

▼ Abstract

The purpose of this analysis is to look into the Poker Hand Data set and analyze the strenght of each hand as a classification exercise. This dataset has already been devided into two sets of train and test dataset. Hence there is no need to split the data before starting the analysis.

UCI Machine Leaning Repository describes the dataset as following:

"Each record is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes (suit and rank), for a total of 10 predictive attributes. There is one Class attribute that describes the Poker Hand. The order of cards is important, which is why there are 480 possible Royal Flush hands as compared to 4.

Attribute Information:

- 1) S1 "Suit of card #1" Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
- 2) C1 "Rank of card #1" Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
- 3) S2 "Suit of card #2" Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
- 4) C2 "Rank of card #2" Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
- 5) S3 "Suit of card #3" Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
- 6) C3 "Rank of card #3" Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
- 7) S4 "Suit of card #4" Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
- 8) C4 "Rank of card #4" Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
- 9) S5 "Suit of card #5" Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
- 10) C5 "Rank of card 5" Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
- 11) CLASS "Poker Hand" Ordinal (0-9)

0: Nothing in hand; not a recognized poker hand 1: One pair; one pair of equal ranks within five cards 2: Two pairs; two pairs of equal ranks within five cards 3: Three of a kind; three equal ranks within five cards 4: Straight; five cards, sequentially ranked with no gaps 5: Flush; five cards with the same suit 6: Full house; pair + different rank three of a kind 7: Four of a kind; four equal ranks within five cards 8: Straight flush; straight + flush 9: Royal flush; {Ace, King, Queen, Jack, Ten} + flush"

We first started with Data Preparation and Pre-prediction Exploratory Data Analysis. For that we made use of functions such as "describe()" which helps to view some basic statiscital details of our data set, "shape" which returns the dimesions of our dataset, "info" which gives the summary of our dataset and "head()" which shows the data of the top five rows from our dataset. We have also plotted the histogram for each of our test and train dataset. The results of this exploratory analysis allowed us to become familiar with our dataset before starting the pipeline process.

To build the pipeline, the data were normalized using "StandardScaler()". This function helps us to make sure that all of our dataset features have the same weight and importance.

Since the category membership of our dataset is known, we are going to use the classification technique on our dataset. We have used models such KNN, SVM, RandomForest, Adaboost, Gradient Boosting and Bagging to analyze the dataset, find their accuracy and predict the labels.

Before fitting the data into the models, hyperparameters were used to train the data even more effectively.

The accuracy of each model was plotted and the results show that MLP has the highest accuracy to predicate the correct strength of the poker hand compared to other models that we have used.

```
import pandas as pd
import numpy as np
from sklearn import tree, svm
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, BaggingClassifier, GradientBoostingClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
```

```
#load in test/train datasets
train = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker-ha
test = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker-ha
```

```
#create column headers
train.columns = ['S1', 'C1', 'S2', 'C2', 'S3', 'C3', 'S4', 'C4', 'S5', 'C5', 'Label']
test.columns = ['S1', 'C1', 'S2', 'C2', 'S3', 'C3', 'S4', 'C4', 'S5', 'C5', 'Label']
```

▼ Train set inspection

```
train.head()
```



```
train.info()
```



```
train.describe()
```



```
train.shape
```



▼ Test set inspection

```
test.head()
```



```
test.info()
```



```
test.describe()
```



```
test.shape
```



```
train['Label'].hist()
```



```
test['Label'].hist()
```



▼ Extract features and labels

```
X_train = train.loc[:,train.columns != 'Label']  
X_test = test.loc[:,test.columns != 'Label']
```

▼ Scaling the Data

```
scaler = StandardScaler()  
  
# Fit only to the training data  
scaler.fit(X_train)  
  
# Transform the training and testing data  
X_train = scaler.transform(X_train)
```

```
X_test = scaler.transform(X_test)
y_train = train['Label']
y_test = test['Label']
```



▼ KNearestNeighbours

Another model that we decided to investigate for this classification model is the K-Nearest Neighbor. We used the param_grids hyperparameters of n_neighbors 'leaf_size' and 'n_jobs'. Also we tested 2 cross validations of CV=3 and 10 for this model and it resulted in CV=10 providing slightly better accuracy.

CV=3

```
#Instantiating the KNN model
knn = KNeighborsClassifier()
param_grid_KNN = {'n_neighbors':[5,10,40,80,100,150,200],
                  'leaf_size':[1,2,3,5],
                  'n_jobs':[-1]}
model_names = ["KNN"]
knn_grid_search3 = RandomizedSearchCV(estimator = knn, param_distributions = param_grid_KNN,
                                     cv = 3, n_jobs = -1, verbose = 2)
#Computing 3-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(knn_grid_search3, X_train, y_train, cv=3)
print(cv_scores)
```



```
#fitting the KNN model
knn_grid_search3.fit(X_train, y_train)
print(knn_grid_search3.best_params_)
best_random =knn_grid_search3.best_estimator_
best_random.fit(X_train, y_train)
y_pred3=best_random.predict(X_test)

#print Accuracy
acc = accuracy_score(y_test, y_pred3)
print("Accuracy Using" + str(acc)+'\n')
print(classification_report(y_test,y_pred3))
print(confusion_matrix(y_test, y_pred3))
```

Fitting 3 folds for each of 10 candidates, totalling 30 fits [Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers. /usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning {'n_neighbors': 150, 'n_jobs': -1, 'leaf_size': 1} [Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 6.3min finished

Accuracy Using 0.547687

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. 'precision', 'predicted', average, warn_for) precision recall f1-score support

0	0.56	0.81	0.66	501209
1	0.51	0.33	0.40	422498
2	0.00	0.00	0.00	47622
3	0.00	0.00	0.00	21121
4	0.00	0.00	0.00	3885
5	0.00	0.00	0.00	1996
6	0.00	0.00	0.00	1424
7	0.00	0.00	0.00	230
8	0.00	0.00	0.00	12
9	0.00	0.00	0.00	3
micro avg	0.55	0.55	0.55	1000000
macro avg	0.11	0.11	0.11	1000000
weighted avg	0.50	0.55	0.50	1000000

[[406572	94637	0	0	0	0	0	0	0	0]
[281383	141115	0	0	0	0	0	0	0	0]
[25798	21824	0	0	0	0	0	0	0	0]
[8674	12447	0	0	0	0	0	0	0	0]
[434	3451	0	0	0	0	0	0	0	0]
[1848	148	0	0	0	0	0	0	0	0]
[508	916	0	0	0	0	0	0	0	0]
[20	210	0	0	0	0	0	0	0	0]
[4	8	0	0	0	0	0	0	0	0]
[3	0	0	0	0	0	0	0	0	0]]

```
#setup arrays to store train and test accuracies for CV=3
n_neighbors_KNN = [5,10,40,80,100,150,200]
train_accuracy = np.empty(len(n_neighbors_KNN))
test_accuracy = np.empty(len(n_neighbors_KNN))
```

```
#loop over different values of k
for i, k in enumerate(n_neighbors_KNN):
```

```

#setup a Random forest Classifier with k neighbors: clf
clf_KNN = RandomizedSearchCV(estimator = KNeighborsClassifier(k), param_distributions = p
                             cv = 3, n_jobs = -1, verbose = 2)

#fit the classifier to the training data
clf_KNN.fit(X_train, y_train)

#compute accuracy on the training set
train_accuracy[i] = clf_KNN.score(X_train, y_train)

#compute accuracy on the testing set
test_accuracy[i] = clf_KNN.score(X_test, y_test)

#generate plot CV=3
plt.title('KNN: Varying Number of estimators')
plt.plot(n_neighbors_KNN, test_accuracy, label = 'Testing Accuracy')
plt.plot(n_neighbors_KNN, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()

```

Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706:

UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 7.3min finished

Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706:

UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 6.7min finished

Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706:

UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 6.3min finished

Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 8.0min finished

Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 7.0min finished

Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

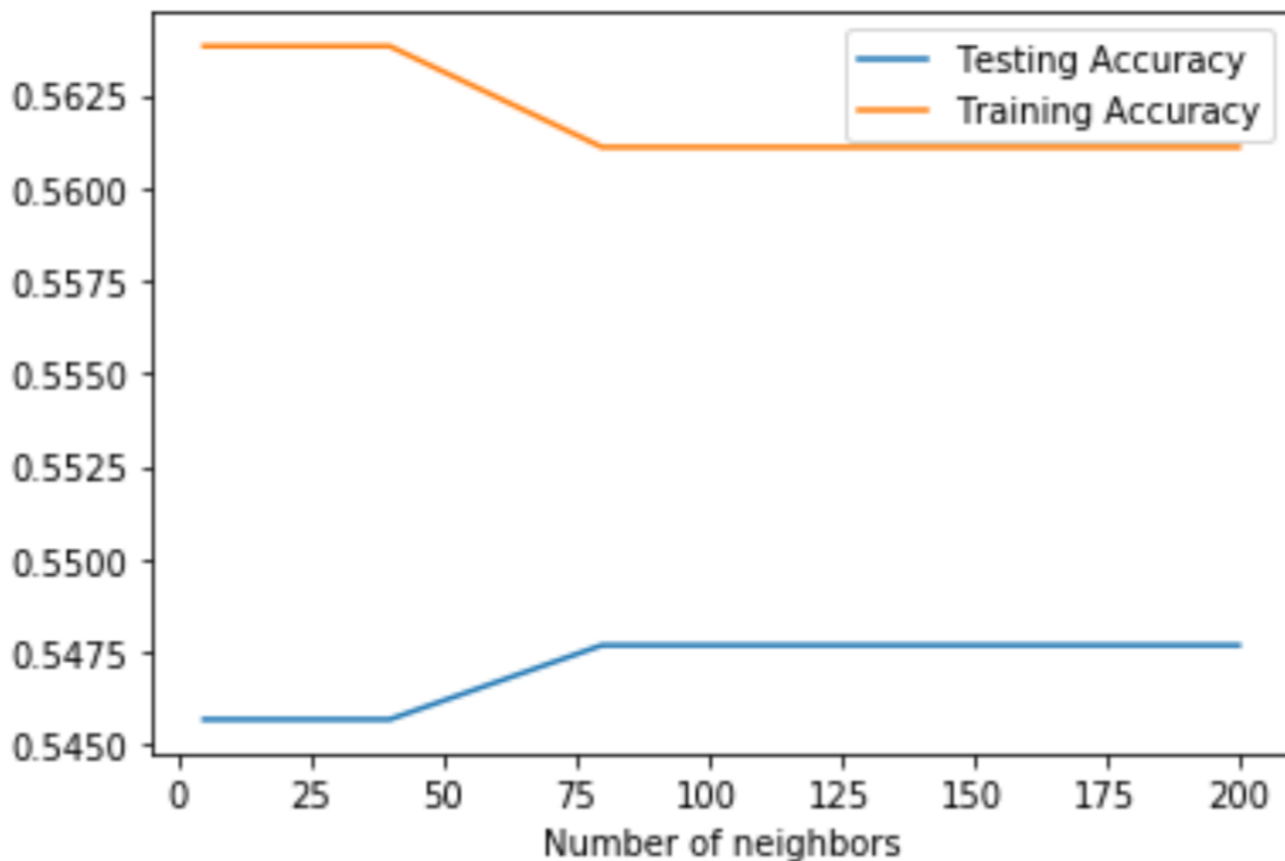
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 8.2min finished

Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 8.1min finished

KNN: Varying Number of Neighbors, CV=3



CV=10

```
#instantiating the KNN model
knn = KNeighborsClassifier()
param_grid_KNN = {'n_neighbors': [5, 10, 40, 80, 100, 150, 200],
                  'leaf_size': [1, 2, 3, 5],
                  'n_jobs': [-1]}
model_names = ["KNN"]
knn_grid_search10 = RandomizedSearchCV(estimator = knn, param_distributions = param_grid_KNN,
                                       cv = 10, n_jobs = -1, verbose = 2)
#computing 3-fold cross-validation scores: cv_scores
cv_scores10 = cross_val_score(knn_grid_search10, X_train, y_train, cv=3)
print(cv_scores10)
```

Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least populated class in y has only 3 members, which is too few. The minimum number of members in any class cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 6.1min

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 21.0min finished

Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least populated class in y has only 3 members, which is too few. The minimum number of members in any class cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 4.3min

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 14.8min finished

Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least populated class in y has only 4 members, which is too few. The minimum number of members in any class cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 6.6min

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 13.0min finished

[0.54581434 0.54360082 0.54745051]

```
#fitting the KNN model
knn_grid_search10.fit(X_train, y_train)
print(knn_grid_search10.best_params_)
best_random =knn_grid_search10.best_estimator_
best_random.fit(X_train, y_train)
y_pred10=best_random.predict(X_test)
```

```
# Print Accuracy
acc = accuracy_score(y_test, y_pred10)
print("Accuracy Using" + str(acc)+'\n')
print(classification_report(y_test,y_pred10))
print(confusion_matrix(y_test, y_pred10))
```

Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least populated class in y has only 5 members, which is too few. The minimum number of members in any class cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 11.8min

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706:

UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

```
{'n_neighbors': 200, 'n_jobs': -1, 'leaf_size': 1}
```

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 29.9min finished

Accuracy Using 0.54831

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. 'precision', 'predicted', average, warn_for) precision recall f1-score support

0	0.56	0.83	0.67	501209
1	0.52	0.32	0.39	422498
2	0.00	0.00	0.00	47622
3	0.00	0.00	0.00	21121
4	0.00	0.00	0.00	3885
5	0.00	0.00	0.00	1996
6	0.00	0.00	0.00	1424
7	0.00	0.00	0.00	230
8	0.00	0.00	0.00	12
9	0.00	0.00	0.00	3
micro avg	0.55	0.55	0.55	1000000
macro avg	0.11	0.11	0.11	1000000
weighted avg	0.50	0.55	0.50	1000000

[413514	87695	0	0	0	0	0	0	0	0]
[287702	134796	0	0	0	0	0	0	0	0]
[26292	21330	0	0	0	0	0	0	0	0]
[9041	12080	0	0	0	0	0	0	0	0]
[487	3398	0	0	0	0	0	0	0	0]
[1862	134	0	0	0	0	0	0	0	0]
[517	907	0	0	0	0	0	0	0	0]
[26	204	0	0	0	0	0	0	0	0]
[4	8	0	0	0	0	0	0	0	0]
[3	0	0	0	0	0	0	0	0	0]]

#setup arrays to store train and test accuracies for CV=10

```
n_neighbors_KNN = [5,10,40,80,100,150,200]
```

```
train_accuracy = np.empty(len(n_neighbors_KNN))
```

```
test_accuracy = np.empty(len(n_neighbors_KNN))
```

#loop over different values of k

```
for i, k in enumerate(n_neighbors_KNN):
```

```
    #setup a Random forest Classifier with k neighbors: clf
```

```
    clf_KNN = RandomizedSearchCV(estimator = KNeighborsClassifier(k), param_distributions = p
                                cv = 10, n_jobs = -1, verbose = 2)
```

```
    #fit the classifier to the training data
```

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

```

#compute accuracy on the training set
train_accuracy[i] = clf_KNN.score(X_train, y_train)

#compute accuracy on the testing set
test_accuracy[i] = clf_KNN.score(X_test, y_test)

#generate KNN plot with CV=10
plt.title('KNN: Varying Number of estimators, CV=10')
plt.plot(n_neighbors_KNN, test_accuracy, label = 'Testing Accuracy')
plt.plot(n_neighbors_KNN, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()

```

Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least populated class in y has only 5 members, which is too few. The minimum number of members in any class cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 7.9min

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 26.2min finished

Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least populated class in y has only 5 members, which is too few. The minimum number of members in any class cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 10.6min

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 24.5min finished

Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least populated class in y has only 5 members, which is too few. The minimum number of members in any class cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 11.7min

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 27.9min finished

Fitting 10 folds for each of 10 candidates, totalling 100 fits

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least
populated class in y has only 5 members, which is too few. The minimum number of members in any class
cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706:
UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 8.1min

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 27.3min finished

Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least
populated class in y has only 5 members, which is too few. The minimum number of members in any class
cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 7.7min

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706:
UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 25.1min finished

Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least
populated class in y has only 5 members, which is too few. The minimum number of members in any class
cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706:
UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 13.2min

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 32.7min finished

Fitting 10 folds for each of 10 candidates, totalling 100 fits

/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least
populated class in y has only 5 members, which is too few. The minimum number of members in any class
cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning)

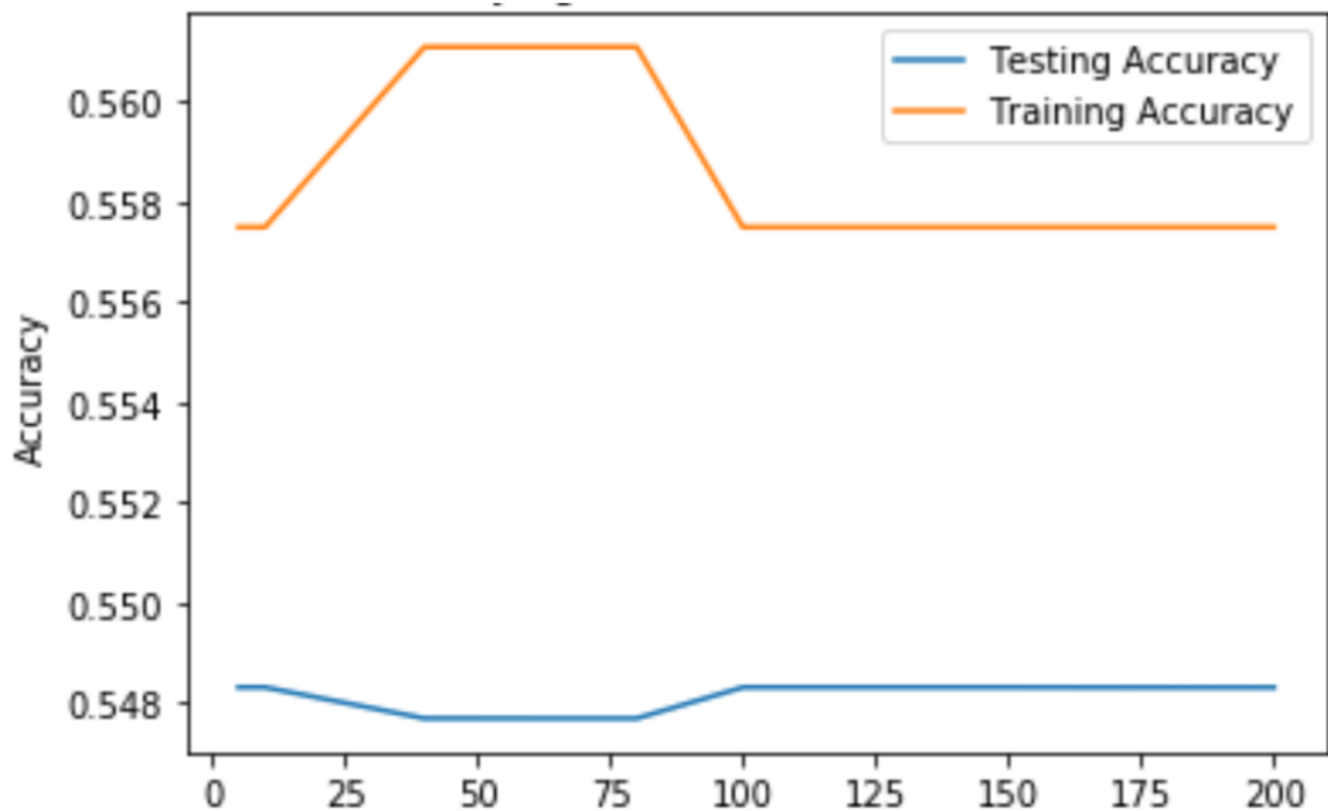
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706:
UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak. "timeout or by a memory leak.", UserWarning

[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 7.8min
```

[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 26.9min finished

KNN: Varying Number of Neighbors, CV=10



▼ SVM

We also used a multiclass svm classifier. We tested this on 3 and 10 fold cross validation. The hyperparameters tuned were the penalty parameter C, the kernel coefficient for Gamma, and the type of decision function (One vs Rest). SVM is best used with small to medium sized datasets, and in the case of this dataset, it doesn't seem to work a well. The 3 CV SVM model seems to perform better on the test set than the 10 CV model, albeit by a fraction of a percent.

CV=3

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingClassifier

param_grid_SVM = {'#penalty':['l1', 'l2'],
                  'C':[0.1, 1, 10],
                  'gamma':[0.00001, 0.0001, 0.001, 0.01, 0.1],
                  'decision_function_shape': ['ovr']}

#instantiating the svm model
svm = SVC()

#GridSearchCV
svm_grid_search3 = RandomizedSearchCV(estimator = svm, param_distributions = param_grid_SVM,
                                     cv = 3, n_jobs = -1, verbose = 2)
```

```
svm_3 = svm_grid_search3.fit(X_train, y_train)
print("Best CV3 params", svm_3.best_params_)
print("Best CV3 accuracy", svm_3.best_score_)

y_pred_svm_3 = svm_3.predict(X_test)
accuracy_svm_3 = accuracy_score(y_test, y_pred_svm_3)
print('Accuracy of SVM_3 Classifier: {:.3f}'.format(accuracy_svm_3))
```



```
#checking the number of labels
np.unique(y_pred_svm_3)
```

```
plt.plot(y_test, 'bo', label='actual')
plt.plot(y_pred_svm_3, 'ro', label='predicted')
plt.legend()
plt.show()
```



CV =10

```
param_grid_SVM = {'penalty':['l1', 'l2'],
                  'C':[0.1, 1, 10],
                  'gamma':[0.00001, 0.0001, 0.001, 0.01, 0.1],
                  'decision_function_shape': ['ovr']}

svm_grid_search10 = RandomizedSearchCV(estimator = svm, param_distributions = param_grid_SVM,
                                       cv = 10, n_jobs = -1, verbose = 2)
```

```
svm_10 = svm_grid_search3.fit(X_train, y_train)
print("Best CV10 params", svm_10.best_params_)
print("Best CV10 accuracy", svm_10.best_score_)
y_pred_svm_10 = svm_10.predict(X_test)
accuracy_svm_10 = accuracy_score(y_test, y_pred_svm_10)
print('Accuracy of SVM_10 Classifier: {:.3f}'.format(accuracy_svm_10))
```

Fitting 10 folds for each of 10 candidates, totalling 100 fits [Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers. [Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 120.2min finished

Best CV10 params {'C': 1, 'decision_function_shape': 'ovr', 'gamma': 0.1}

Best CV10 accuracy 0.5537194322271092

'Accuracy of SVM_10 Classifier: 0.557

```
z = confusion_matrix(y_test.values, y_pred_svm_3)
z
```



```
np.unique(y_pred_svm_10)
```



```
np.unique(X_test)
```




```
np.unique(y_test)
```



```
plt.plot(y_test, 'bo', label='actual')  
plt.plot(y_pred_svm_10, 'ro', label='predicted')  
plt.legend()  
plt.show()
```



▼ Random Forest

Here we loop over different values of `n_estimators` and plot a graph of training vs test accuracy for each one.

We can see that the highest accuracy occurs at `n_estimators=25` so that's the one we choose for our cross validation.

We then run a 3-fold cross validation, getting an average accuracy score of 0.576.

We also ran a 10-fold cross validation getting an average score of 0.580, not that much better, but slightly better than 3-fold.

```
#setup arrays to store train and test accuracies
estimators = [5,10,15,20,25]
train_accuracy = np.empty(len(estimators))
test_accuracy = np.empty(len(estimators))

#loop over different values of k
for i, k in enumerate(estimators):
    #setup a Random forest Classifier with k estimators: rf
    rf = RandomForestClassifier(n_estimators=k)

    #fit the classifier to the training data
    rf.fit(X_train, y_train)

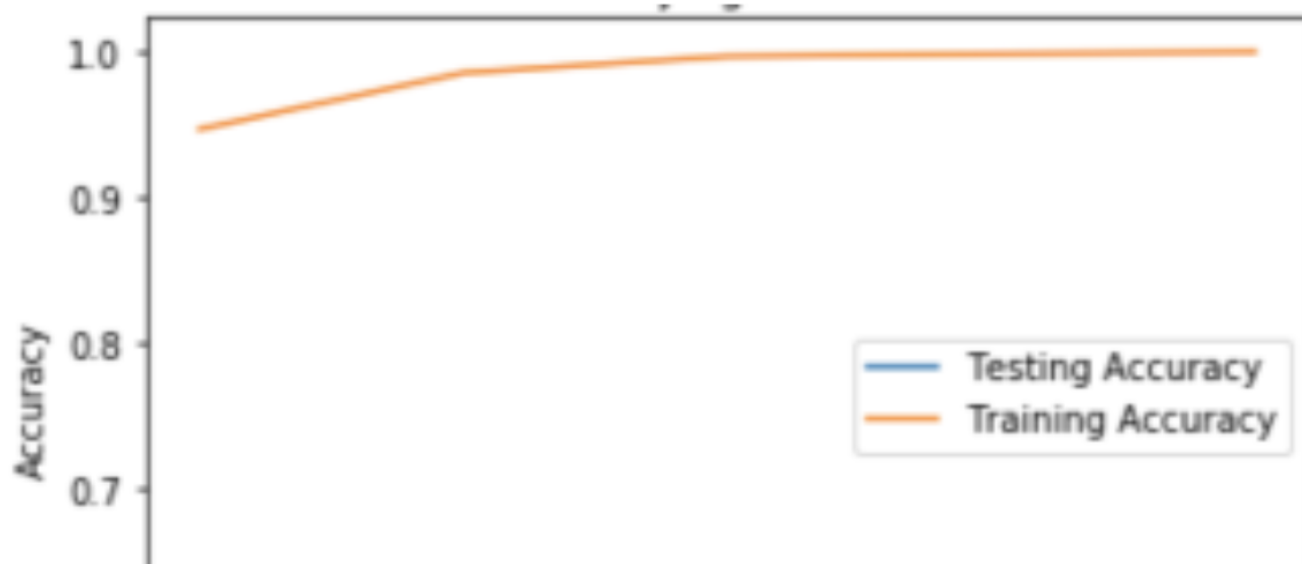
    #compute accuracy on the training set
    train_accuracy[i] = rf.score(X_train, y_train)

    #compute accuracy on the testing set
    test_accuracy[i] = rf.score(X_test, y_test)

#generate plot

plt.title('Random Forest: Varying Number of estimators')
plt.plot(estimators, test_accuracy, label = 'Testing Accuracy')
plt.plot(estimators, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of estimators')
plt.ylabel('Accuracy')
plt.show()
```

RandomForest: Varying Number of Estimators



CV=3

```
#cross validation of rf scores
rf=RandomForestClassifier(n_estimators=25)

#computing 3-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(rf, X_train, y_train, cv=3)

#print the 3-fold cross-validation scores
print(cv_scores)

print("Average 3-Fold CV Score: {}".format(np.mean(cv_scores)))
```

Average 3-Fold CV Score: 0.576

CV=10

```
#cross-validation with 10-folds
rf=RandomForestClassifier(n_estimators=25)

#computing 10-fold cross-validation scores: cv_scores
cv_scores = cross_val_score(rf, X_train, y_train, cv=10)

#print the 10-fold cross-validation scores
print(cv_scores)

print("Average 10-Fold CV Score: {}".format(np.mean(cv_scores)))
```

Average 10-Fold CV Score: 0.580

▼ Decision Tree

Here we import and set up a parameter grid for a Decision Tree Classifier.

For hyper parameter tuning: We run a randomized search CV with cv=3 and 50 iterations to see what the best parameters for our decision tree are. We also run a randomized search CV with cv=10 as well.

Looking at the accuracy scores we see that using cv10 vs cv3 has only increased our accuracy score by ~0.0016 which is quite an insignificant change so it's probably better to use CV3 as it takes less time to run.

```
dt = DecisionTreeClassifier()
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(dt.get_params())
parameters={'min_samples_split' : range(10,500,20),
            'max_depth': range(1,20,2)}
```

CV=3

```
# Use the random grid to search for best hyperparameters usin 3 fold cross validation
dt_random3 = RandomizedSearchCV(estimator = dt, param_distributions = parameters, n_iter = 50)
# Fit the random search model
dt_random3.fit(X_train, y_train)
```

```
#printing best 3 fold cross validation score and parameters
print("Best CV 3 params", dt_random3.best_params_)
print("Best CV 3 accuracy", dt_random3.best_score_)
```

```
#extract best estimator from 3-fold Randomized Search CV
dt_best3 = dt_random3.best_estimator_
```

```
#use the random grid to search for best hyperparameters
dt_random10 = RandomizedSearchCV(estimator = dt, param_distributions = parameters, n_iter = 5)
#uit the random search model
dt_random10.fit(X_train, y_train)
```



CV=10

```
#printing best 10 fold cross validation score and parameters
print("Best CV 10 params", dt_random10.best_params_)
print("Best CV 10 accuracy", dt_random10.best_score_)
```



```
#extract best estimator from 10-fold Randomized Search CV
dt_best10 = dt_random10.best_estimator_
```

▼ Bagging Classifier

We then fit a bagging classifier using the best decision tree model from cv3, predicted on the test set and computed the accuracy getting an accuracy of 0.609

We also fit a bagging classifier using the best decision tree from cv10, predicted on the test set and computed the accuracy getting an accuracy of 0.599.

Again, this is consistent with our earlier decision to go with 3fold cross validated decision tree as the base estimator.

```
#setup arrays to store train and test accuracies
```

```
estimators = [10, 20, 50, 100, 200, 500]
train_accuracy = np.empty(len(estimators))
test_accuracy = np.empty(len(estimators))

#loop over different values of k
for i, k in enumerate(estimators):
    #setup a Random forest Classifier with k estimators: clf
    bcf = BaggingClassifier(n_estimators=k)

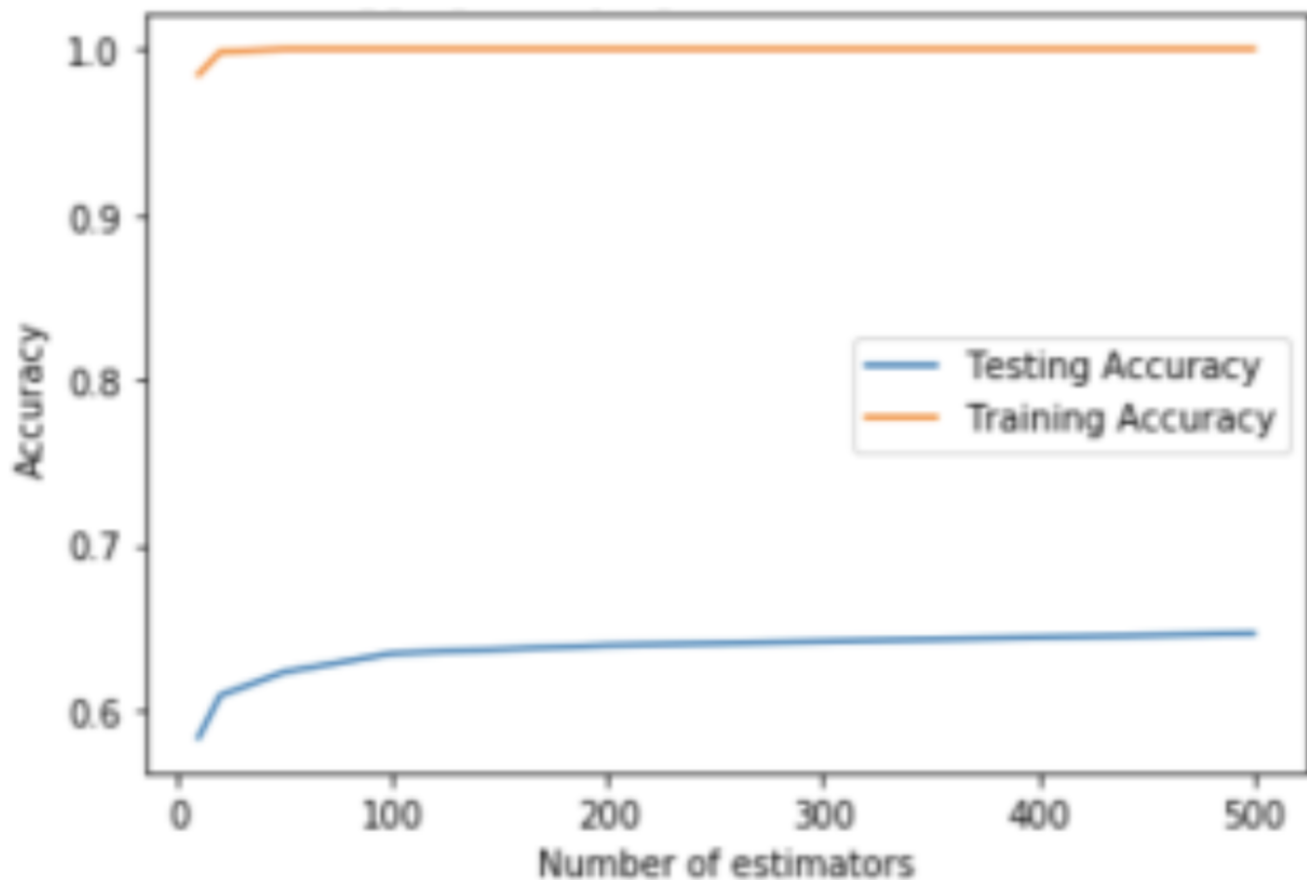
    #fit the classifier to the training data
    bcf.fit(X_train, y_train)

    #compute accuracy on the training set
    train_accuracy[i] = bcf.score(X_train, y_train)

    #compute accuracy on the testing set
    test_accuracy[i] = bcf.score(X_test, y_test)

#generate plot
plt.title('Bagging: Varying Number of estimators')
plt.plot(estimators, test_accuracy, label = 'Testing Accuracy')
plt.plot(estimators, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of estimators')
plt.ylabel('Accuracy')
plt.show()
```

Bagging : Varying Number of Estimators



CV=3

```
bc3 = BaggingClassifier(base_estimator=dt_best3, n_estimators=500, n_jobs=-1)

#fit Bagging Classifier using 3-fold cross validated Decision Tree as the base estimator
bc3.fit(X_train, y_train)
```



```
y_pred3=bc3.predict(X_test)
accuracy3 = accuracy_score(y_test, y_pred3)
print('Accuracy of Bagging Classifier using 3 fold cross validated Decision Tree: {:.3f}'.for
```

CV=10

```
bc10 = BaggingClassifier(base_estimator=dt_best10, n_estimators=500, n_jobs=-1)
#fit Bagging Classifier using 10-fold cross validated Decision Tree as the base estimator
bc10.fit(X_train, y_train)
```

```
y_pred10=bc10.predict(X_test)
accuracy10 = accuracy_score(y_test, y_pred10)
print('Accuracy of Bagging Classifier using 10 fold cross validated Decision Tree: {:.3f}'.fc
```

▼ AdaBoost Classifier

Here we run a for loop to iter through a list of n_estimators and compute the accuracy of the adaboost classifier accordingly.

We choose n_estimators=20 as there doesn't seem to be a difference in accuracy as we increase the number of estimators.

We then fit the ABD classifier using our 3 fold cross validated decision tree and compute the accuracy which turns out to be 0.513.

We also fit it using our 10 fold cross validated decision tree and compute the accuracy to find out it's 0.491, so again, we are better off using our 3 fold cross validated decision tree as our base estimator.

```
#setup arrays to store train and test accuracies
estimators = [10, 20, 50, 70, 100]
train_accuracy = np.empty(len(estimators))
test_accuracy = np.empty(len(estimators))

#loop over different values of k
for i, k in enumerate(estimators):
    #setup a Random forest Classifier with k estimators: clf
    adb_clf = AdaBoostClassifier(n_estimators=k)

    #fit the classifier to the training data
    adb_clf.fit(X_train, y_train)

    #compute accuracy on the training set
    train_accuracy[i] = adb_clf.score(X_train, y_train)

    #compute accuracy on the testing set
    test_accuracy[i] = adb_clf.score(X_test, y_test)

    #generate plot
plt.title('AdaBoost: Varying Number of estimators')
plt.plot(estimators, test_accuracy, label = 'Testing Accuracy')
plt.plot(estimators, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of estimators')
plt.ylabel('Accuracy')
plt.show()
```



CV=3

```
adb3 = AdaBoostClassifier(base_estimator=dt_best3, n_estimators=20)
adb3.fit(X_train, y_train)
```

```
y_pred_adb3=adb3.predict(X_test)
accuracyadb3 = accuracy_score(y_test, y_pred_adb3)
print('Accuracy of 3 fold CV AdaBoost Classifier: {:.3f}'.format(accuracyadb3))
```

Accuracy of 3 fold CV AdaBoost Classifier: 0.513

CV=10

```
adb10 = AdaBoostClassifier(base_estimator=dt_best10, n_estimators=20)
adb10.fit(X_train, y_train)
```

```
y_pred_adb10=adb10.predict(X_test)
accuracyadb10 = accuracy_score(y_test, y_pred_adb10)
print('Accuracy of 10 fold CV AdaBoost Classifier: {:.3f}'.format(accuracyadb10))
```

Accuracy of 10 fold CV AdaBoost Classifier: 0.491

▼ Gradient Boosting Classifier

Here we print the parameters used by our current model. We then set a parameter grid with values for hyperparameter tuning.

We then run a RandomizedSearchCV using our parameter with 3-fold CV and fit it to our training data to get the best hyperparameters for our model.

We then print the best parameters and extract the best model and use it to predict on X_test.

We bring the accuracy getting a training accuracy of ... and a test accuracy of ...

We did not run 10-fold CV randomized search as it took way too long to run for this dataset.

```
clf = GradientBoostingClassifier(random_state = 13)
from pprint import pprint
#Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(clf.get_params())
```

```
#number of trees in random forest
n_estimators = [10, 20, 50]
#number of features to consider at every split
max_features = ['auto', 'sqrt']
#maximum number of levels in tree
max_depth = [10, 20, 50]
max_depth.append(None)
#minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
#minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
#create the random grid
random_grid1 = {'n_estimators': n_estimators,
                 'max_features': max_features,
                 'max_depth': max_depth,
                 'min_samples_split': min_samples_split,
                 'min_samples_leaf': min_samples_leaf}
pprint(random_grid1)
```



```
#use the random grid to search for best hyperparameters
#first create the base model to tune
clf = GradientBoostingClassifier()
#random search of parameters, using 3 fold cross validation,
#search across 100 different combinations, and use all available cores
clf_random = RandomizedSearchCV(estimator = clf, param_distributions = random_grid1, n_iter =
#fit the random search model
clf_random.fit(X_train, y_train)
```



```
clf_random.best_params_
```



```
best_clf = clf_random.best_estimator_

y_pred13=best_clf.predict(X_test)
print('Testing Accuracy with 3 fold CV = {:.2f}%'.format((best_clf.score(X_test, y_test)*100))
print('Training Accuracy with 3 fold CV = {:.2f}%'.format((best_clf.score(X_train, y_train)
```



▼ MLP

We decided to try and use Multilayered Perceptrons on the dataset as well. We went this model as it is an artificial neural network that tends to perform well on classification problems, and with tabular data. It also performs well as a base neural network model. We found that with this model, both cv3 and cv10 perform exceptionally well (~99%), but when looking at the distribution of the predictions, we noticed that cv3 seems to be doing better on cards with higher strengths.

CV = 3

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import RandomizedSearchCV

params_grid = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}

mlp = MLPClassifier()

mlp_cv3 = RandomizedSearchCV(estimator = mlp, param_distributions = params_grid, cv = 3, n_jc
mlp_cv3.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits [Parallel(n_jobs=-1)]: Using backend LokyBackend
with 2 concurrent workers. [Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 32.6min finished
/usr/local/lib/python3.6/dist-packages/sklearn/neural_network/multilayer_perceptron.py:562:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't
converged yet. % self.max_iter, ConvergenceWarning) RandomizedSearchCV(cv=3, error_score='raise-
deprecating', estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant',
learning_rate_init=0.001, max_iter=200, momentum=0.9, n_iter_no_change=10,
nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001,
validation_fraction=0.1, verbose=False, warm_start=False), fit_params=None, iid='warn', n_iter=10,
n_jobs=-1, param_distributions={'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)], 'activation': ['tanh',
'relu'], 'solver': ['sgd', 'adam'], 'alpha': [0.0001, 0.05], 'learning_rate': ['constant', 'adaptive']},
pre_dispatch='2*n_jobs', random_state=42, refit=True, return_train_score='warn', scoring=None, verbose=2)
```

```
mlp_cv3.get_params_
```

```
{'activation': 'tanh',
'alpha': 0.05,
'hidden_layer_sizes': (50, 100, 50),
'learning_rate': 'adaptive',
'solver': 'adam'}
```

```
mlp_cv3.get_score_
```

```
0.9534586165533786
```

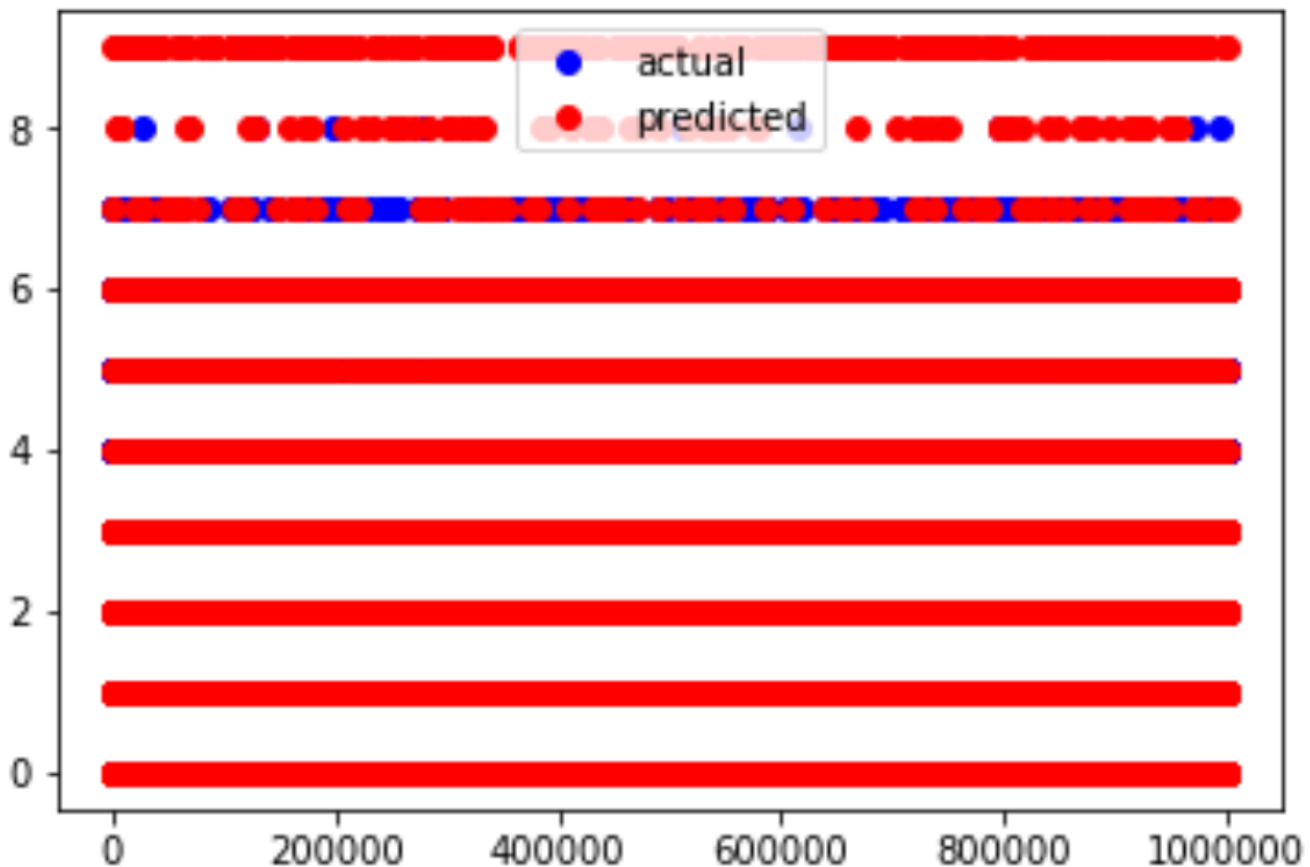
```
y_pred_mlp3 = clf.predict(X_test)
```

```
acc_mlp3 = accuracy_score(y_test, y_pred_mlp3)
print(acc_mlp3)
```

```
0.993452
```

```
plt.plot(y_test, 'bo', label='actual')
plt.plot(y_pred_mlp3, 'ro', label='predicted')
plt.legend()
plt.show()
```

MLP: Varying Number of Estimators



CV = 10

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import RandomizedSearchCV

params_grid = {
    'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}

mlp = MLPClassifier()

mlp_cv10 = RandomizedSearchCV(estimator = mlp, param_distributions = params_grid, cv = 10, n_
mlp_cv10.fit(X_train, y_train)
```

Fitting 10 folds for each of 10 candidates, totalling 100 fits /usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py:652: Warning: The least populated class in y has only 5 members, which is too few. The minimum number of members in any class cannot be less than n_splits=10. % (min_groups, self.n_splits)), Warning) [Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers. [Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 66.6min [Parallel(n_jobs=-1)]: Done

```
100 out of 100 | elapsed: 212.2min finished /usr/local/lib/python3.6/dist-
packages/sklearn/neural_network/multilayer_perceptron.py:562: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (200) reached and the optimization hasn't converged yet. % self.max_iter,
ConvergenceWarning) RandomizedSearchCV(cv=10, error_score='raise-deprecating',
estimator=MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999,
early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(100,), learning_rate='constant',
learning_rate_init=0.001, max_iter=200, momentum=0.9, n_iter_no_change=10,
nesterovs_momentum=True, power_t=0.5, random_state=None, shuffle=True, solver='adam', tol=0.0001,
validation_fraction=0.1, verbose=False, warm_start=False), fit_params=None, iid='warn', n_iter=10,
n_jobs=-1, param_distributions={'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50), (100,)], 'activation': ['tanh',
'relu'], 'solver': ['sgd', 'adam'], 'alpha': [0.0001, 0.05], 'learning_rate': ['constant', 'adaptive']},
pre_dispatch='2*n_jobs', random_state=42, refit=True, return_train_score='warn', scoring=None, verbose=2)
mlp_cv10.best_params_
```

```
{'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}
```

```
mlp_cv10.best_score_
```

```
0.9935225909636145
```

```
y_pred_mlp10 = mlp_cv10.predict(X_test)
```

```
acc_mlp10 = accuracy_score(y_test, y_pred_10)
print(acc_mlp10)
```

```
0.994437
```

```
plt.plot(y_test, 'bo', label='actual')
plt.plot(y_pred_mlp10, 'ro', label='predicted')
plt.legend()
plt.show()
```

