Abstract

The purpose of this analysis is to look into the Poker Hand hand as a classification exercise. This dataset has already test dataset. Hence there is no need to split the data befor

UCI Machine Leaning Repository describes the dataset as

"Each record is an example of a hand consisting of five pla 52. Each card is described using two attributes (suit and ra There is one Class attribute that describes the Poker Hand why there are 480 possible Royal Flush hands as compare

Attribute Information:

1) S1 "Suit of card #1" Ordinal (1-4) representing {Hearts, S

2) C1 "Rank of card #1" Numerical (1-13) representing (Ace

3) S2 "Suit of card #2" Ordinal (1-4) representing {Hearts, S

4) C2 "Rank of card #2" Numerical (1-13) representing (Ace

5) S3 "Suit of card #3" Ordinal (1-4) representing {Hearts, S

6) C3 "Rank of card #3" Numerical (1-13) representing (Ace

7) S4 "Suit of card #4" Ordinal (1-4) representing {Hearts, S

8) C4 "Rank of card #4" Numerical (1-13) representing (Ace

9) S5 "Suit of card #5" Ordinal (1-4) representing {Hearts, S

10) C5 "Rank of card 5" Numerical (1-13) representing (Ace

11) CLASS "Poker Hand" Ordinal (0-9)

0: Nothing in hand; not a recognized poker hand 1: One pai 2: Two pairs; two pairs of equal ranks within five cards 3: T five cards 4: Straight; five cards, sequentially ranked with n suit 6: Full house; pair + different rank three of a kind 7: Fou cards 8: Straight flush; straight + flush 9: Royal flush; {Ace,

We first started with Data Preparation and Pre-prediction E made use of functions such as “describe( )” which helps to data set, “shape” which returns the dimesions of our datas dataset and “head( )” which shows the data of the top five plotted the histogram for each of our test and train dataset allowed us to become familiar with our dataset before star

To build the pipeline, the data were normalized using “Stan make sure that all of our dataset features have the same w

Since the category membership of our dataset is known, w technique on our dataset. We have used models such KNN Boosting and Bagging to analyze the dataset, find their acc

Before fitting the data into the models, hyperparameters w
effectively.

The accuracy of each model was plotted and the results sh
predicate the correct strength of the poker hand compared

```
import pandas as pd
import numpy as np
from sklearn import tree, svm
from sklearn.metrics import accuracy_score, classif
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV, t
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearc
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precisi
from sklearn.tree import DecisionTreeClassifier
```

```
#load in test/train datasets
train = pd.read_csv('https://archive.ics.uci.edu/ml
test = pd.read_csv('https://archive.ics.uci.edu/ml/
```

```
#create column headers
train.columns = ['S1', 'C1','S2', 'C2','S3', 'C3',
test.columns = ['S1', 'C1','S2', 'C2','S3', 'C3','S
```

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 class
We used the param_grids hyperparameters of n_neighbors
cross validations of CV=3 and 10 for this model and it resu
accuracy.

CV=3

```
#Instantiating the KNN model
knn = KNeighborsClassifier()
param_grid_KNN ={'n_neighbors':[5,10,40,80,100,150,
          'leaf_size':[1,2,3,5],
          'n_jobs':[-1]}
model_names = ["KNN"]
knn_grid_search3 = RandomizedSearchCV(estimator = k
                              cv = 3, n_jobs = -1, ve
#Computing 3-fold cross-validation scores: cv_score
cv_scores = cross_val_score(knn_grid_search3, X_tra
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 [P
LokyBackend with 2 concurrent workers. /usr/local/lib/pyt
packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This
timeout or by a memory leak. "timeout or by a memory leal
'n_jobs': -1, 'leaf_size': 1} [Parallel(n_jobs=-1)]: Done 30 out

Accuracy Using0.547687

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/cl
UndefinedMetricWarning: Precision and F-score are ill-defi
predicted samples. 'precision', 'predicted', average, warn_fo

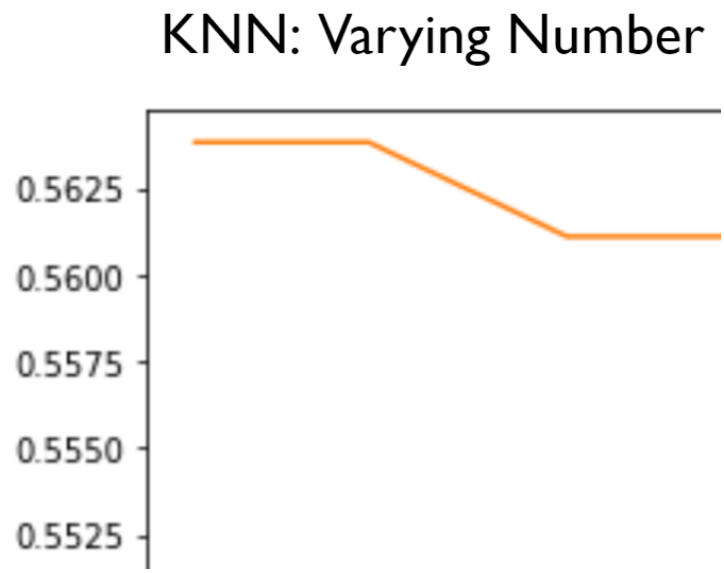
```
              0       0.56      0.81      0.66
              1       0.51      0.33      0.40
              2       0.00      0.00      0.00
              3       0.00      0.00      0.00
              4       0.00      0.00      0.00
              5       0.00      0.00      0.00
              6       0.00      0.00      0.00
              7       0.00      0.00      0.00
              8       0.00      0.00      0.00
              9       0.00      0.00      0.00

    micro avg       0.55      0.55      0.55
    macro avg       0.11      0.11      0.11
    weighted avg       0.50      0.55      0.50

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

```
       [     4      8      0      0      0
    0]
       [     3      0      0      0      0
    0]]
```

```
#setup arrays to store train and test accuracies fo
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 neighb
    clf_KNN = RandomizedSearchCV(estimator = KNeigh
                           cv = 3, n_jobs = -1, ve

    #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_tr

    #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 = 'T
plt.plot(n_neighbors_KNN, train_accuracy, label =
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 c
/usr/local/lib/python3.6/dist-
packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 7.3min fi
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
/usr/local/lib/python3.6/dist-
packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 6.7min fi
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 6.3min fi
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 8.0min fi
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 7.0min fi
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 8.2min fi
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
[Parallel(n_jobs=-1)]: Done 30 out of 30 | elapsed: 8.1min fi
```



CV=10

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

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
/usr/local/lib/python3.6/dist-packages/sklearn/model_sel
populated class in y has only 3 members, which is too few.
class cannot be less than n_splits=10. % (min_groups, self
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 6.1min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 21.0m
Fitting 10 folds for each of 10 candidates, totalling 100 fits
/usr/local/lib/python3.6/dist-packages/sklearn/model_sel
populated class in y has only 3 members, which is too few.
class cannot be less than n_splits=10. % (min_groups, self
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 4.3min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 14.8m
Fitting 10 folds for each of 10 candidates, totalling 100 fits
/usr/local/lib/python3.6/dist-packages/sklearn/model_sel
populated class in y has only 4 members, which is too few.
```

```
class cannot be less than n_splits=10. % (min_groups, self

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c

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

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

[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_sel
populated class in y has only 5 members, which is too few.
class cannot be less than n_splits=10. % (min_groups, self

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c

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

/usr/local/lib/python3.6/dist-
packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal

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

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

Accuracy Using0.54831

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/cl
UndefinedMetricWarning: Precision and F-score are ill-defi
predicted samples. 'precision', 'predicted', average, warn_fc

              0       0.56      0.83      0.67
              1       0.52      0.32      0.39
              2       0.00      0.00      0.00
              3       0.00      0.00      0.00
              4       0.00      0.00      0.00
              5       0.00      0.00      0.00
              6       0.00      0.00      0.00
              7       0.00      0.00      0.00
              8       0.00      0.00      0.00
```

```
           9       0.00      0.00      0.00

    micro avg       0.55      0.55      0.55
    macro avg       0.11      0.11      0.11
    weighted avg       0.50      0.55      0.50

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

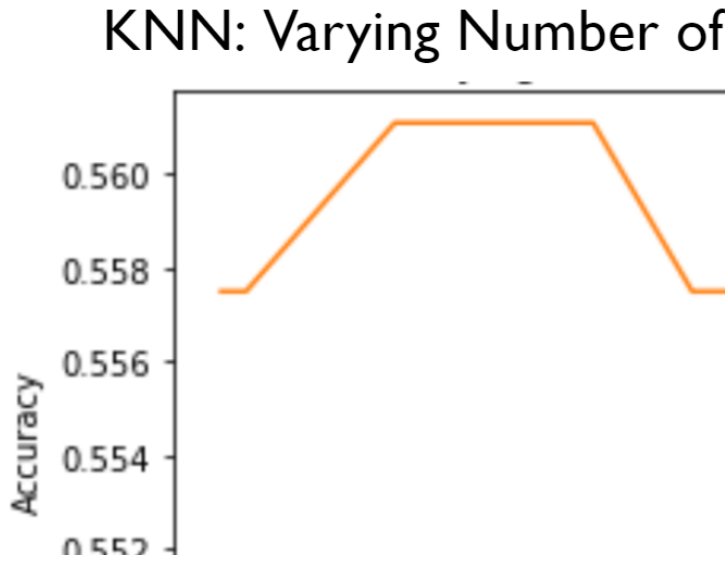
```
#setup arrays to store train and test accuracies fo
n_neighbors_KNN = [5,10,40,80,100,150,200]
train_accuracy = np.empty(len(n_neighbors_KNN))
test_accuracy = np.empty(len(n_neighbors_KNN))

#loop over different values of k
for i, k in enumerate(n_neighbors_KNN):
    #setup a Random forest Classifier with k neighb
    clf_KNN = RandomizedSearchCV(estimator = KNeigh
                                 cv = 10, n_jobs = -1, v

    #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_tr

    #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=16
plt.plot(n_neighbors_KNN, test_accuracy, label = 'T
plt.plot(n_neighbors_KNN, train_accuracy, label =
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_sel
populated class in y has only 5 members, which is too few.
class cannot be less than n_splits=10. % (min_groups, self

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c

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

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 26.2m
Fitting 10 folds for each of 10 candidates, totalling 100 fits
/usr/local/lib/python3.6/dist-packages/sklearn/model_sel
populated class in y has only 5 members, which is too few.
class cannot be less than n_splits=10. % (min_groups, self
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal
[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 10.6min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 24.5m
Fitting 10 folds for each of 10 candidates, totalling 100 fits
/usr/local/lib/python3.6/dist-packages/sklearn/model_sel
populated class in y has only 5 members, which is too few.
class cannot be less than n_splits=10. % (min_groups, self
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 11.7min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 27.9m
Fitting 10 folds for each of 10 candidates, totalling 100 fits
/usr/local/lib/python3.6/dist-packages/sklearn/model_sel
populated class in y has only 5 members, which is too few.
class cannot be less than n_splits=10. % (min_groups, self
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal
[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 8.1min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 27.3m
Fitting 10 folds for each of 10 candidates, totalling 100 fits
/usr/local/lib/python3.6/dist-packages/sklearn/model_sel
populated class in y has only 5 members, which is too few.
class cannot be less than n_splits=10. % (min_groups, self
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c
[Parallel(n_jobs=-1)]: Done 37 tasks | elapsed: 7.7min
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal

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

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

/usr/local/lib/python3.6/dist-packages/sklearn/model_sel
populated class in y has only 5 members, which is too few.
class cannot be less than n_splits=10. % (min_groups, self

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal

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

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

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

/usr/local/lib/python3.6/dist-packages/sklearn/model_sel
populated class in y has only 5 members, which is too few.
class cannot be less than n_splits=10. % (min_groups, self

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 c

/usr/local/lib/python3.6/dist-packages/sklearn/externals/joblib/externals/loky/process
stopped while some jobs were given to the executor. This c
timeout or by a memory leak. "timeout or by a memory leal

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

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



SVM

We also used a multiclass svm classifier. We tested this or
hyperparameters tuned were the penalty parameter C, the
type of decision function (One vs Rest). SVM is best used
in the case of this dataest, it doesn't seem to work a well. T
better on the test set than the 10 CV model, albeit by a frac

CV=3

```
from sklearn.model_selection import RandomizedSearc
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,
            'decision_function_shape': ['ovr']
             }


#instantiating the svm model
svm = SVC()

#GridSearchCV
svm_grid_search3 = RandomizedSearchCV(estimator = s
                                      cv = 3, n_job
```

```
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_
print('Accuracy of SVM_3 Classifier: {:.3f}'.format
```

```
#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,
            'decision_function_shape': ['ovr']
             }

svm_grid_search10 = RandomizedSearchCV(estimator =
                                       cv = 10, n_jo
```

```
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_svn
print('Accuracy of SVM_10 Classifier: {:.3f}'.forma
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
LokyBackend with 2 concurrent workers. [Parallel(n_jobs=-
120.2min finished

Best CV10 params {'C': 1, 'decision_function_shape': 'ovr', 'g

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 plo
each one.

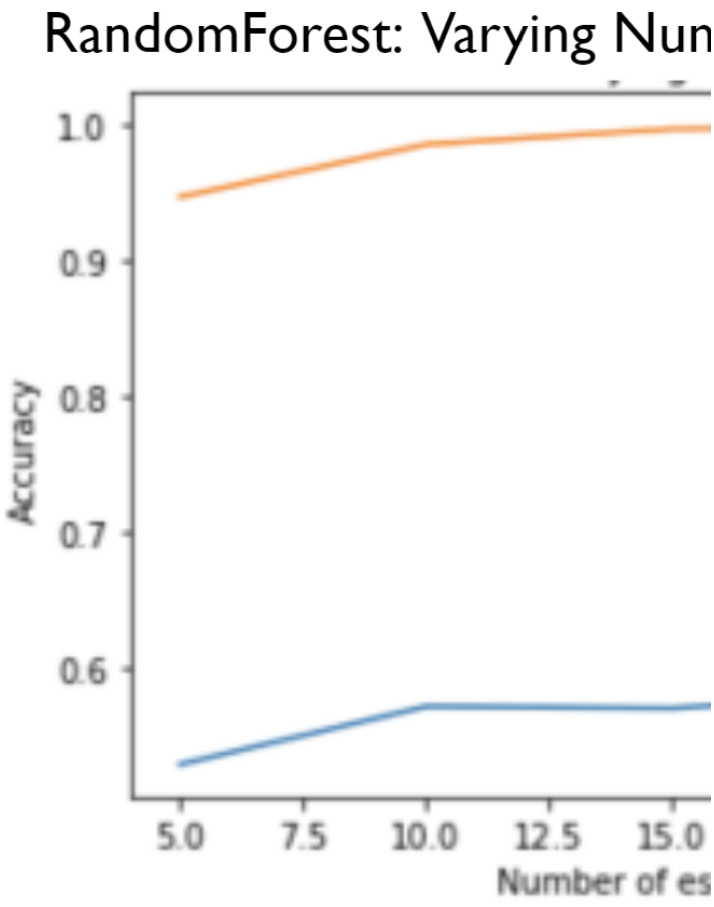
We can see that the highest accuracy occurs at n_estimat
cross validation.

We then run a 3-fold cross validation, getting an average a

We also ran a 10-fold cross validation getting an average s
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 estima
    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 estimat
plt.plot(estimators, test_accuracy, label = 'Testir
plt.plot(estimators, train_accuracy, label = 'Trair
plt.legend()
plt.xlabel('Number of estimators')
plt.ylabel('Accuracy')
plt.show()
```



CV=3

```
#cross validation of rf scores

rf=RandomForestClassifier(n_estimators=25)

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

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

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

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_scor
cv_scores = cross_val_score(rf, X_train, y_train, c

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

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

Average 10-Fold CV Score:0.580

Decision Tree

Here we import and set up a parameter grid for a Decision

For hyper parameter tuning: We run a randomized search C
the best parameters for our decision tree are. We also run
well.

Looking at the accuracy scores we see that using cv10 vs
score by ~0.0016 which is quite an insignificant change sc
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 hyperparam
dt_random3 = RandomizedSearchCV(estimator = dt, par
# Fit the random search model
dt_random3.fit(X_train, y_train)
```

```
#printing best 3 fold cross validation score and pa
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 Sear
dt_best3 = dt_random3.best_estimator_
```

```
#use the random grid to search for best hyperparame
dt_random10 = RandomizedSearchCV(estimator = dt, pa
#uit the random search model
dt_random10.fit(X_train, y_train)
```

CV=10

```
#printing best 10 fold cross validation score and p
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 Sea
dt_best10 = dt_random10.best_estimator_
```

Bagging Classifier

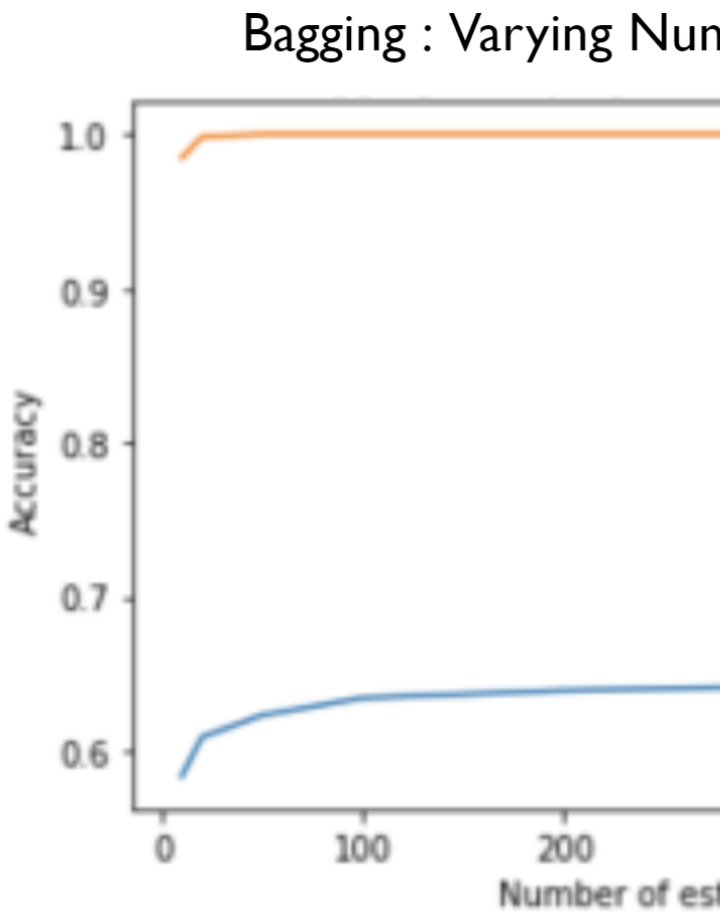
We then fit a bagging classifier using the best decision tree set and computed the accuracy getting an accuracy of 0.6

We also fit a bagging classifier using the best decision tree computed the accuracy getting an accuracy of 0.599.

Again, this is consistent with our earlier decision to go with 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 estima
    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('Baggingt: Varying Number of estimators')
plt.plot(estimators, test_accuracy, label = 'Testir
plt.plot(estimators, train_accuracy, label = 'Trair
plt.legend()
plt.xlabel('Number of estimators')
plt.ylabel('Accuracy')
plt.show()
```



CV=3

```
bc3 = BaggingClassifier(base_estimator=dt_best3, n_
```

```
#fit Bagging Classifier using 3-fold cross validate
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
```

CV=10

```
bc10 = BaggingClassifier(base_estimator=dt_best10,
#fit Bagging Classifier using 10-fold cross validat
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
```

AdaBoost Classifier

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

We choose n_estimators=20 as there doesn't seem to be a number of estimators.

We then fit the ABD classifier using our 3 fold cross validat accuracy which turns out to be 0.513.

We also fit it using our 10 fold cross validated decision tre 0.491, so again, we are better off using our 3 fold cross val 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 estima
    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_tr

    #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 = 'Testir
plt.plot(estimators, train_accuracy, label = 'Trair
plt.legend()
plt.xlabel('Number of estimators')
plt.ylabel('Accuracy')
plt.show()
```

CV=3

```
adb3 = AdaBoostClassifier(base_estimator=dt_best3,
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: {
```

Accuracy of 3 fold CV AdaBoost Classifier: 0.513

CV=10

```
adb10 = AdaBoostClassifier(base_estimator=dt_best1(
adb10.fit(X_train, y_train)
```

```
y_pred_adb10=adb10.predict(X_test)
accuracyadb10 = accuracy_score(y_test, y_pred_adb1(
print('Accuracy of 10 fold CV AdaBoost Classifier:
```

Accuracy of 10 fold CV AdaBoost Classifier: 0.491

Gradient Boosting Classifier

Here we print the parameters used by our current model. W
for hyper parameter tuning.

We then run a RandomizedSearchCV using our parameter
to get the best hyperparameters for our model.

We then print the best parameters and extract the best mo

We bring the accuracy getting a training accuracy of … and

We did not run 10-fold CV randomized search as it took wa

```
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 no
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_spl
               'min_samples_leaf': min_samples_leaf
pprint(random_grid1)
```

```
#use the random grid to search for best hyperparame
#first create the base model to tune
clf = GradientBoostingClassifier()
#random search of parameters, using 3 fold cross va
#search across 100 different combinations, and use
clf_random = RandomizedSearchCV(estimator = clf, pa
#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 = {:0.2f}%.
print('Training Accuracy with 3 fold CV = {:0.2f}%.
```

MLP

We decided to try and use Multilayered Perceptrons on the
is an artificial neural network that tends to perform well on
data. It also performs well as a base neural network mode
and cv10 perform exceptionally well (~99%), but when lool
we noticed that cv3 seems to be doing better on cards witl

CV = 3

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

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

mlp = MLPClassifier()

mlp_cv3 = RandomizedSearchCV(estimator = mlp, parar

mlp_cv3.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits [P
LokyBackend with 2 concurrent workers. [Parallel(n_jobs=-
finished /usr/local/lib/python3.6/dist-
packages/sklearn/neural_network/multilayer_perceptron.p
Optimizer: Maximum iterations (200) reached and the optii
self.max_iter, ConvergenceWarning) RandomizedSearchC\
estimator=MLPClassifier(activation='relu', alpha=0.0001, b
beta_2=0.999, early_stopping=False, epsilon=1e-08, hidder
learning_rate='constant', learning_rate_init=0.001, max_iter
n_iter_no_change=10, nesterovs_momentum=True, power_
solver='adam', tol=0.0001, validation_fraction=0.1, verbose
fit_params=None, iid='warn', n_iter=10, n_jobs=-1, param_d
50), (50, 100, 50), (100,)], 'activation': ['tanh', 'relu'], 'solver': [
'learning_rate': ['constant', 'adaptive']}, pre_dispatch='2*n_jo
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_
```

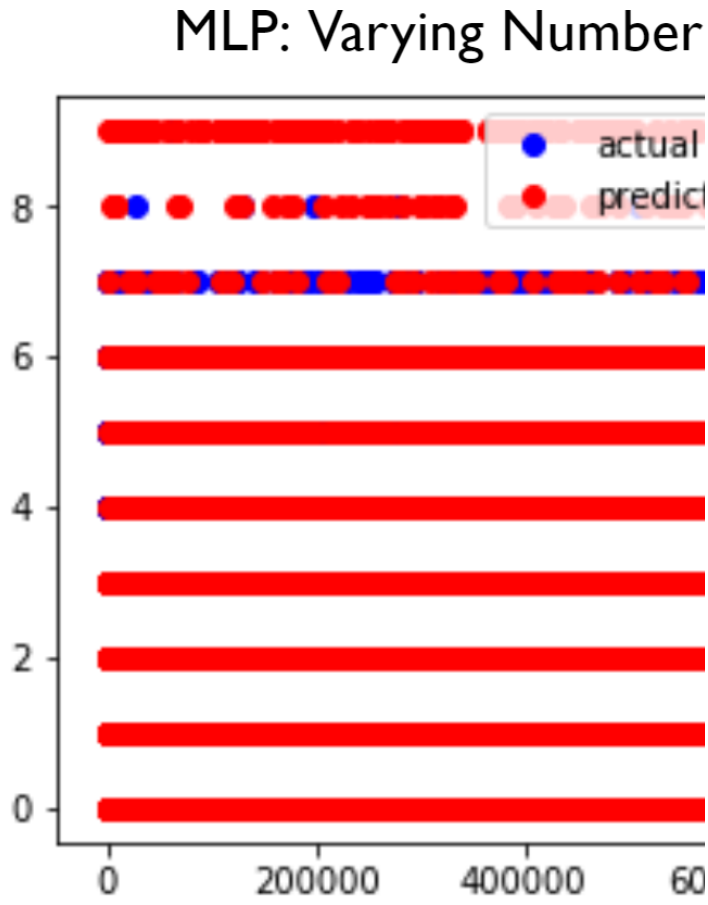
```
0.9534586165533786
```

```
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()
```



CV = 10

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

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

mlp = MLPClassifier()

mlp_cv10 = RandomizedSearchCV(estimator = mlp, para

mlp_cv10.fit(X_train, y_train)
```

Fitting 10 folds for each of 10 candidates, totalling 100 fits
packages/sklearn/model_selection/_split.py:652: Warning
members, which is too few. The minimum number of mem
n_splits=10. % (min_groups, self.n_splits)), Warning) [Paral
LokyBackend with 2 concurrent workers. [Parallel(n_jobs=-
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 212.2r
packages/sklearn/neural_network/multilayer_perceptron.p

```
Optimizer: Maximum iterations (200) reached and the optii
self.max_iter, ConvergenceWarning) RandomizedSearchC\
estimator=MLPClassifier(activation='relu', alpha=0.0001, b
beta_2=0.999, early_stopping=False, epsilon=1e-08, hidder
learning_rate='constant', learning_rate_init=0.001, max_iter
n_iter_no_change=10, nesterovs_momentum=True, power_
solver='adam', tol=0.0001, validation_fraction=0.1, verbose
fit_params=None, iid='warn', n_iter=10, n_jobs=-1, param_d
50), (50, 100, 50), (100,)], 'activation': ['tanh', 'relu'], 'solver': [
'learning_rate': ['constant', 'adaptive']}, pre_dispatch='2*n_jc
return_train_score='warn', scoring=None, verbose=2)
```

```
mlp_cv10.best_params_
```

```
{'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 1(
'adam'}
```

```
mlp_cv10.best_score_
```

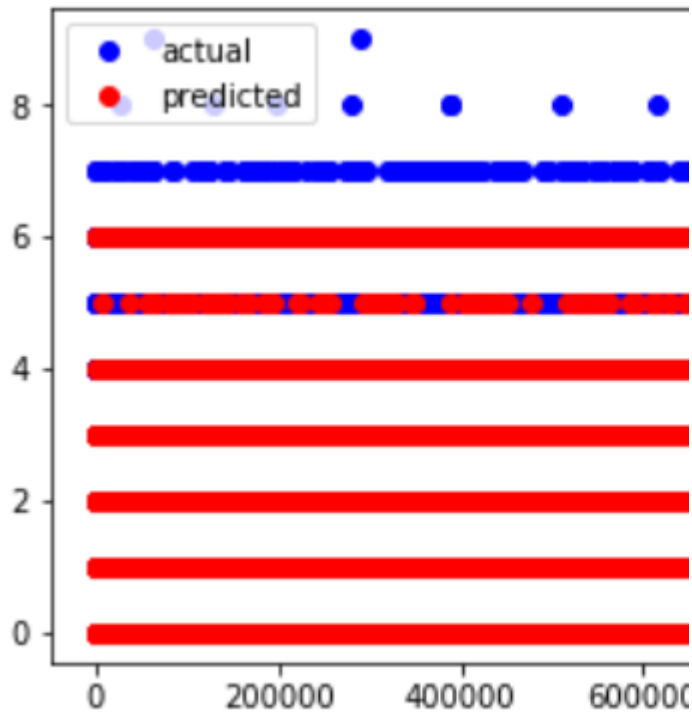
```
0.9935225909636145
```

```
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()
```


actual
predicted
8
6
4
2
0
0
200000
400000
600000