PROJECT REPORT

Name: Bala Chandrashekar  
Roll Number: 40  
Reg No: 12002137  
Topic: POKER HANDS

Abstract:

The purpose of this analysis is to look into the Poker Hand Data set and analyze the strength of each hand as a classification exercise. This dataset has already been divided 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 ve 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:

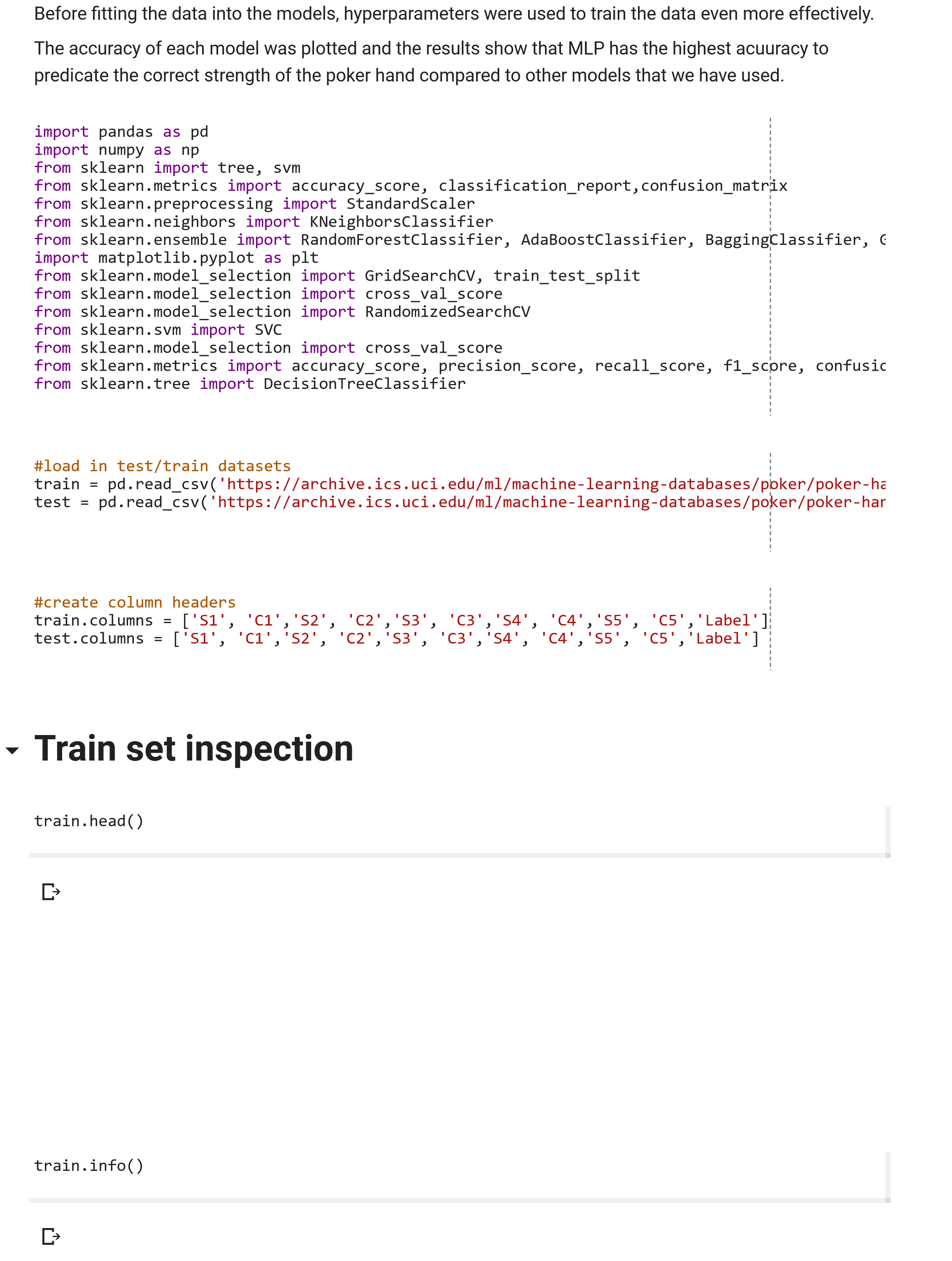
1. S1 "Suit of card #1" Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
2. C1 "Rank of card #1" Numerical (1-13) representing (Ace, 2, 3, ... , Queen, King)
3. S2 "Suit of card #2" Ordinal (1-4) representing {Hearts, Spades, Diamonds, Clubs}
4. C2 "Rank of card #2" Numerical (1-13) representing (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 ve cards 2: Two pairs; two pairs of equal ranks within ve cards 3: Three of a kind; three equal ranks within ve cards 4: Straight; ve cards, sequentially ranked with no gaps 5: Flush; ve cards with the same suit 6: Full house; pair + different rank three of a kind 7: Four of a kind; four equal ranks within ve cards 8: Straight ush; straight + ush 9: Royal ush; {Ace, King, Queen, Jack, Ten} + ush"

We rst 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 ve 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 classi cation technique on our dataset. We have used models such KNN, SVM, RandomForest, Adaboost, Gradient Boosting and Bagging to analyze the dataset, nd their accuracy and predict the labels.



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

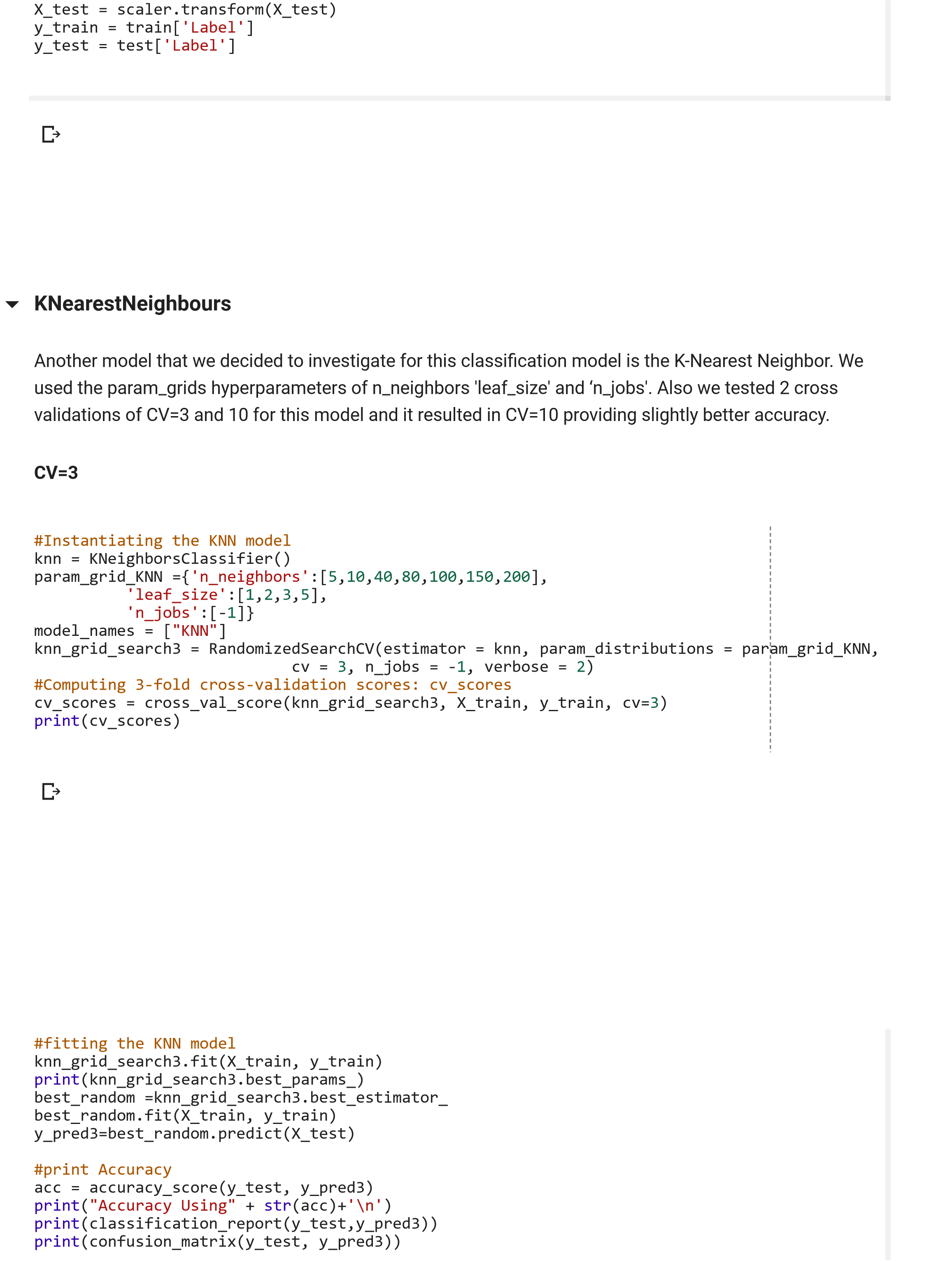
.

transform

(

X\_train

)



Fitting 3 folds for each of 10 candidates, totalling 30 ts [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 nished

Accuracy Using0.547687

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classi cation.py:1143: Unde nedMetricWarning:

Precision and F-score are ill-de ned and being set to 0.0 in labels with no predicted samples. 'precision',

'predicted', average, warn\_for) precision recall f1-score support

1. 0.56 0.81 0.66 501209
2. 0.51 0.33 0.40 422498
3. 0.00 0.00 0.00 47622
4. 0.00 0.00 0.00 21121
5. 0.00 0.00 0.00 3885
6. 0.00 0.00 0.00 1996
7. 0.00 0.00 0.00 1424
8. 0.00 0.00 0.00 230
9. 0.00 0.00 0.00 12
10. 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 ts

[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 nished

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

[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 nished

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

[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 nished

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

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

[Parallel(n\_jobs=-1)]: Done 30 out of 30 | elapsed: 8.0min nished

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

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

[Parallel(n\_jobs=-1)]: Done 30 out of 30 | elapsed: 7.0min nished

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

[

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

out of

30

| elapsed:

30

min nished

8.2

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

[

Parallel(n\_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.

[

Parallel(n\_jobs=-1)]: Done

30

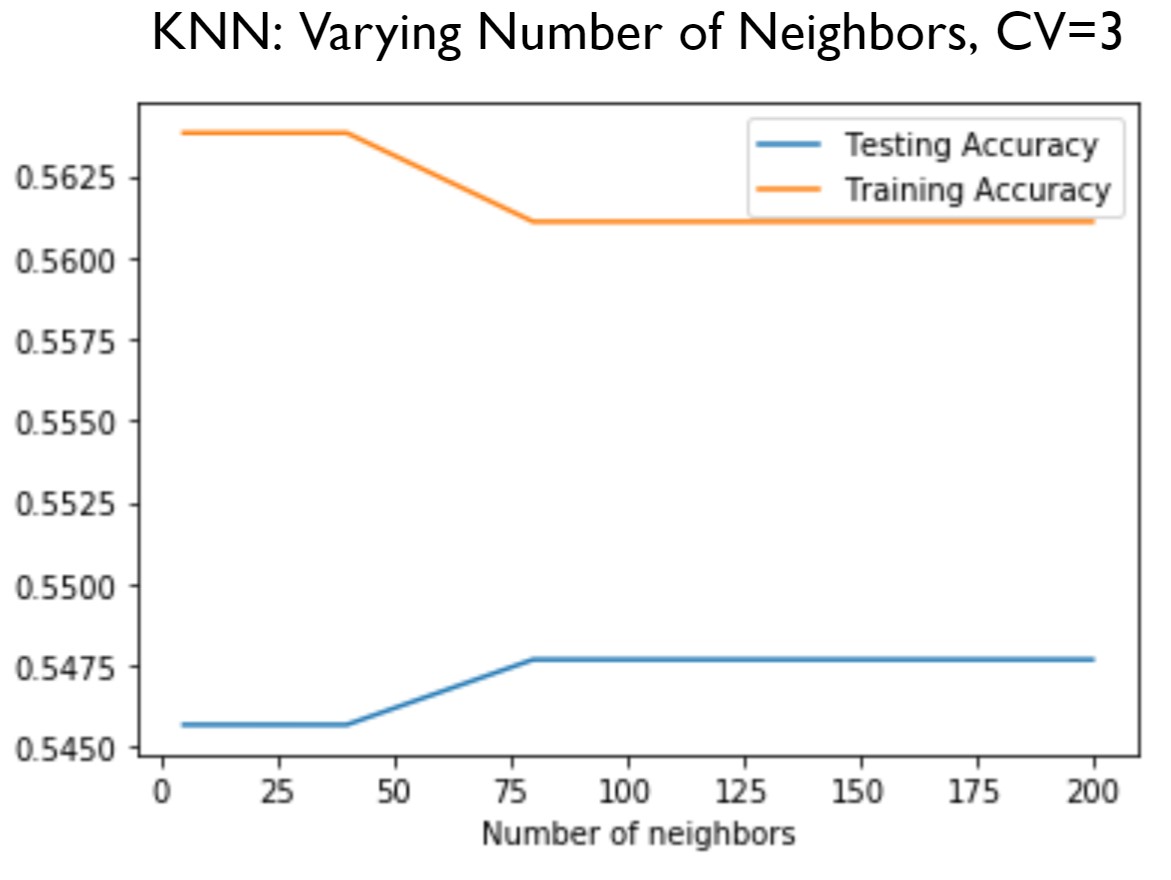
out of

| elapsed:

30

min nished

8.1



**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 ts

/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 nished

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

/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 nished

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

/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 nished [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 ts

/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 nished

Accuracy Using0.54831

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/classi cation.py:1143: Unde nedMetricWarning:

Precision and F-score are ill-de ned and being set to 0.0 in labels with no predicted samples. 'precision',

'predicted', average, warn\_for) precision recall f1-score support

1. 0.56 0.83 0.67 501209
2. 0.52 0.32 0.39 422498
3. 0.00 0.00 0.00 47622
4. 0.00 0.00 0.00 21121
5. 0.00 0.00 0.00 3885
6. 0.00 0.00 0.00 1996
7. 0.00 0.00 0.00 1424
8. 0.00 0.00 0.00 230
9. 0.00 0.00 0.00 12
10. 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 ts

/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 nished

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

/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 nished

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

/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 nished

Fitting 10 folds for each of 10 candidates, totalling 100 ts /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 nished

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

/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 nished

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

/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 nished

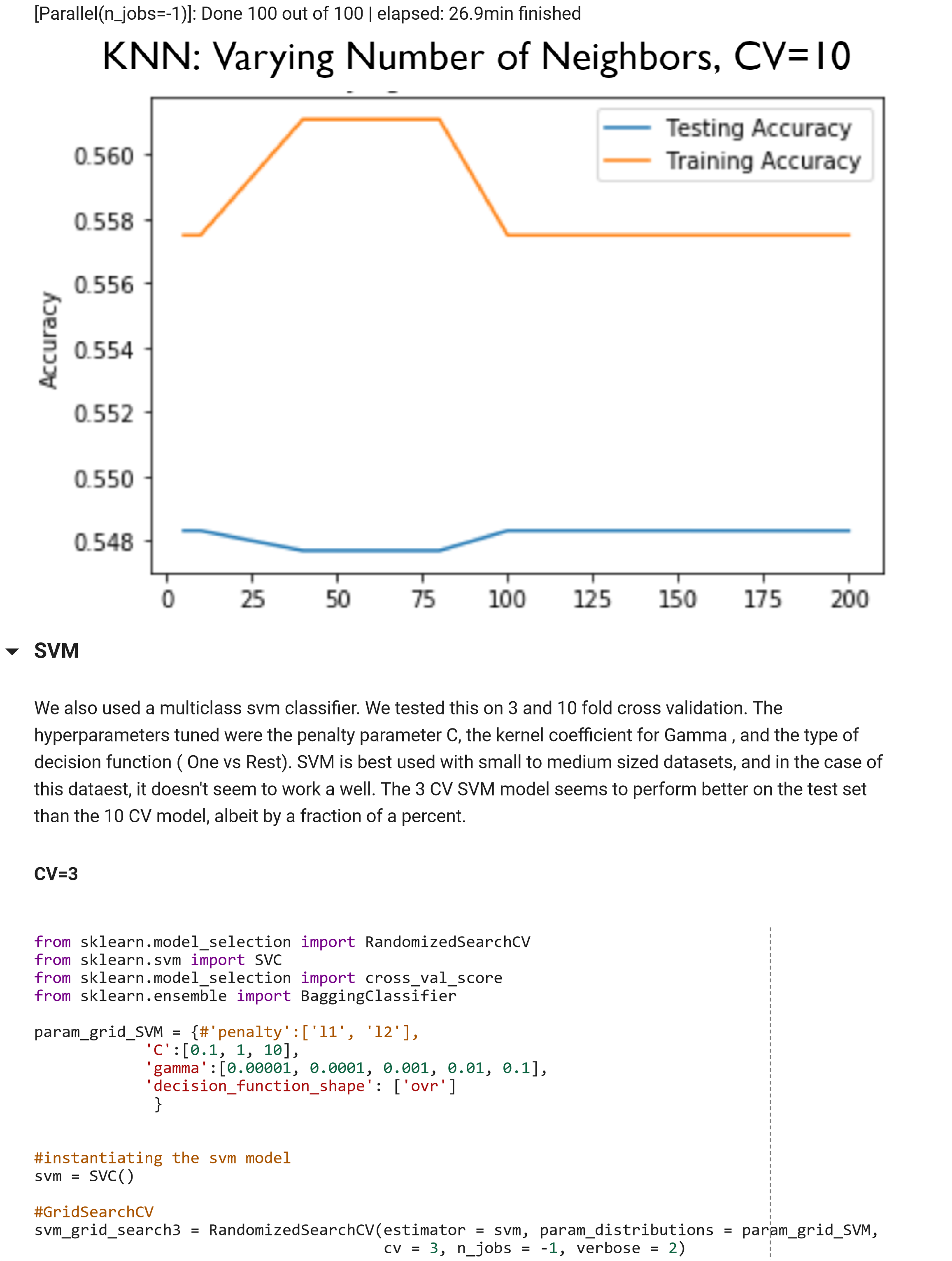
Fitting 10 folds for each of 10 candidates, totalling 100 ts

/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



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 ts [Parallel(n\_jobs=-1)]: Using backend

LokyBackend with 2 concurrent workers. [Parallel(n\_jobs=-1)]: Done

100

out of

100

| elapsed: 120.2min

nished

Best CV10 params {'C': 1, 'decision\_function\_shape': 'ovr', 'gamma': 0.1}

Best CV10 accuracy 0.5537194322271092

'Accuracy of SVM\_10 Classi er: 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'

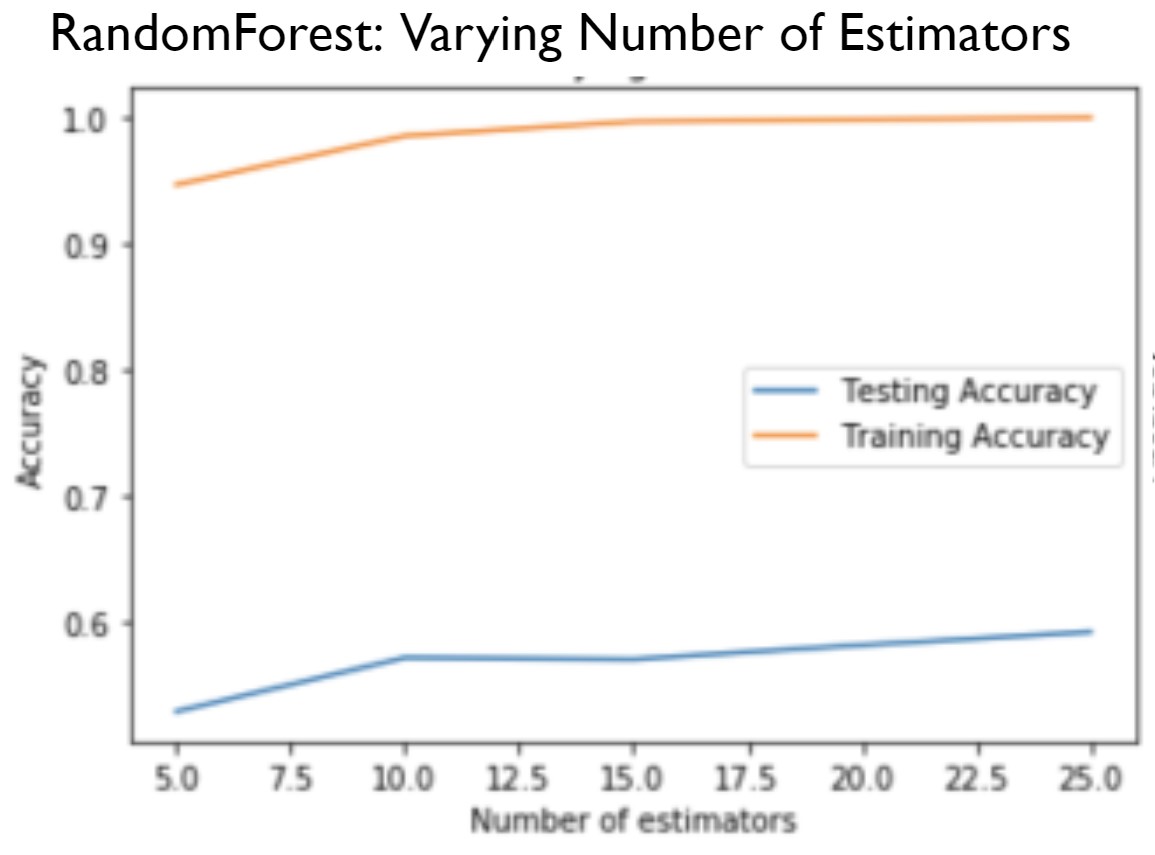
)

plt

.

show

()



**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



#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 t a bagging classifier using the best decision tree model form cv3, predicted on the test set and

computed the accuracy getting an accuracy of 0.609

We also t a bagging Classifier using the best decision tree from cv10, predicted on the rest 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

(

'Baggingt: 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'

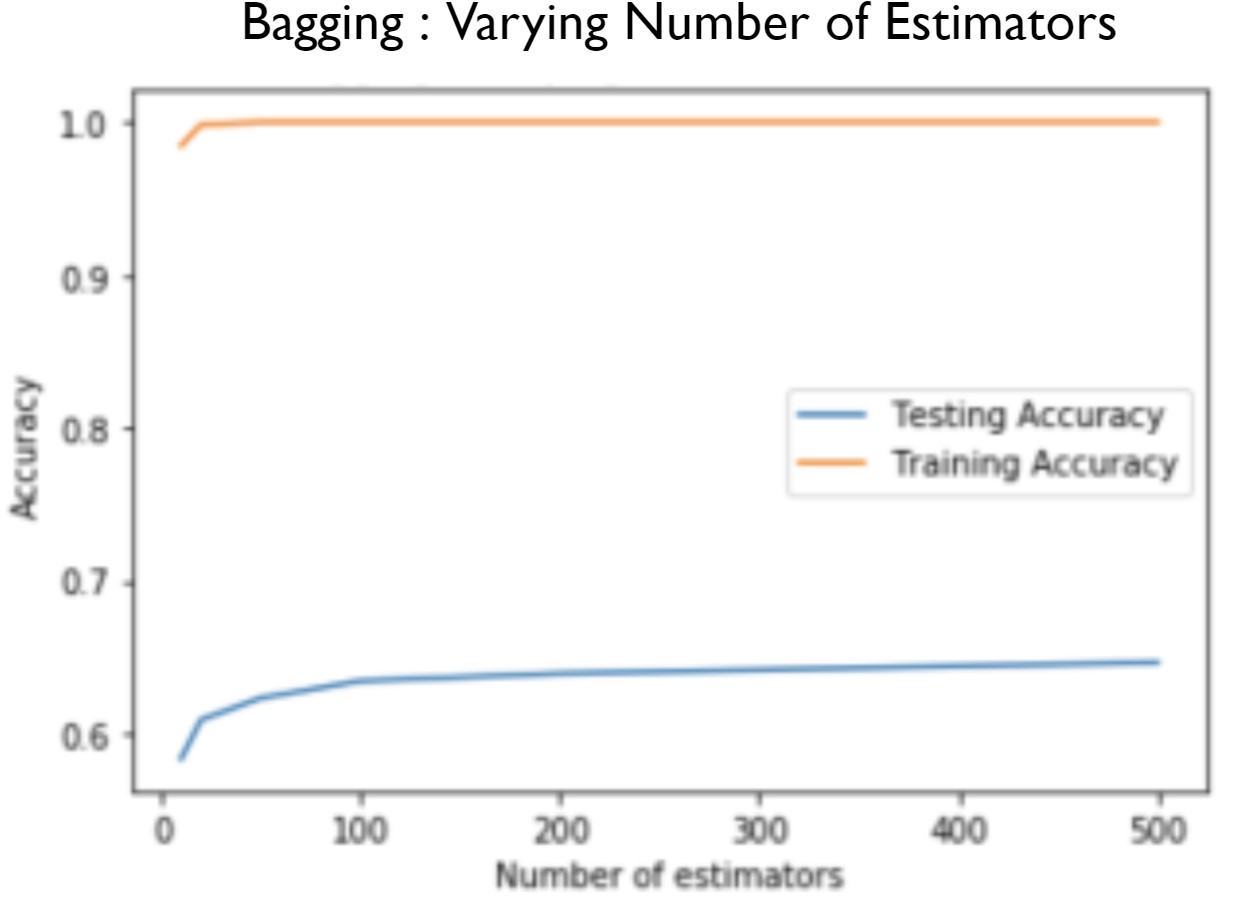
)

plt

.

show

()





**AdaBoost Classifier**

Here we run a for loop to iter through a list of n\_estimators and compute the accuracy of the adaboost

classi er 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 t the ABD classi er using our 3 fold cross validated decision tree and compute the accuracy

which turns out to be 0.513.

We also t it using our 10 fold cross validated decision tree and compute the accuracy to nd 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'

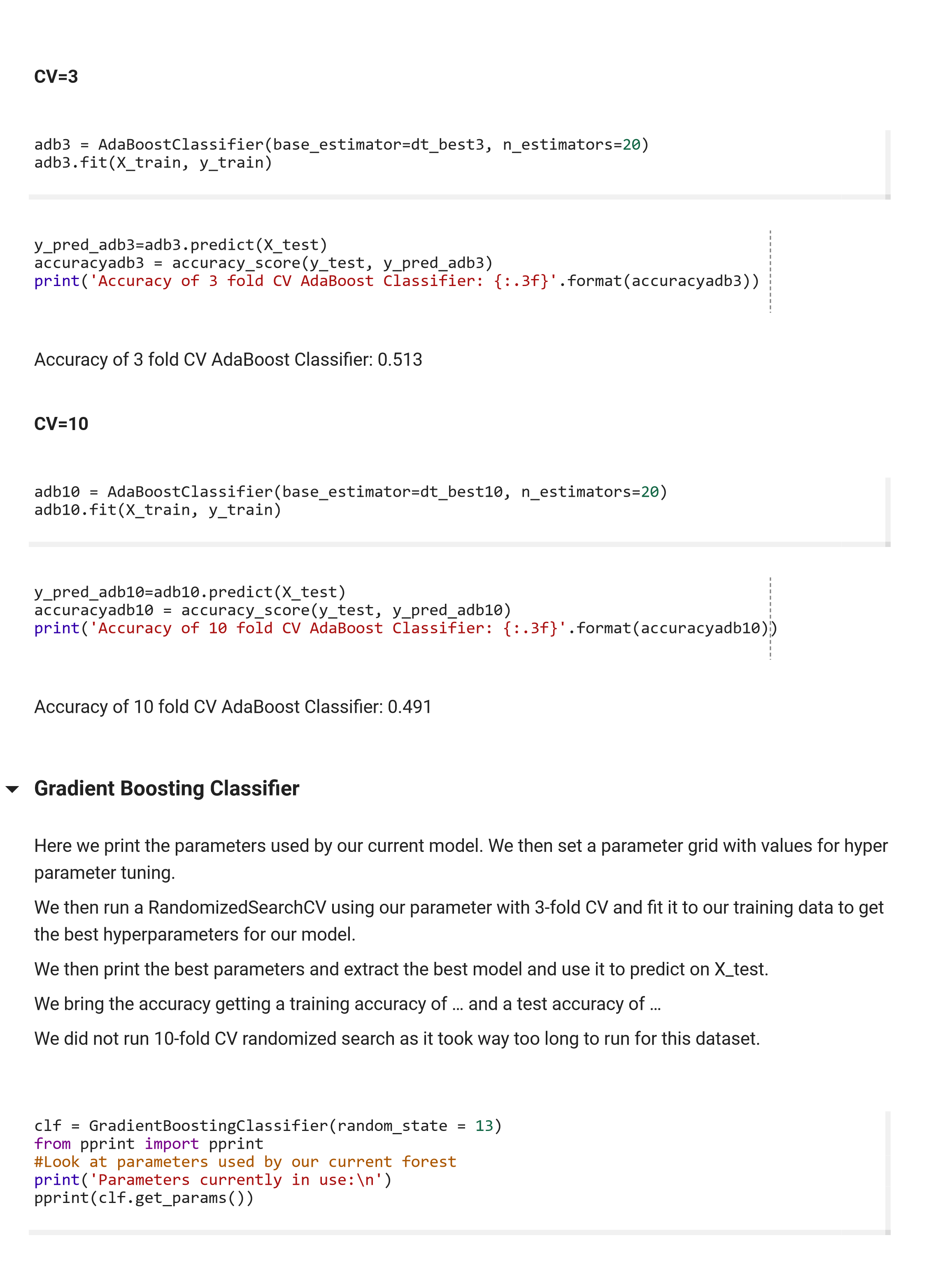
)

plt

.

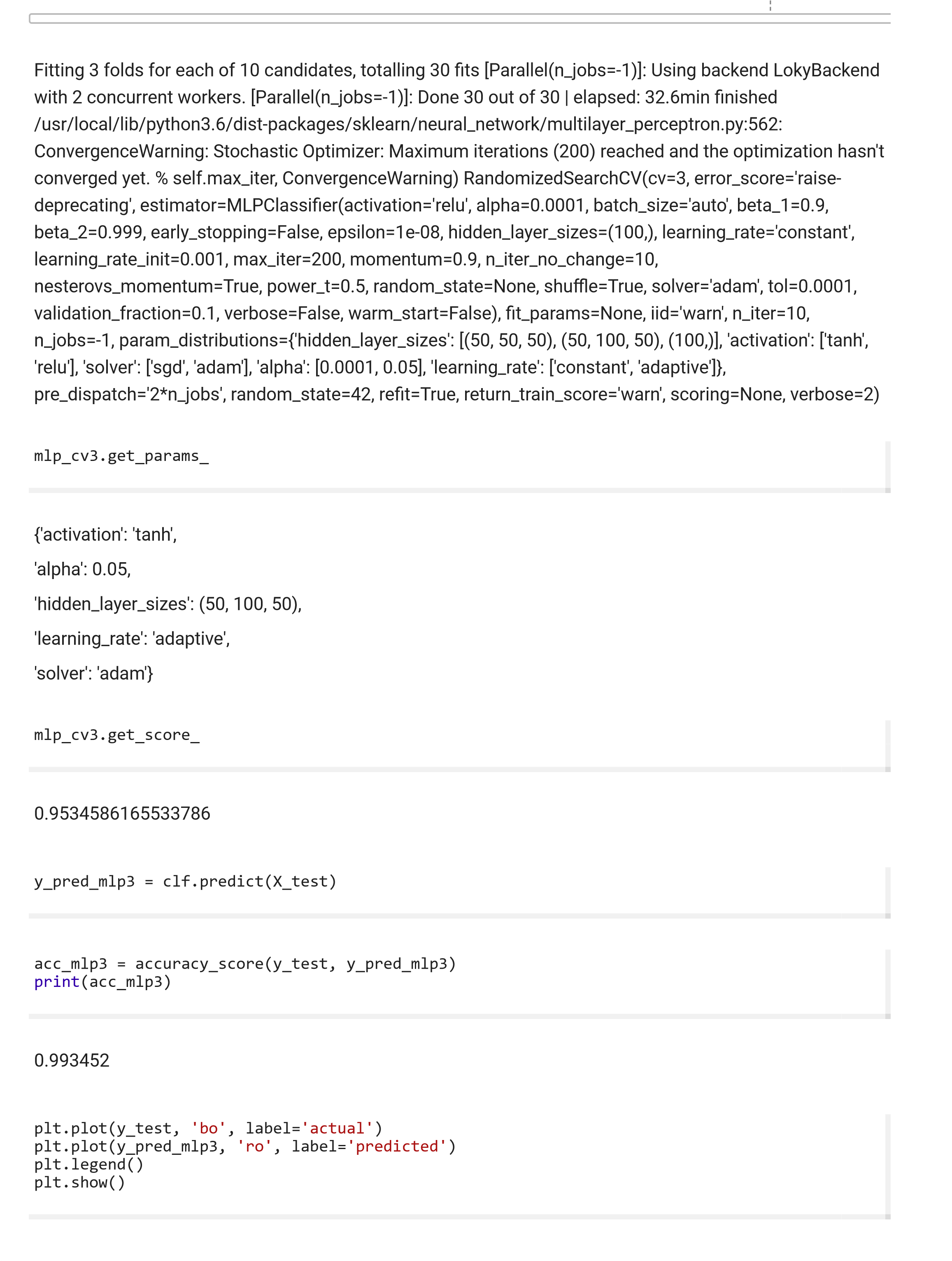
show

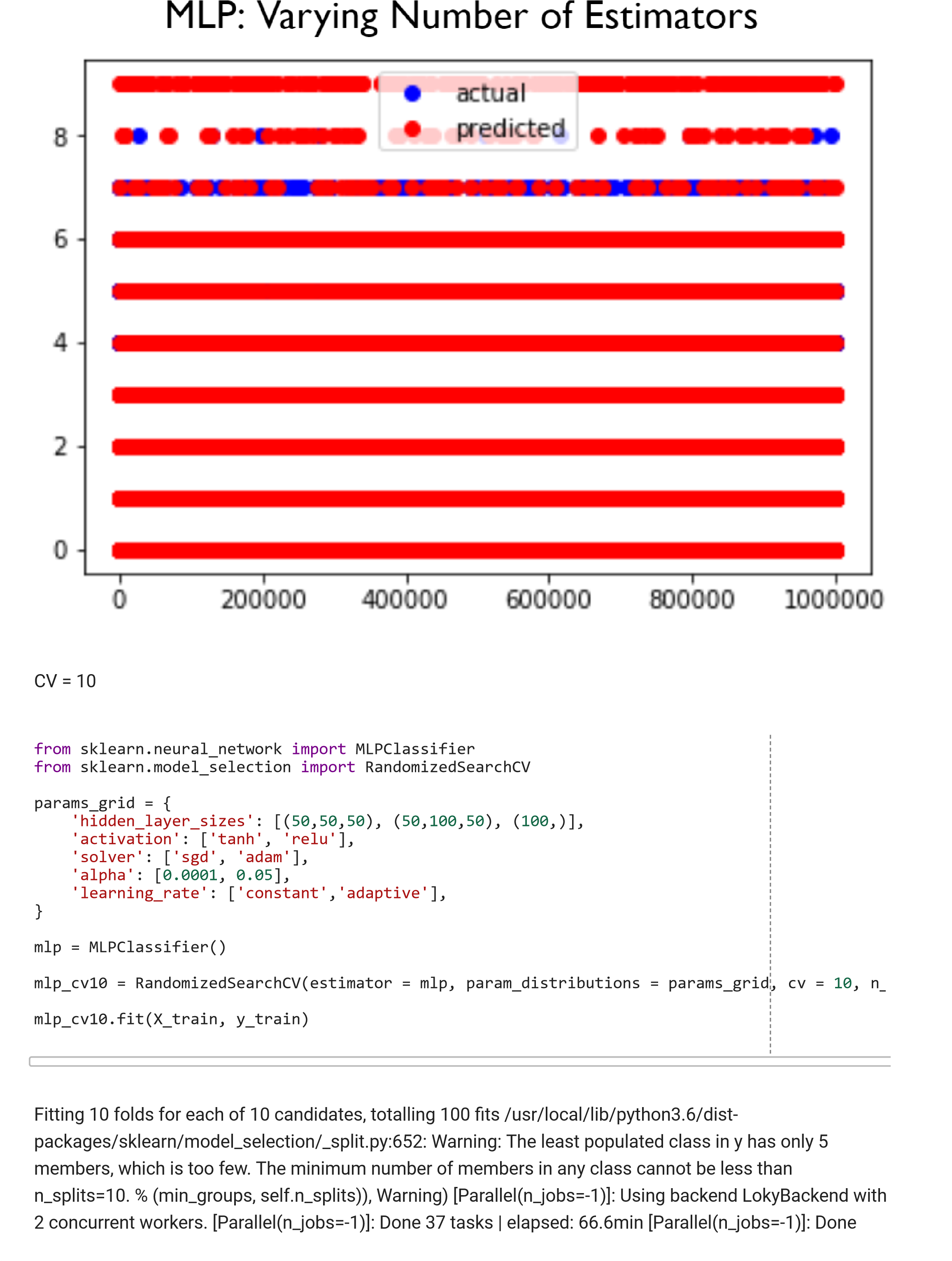
()











100

out of 100 | elapsed: 212.2min nished /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=MLPClassi er(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,

e=True, solver='adam', tol=0.0001,

random\_state=None, shu

validation\_fraction=0.1, verbose=False, warm\_start=False),

t\_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, re t=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

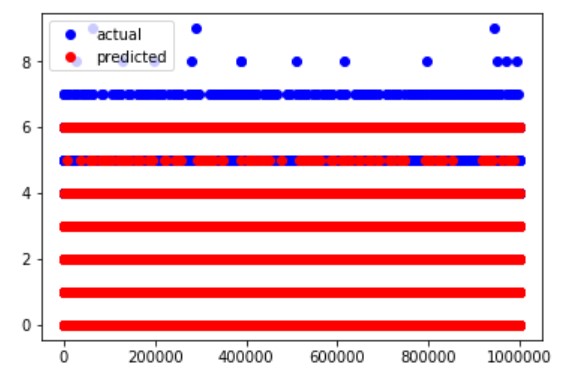
()

plt

.

show

()



Reference:

GitHub Link: <https://github.com/Chandrashekar-141/Poker-Hand.git>

Data Set:

<https://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker-hand-training-true.data>

https://archive.ics.uci.edu/ml/machine-learning-databases/poker/poker-hand-testing.data