Assignment-2

February 13, 2018

Assignment 2
Name: Hritik Soni
ID: 2014A2PS480P
Email: f2014480@pilani.bits-pilani.ac.in
Dataset Information:
45000 Rows 62 Cols
Type: csv
Target Type: Nominal (0 or 1)
Since the target is nominal, we cannot use linear regression.
We will attempt to use the following classifiers:

1. KNN - We will use k = sqrt(# of instances) as it is expected to give decent results.
2. D-Trees
3. Random Forest
4. Naive Bayes
5. MLP

First step is to import all required libraries.

In [174]:
```
print("Importing Libraries...")

import math
import numpy as np
import pandas as pd
#import matplotlib.pyplot as plt
#%matplotlib inline
import seaborn as sns

from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

```
#from sklearn.svm import SVC
#SVC was taking too much time
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
```

Let us set some globals related to our dataset.

```
In [175]: dims = 61 #Known number of attributes
          instances = 45000 #number of datapoints

          classifiers = ["knn", "dtrees", "rforest", "nb", "mlp"]#This classifiers in this lis
          #Classfier can be one of these - knn, dtrees, rforest, nb, mlp

          k = int(math.sqrt(instances))#Sets k for knn classifier

          numiter = 3 #Number of iterations per classifier for determining score
```

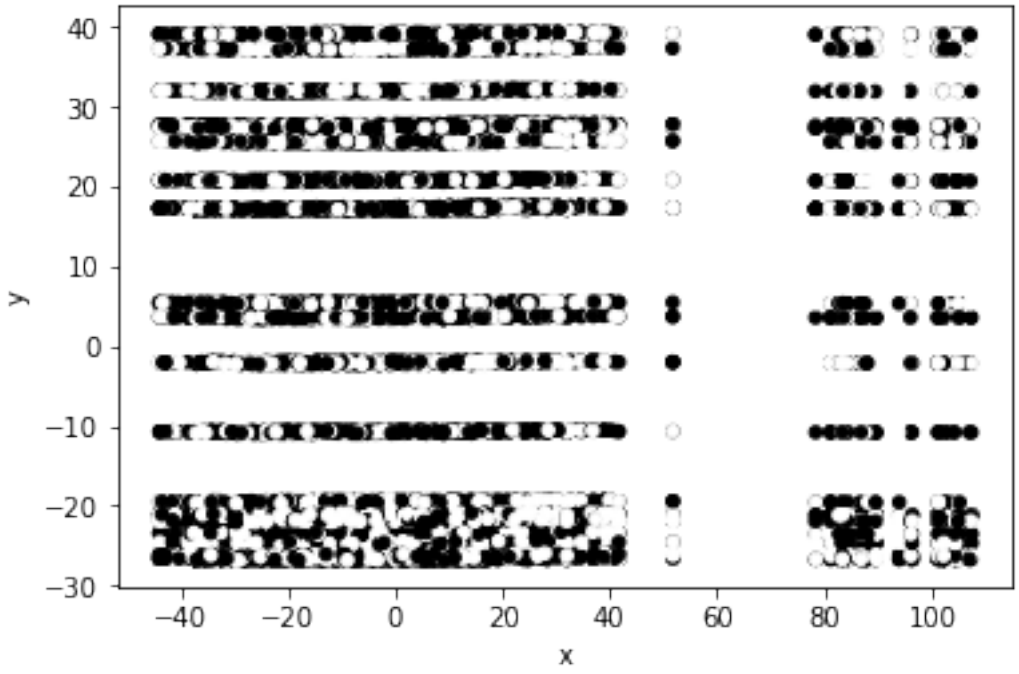
```
In [176]: print("Loading Dataset")
          dataset = np.loadtxt("dataA2.csv", delimiter=",").astype(int)
          X, y = dataset[:, 0:61], dataset[:, 61]
```

Let us try to visualize the data in two dimensions and see if it is linearly separable or some kind of observable pattern is present.

```
In [177]: dim2 = PCA(n_components=2)
          dim2.fit(X)
          X3 = dim2.fit_transform(X)
```

```
In [178]: dfplot = pd.DataFrame(X3, columns=['x', 'y'])
          dfplot.plot.scatter('x', 'y', c=y)
```

```
Out[178]: <matplotlib.axes._subplots.AxesSubplot at 0x7f35d1523ef0>
```



From above plot, we can see that the data can be discretized well especially in the second dimension and thus decision trees or forests can be expected to perform well.

```
In [179]: print("Analyzing Data Using PCA ...")
          for i in range(1,dims):
             pca = PCA(n_components=i)
             pca.fit(X)
             print (i, "\t:", sum(pca.explained_variance_ratio_))
```

```
1          : 0.25714755525572
2          : 0.4522276793648473
3          : 0.6206242981581517
4          : 0.6953250652008685
5          : 0.7402942364598554
6          : 0.7847761759035773
7          : 0.8190448369914763
8          : 0.8470013968504166
9          : 0.863087538766515
10          : 0.8749051036169754
11          : 0.8846718897003077
12          : 0.8943586856662359
13          : 0.9027171843412429
14          : 0.9106009793779547
15          : 0.9182040602971723
16          : 0.9252439827406288
17          : 0.9314286409319192
```

```
18          : 0.9371495260493802
19          : 0.9423471275037831
20          : 0.9473244453523865
21          : 0.9520846591826406
22          : 0.9565353142450106
23          : 0.9607401614934672
24          : 0.9645358705575101
25          : 0.9681572720326954
26          : 0.9715551608689391
27          : 0.9748765966816224
28          : 0.9780403194821139
29          : 0.9806972888966614
30          : 0.9832167116402207
31          : 0.9856184119518779
32          : 0.9876790347969997
33          : 0.9896007646073317
34          : 0.9910237047429602
35          : 0.9923599912762661
36          : 0.9934682572766139
37          : 0.9945312315513741
38          : 0.9955510428065616
39          : 0.9964582267450189
40          : 0.9971414572965128
41          : 0.9978232046736494
42          : 0.9984269302844183
43          : 0.9989568999844016
44          : 0.9993485065986566
45          : 0.9995679469239718
46          : 0.9997465474053548
47          : 0.9998668893866752
48          : 0.9999401251531697
49          : 0.9999674363699455
50          : 0.9999907702705862
51          : 0.999995886566568
52          : 1.0
53          : 1.0
54          : 1.0
55          : 1.0
56          : 1.0
57          : 1.0
58          : 1.0
59          : 1.0
60          : 1.0
```

From above decomposition, it is easy to see that we capture just over 99% data by considering 34 dimensions. Therefore, for speed reasons and avoiding overfitting we can consider 34 dimensions instead of 61.

```
In [180]: print("Reducing data dimensions to 34")
          pca = PCA(n_components=34)
          pca.fit(X)
          X2 = pca.fit_transform(X)
```

Now, we have to split the data into testing and training data. We will use 10% of the data for testing.

```
In [181]: results = {}

          for c in classifiers:
              print("Training using classifier:", c)
              results[c] = []
              for itr in range(numiter):
                  X_train, X_test, y_train, y_test = train_test_split(X2, y, test_size=0.1)
                  if c == "knn":
                      classifier = KNeighborsClassifier(n_neighbors=k)
                  elif c == "dtrees":
                      classifier = DecisionTreeClassifier()
                  elif c == "nb":
                      classifier = GaussianNB()
                  elif c == "rforest":
                      classifier = RandomForestClassifier(n_estimators=45, max_depth = 20)
                  elif c == "mlp":
                      scaler = StandardScaler()
                      scaler.fit(X_train)
                      X_train = scaler.transform(X_train)
                      X_test = scaler.transform(X_test)
                      classifier = MLPClassifier(hidden_layer_sizes=(34, 34, 34),max_iter=100)
                  classifier.fit(X_train, y_train)
                  results[c].append(classifier.score(X_test, y_test))
                  print("Done")
          #         print("Training Score: ", classifier.score(X_train, y_train))
          #         print("Testing Score: ", results[c][-1])
```

```
/phoenix/.local/lib/python3.6/site-packages/sklearn/neural_network/multilayer_perceptron.py:564
  % self.max_iter, ConvergenceWarning)
```

The code below will print results.

```
In [182]: print("Here are the results:")
          print()

          maxscore = -math.inf
          chosenc = None

          for c in classifiers:
```

```
            print("Using Classifier:", c)
            print()
            for itr in range(numiter):
                print("Execution %s Score:"%(itr), results[c][itr])
            thisavg = sum(results[c])/numiter
            if thisavg > maxscore:
                maxscore = thisavg
                chosenc = c
            print("Execution Average Score:", thisavg)
            print()

        print("The maximum score was obtained by classifer: %s"%(chosenc))
        print("The score was", maxscore)
        print("There we should pick classifier: %s"%(chosenc), "as our base model.")
```

```
Here are the results:

Using Classifier: knn

Execution 0 Score: 0.5713333333333334
Execution 1 Score: 0.564
Execution 2 Score: 0.5713333333333334
Execution Average Score: 0.5688888888888889

Using Classifier: dtrees

Execution 0 Score: 0.5917777777777777
Execution 1 Score: 0.6073333333333333
Execution 2 Score: 0.6084444444444445
Execution Average Score: 0.6025185185185186

Using Classifier: rforest

Execution 0 Score: 0.6313333333333333
Execution 1 Score: 0.6293333333333333
Execution 2 Score: 0.64
Execution Average Score: 0.6335555555555555

Using Classifier: nb

Execution 0 Score: 0.5486666666666666
Execution 1 Score: 0.5688888888888889
Execution 2 Score: 0.5597777777777778
Execution Average Score: 0.5591111111111111

Using Classifier: mlp

Execution 0 Score: 0.5637777777777778
```

```
Execution 1 Score: 0.5808888888888889
Execution 2 Score: 0.552
Execution Average Score: 0.5655555555555556

The maximum score was obtained by classifer: rforest
The score was 0.6335555555555555
There we should pick classifier: rforest as our base model.
```

Considering Random Forests as our base model, we can improve the accuracies by fiddling with tree depth, number of trees, etc. The following code rigorously checks validation accuracy for various depths and tree counts.

```
In [183]: PERFORM_1 = False #Set this to True to perform all the tests

          cycles = 3

          max_train = -math.inf
          maxtr_n = None
          maxtr_d = None

          max_test = -math.inf
          maxte_n = None
          maxte_d = None

          for n_trees in [1, 43, 44, 45, 46, 47, 100]:
              if not PERFORM_1:
                  break
              for depth in [10, 18, 19, 20, 21, 22, 30]:
                  print("Trees %s Depth %s"%(n_trees, depth))
                  print()
                  X_train, X_test, y_train, y_test = train_test_split(X2, y, test_size=0.1)
                  this_tescoreavg = 0
                  this_trscoreavg = 0
                  for c_i in range(cycles):
                      classifier = RandomForestClassifier(n_estimators=n_trees, max_depth = dep
                      classifier.fit(X_train, y_train)
                      thistescore = classifier.score(X_test, y_test)
                      thistrscore = classifier.score(X_train, y_train)
                      this_tescoreavg += thistescore
                      this_trscoreavg += thistrscore
                      print("Cycle %s Train %s Test %s"%(c_i, thistrscore, thistescore))
                  this_tescoreavg /= cycles
                  this_trscoreavg /= cycles
                  if this_tescoreavg > max_test:
                      maxte_n = n_trees
                      maxte_d = depth
                      max_test = this_tescoreavg
```

```
                if this_trscoreavg > max_train:
                    maxtr_n = n_trees
                    maxtr_d = depth
                    max_train = this_trscoreavg
                print("Average Train Score: %s"%(this_trscoreavg))
                print("Average Test Score: %s"%(this_tescoreavg))
                print()

        if PERFORM_1:
            print()
            print("Best Train Score: %s Trees %s Depth %s"%(max_train, maxtr_n, maxtr_d))
            print("Best Test Score: %s Trees %s Depth %s"%(max_test, maxte_n, maxte_d))
```

Over several fold execution of the above code the best validation score was obtained at Tree Depth = 20 and Trees = 45.

Result Summary

Classifier: Random Forest Classifier (inside sklearn.ensemble) Additional Tuning: max_depth = 20 num_estimators = 46

Accuracy Details:

Cycle 0 Train 0.8164197530864198 Test 0.6433333333333333

Cycle 1 Train 0.8147654320987654 Test 0.642

Cycle 2 Train 0.816 Test 0.646

Average Train Score: 0.8157283950617283

Average Test Score: 0.6437777777777778