

    #calculate the model accuracy
    oob_rf_accuracy = oob_rf_classifier.oob_score_

    #store the results
    oob_rf_results.append([i, oob_rf_accuracy])

oob_rf_results = pd.DataFrame(oob_rf_results)
oob_rf_results.columns = ['forest_size', 'accuracy']
oob_rf_results.head()
```

Out[14]:

	forest_size	accuracy
0	17	0.890585
1	18	0.893130
2	19	0.895674
3	20	0.875318
4	21	0.890585

In [15]:
```
#plot the test performance results
sns.lineplot(data = bagging_results, x = 'forest_size', y = 'accuracy', alpha = 0.7
sns.lineplot(data = oob_bagging_results, x = 'forest_size', y = 'accuracy', alpha =
sns.lineplot(data = rf_results, x = 'forest_size', y = 'accuracy', alpha = 0.7,  la
```

```
sns.lineplot(data = oob_rf_results, x = 'forest_size', y = 'accuracy', alpha = 0.7,
plt.axhline(single_tree_accuracy, linestyle = "--", color = 'black', label = 'singl
plt.xlabel('number of trees')
plt.ylabel('accuracy')
plt.title('Ensemble method accuracy scores to classify real estate in SF and NY')
plt.legend()
plt.show()
```



Conclusion

All Ensemble methods including bagging, OOB bagging, random forest, and OOB random forest improved the predictive accuracy when compared to the base decision tree classifier. The improved accuracy can be observed when the number of trees gets larger (above ~25 trees). OOB bagging performed slightly better than bagging, and random forests performed the best out of all the methods. OOB random forest performed the worst, likely because of the smaller sample size resulting in increased bias and decreased variance leading to poorer performance accuracy.