SEP 786 – Artificial Intelligence and Machine Learning Fundamentals

Project Report

Author:

Soham Tushar Bodas – 400376265

Kushang Patel – 400425820

Dataset:

The dataset used for this project is "Wireless Indoor Localization Data", which is sourced at UCI ML (https://archive.ics.uci.edu/ml/datasets/Wireless+Indoor+Localization#). The data set is gathered by observing the WiFi signal strengths in an indoor location. It is collected by observing the strengths visible on a smart phone. The dataset contains in total 7 attributes and a single target variable. Each attribute is WiFi signal strength observed on a smartphone. The target value is one of the four rooms. The room is predicted based on the 7 signal strengths.

In total there are 2000 data points. 1500 will be used to train and 500 will be tested.

Data Set Characteristics:	Multivariate	**Number of Instances:**	2000	**Area:**	Computer
Attribute Characteristics:	Real	**Number of Attributes:**	7	**Date Donated**	2017-12-04
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	64421

Dataset Description (https://archive.ics.uci.edu/ml/datasets/Wireless+Indoor+Localization#)

```
import numpy as np
import pandas as pd
import time
```

```
cols = ['A', "B", "C", "D", "E", "F", "G", "Room"]
data = pd.read_csv("/content/drive/MyDrive/mcmaster/wifi.csv", names =
 cols, index_col = False)
data.head()
```

	A	B	C	D	E	F	G	Room
0	-64	-56	-61	-66	-71	-82	-81	1
1	-68	-57	-61	-65	-71	-85	-85	1
2	-63	-60	-60	-67	-76	-85	-84	1
3	-61	-60	-68	-62	-77	-90	-80	1
4	-63	-65	-60	-63	-77	-81	-87	1

```
df = data.iloc[:,0:7]
op = data.iloc[:, -1]
df.head()
```

	A	B	C	D	E	F	G
0	-64	-56	-61	-66	-71	-82	-81
1	-68	-57	-61	-65	-71	-85	-85
2	-63	-60	-60	-67	-76	-85	-84
3	-61	-60	-68	-62	-77	-90	-80
4	-63	-65	-60	-63	-77	-81	-87

PCA

Mean Centering the data for PCA:

```
df.mean().values
```

```
array([-52.3305, -55.6235, -54.964 , -53.5665, -62.6405, -80.985 ,
       -81.7265])
```

```
df_mc = pd.DataFrame()
# mc = Mean centered

for i in range(len(df.columns)):
    array = np.array(df.iloc[:,i] - df.mean().values[i])
    df_mc[df.columns[i]] = array
df_mc.head()
```

	A	B	C	D	E	F	G
0	-11.6695	-0.3765	-6.036	-12.4335	-8.3595	-1.015	0.7265
1	-15.6695	-1.3765	-6.036	-11.4335	-8.3595	-4.015	-3.2735
2	-10.6695	-4.3765	-5.036	-13.4335	-13.3595	-4.015	-2.2735
3	-8.6695	-4.3765	-13.036	-8.4335	-14.3595	-9.015	1.7265
4	-10.6695	-9.3765	-5.036	-9.4335	-14.3595	-0.015	-5.2735

Scaling the data for PCA:

```
df.std().values
```

```
array([11.32167655,  3.41768753,  5.31618613, 11.47198243,  9.10509259,
        6.51667158,  6.51981225])
```

```
df_mcs = pd.DataFrame()
```

```
# mc = Mean centered scaled

for i in range(len(df_mc.columns)):
    array = np.array(df_mc.iloc[:,i] / df_mc.std().values[i])
    df_mcs[df_mc.columns[i]] = array
df_mcs.head()
```

	A	B	C	D	E	F	G
0	-1.030722	-0.110162	-1.135400	-1.083814	-0.918113	-0.155754	0.111430
1	-1.384026	-0.402758	-1.135400	-0.996646	-0.918113	-0.616112	-0.502085
2	-0.942396	-1.280544	-0.947296	-1.170983	-1.467256	-0.616112	-0.348706
3	-0.765743	-1.280544	-2.452134	-0.735139	-1.577084	-1.383375	0.264808
4	-0.942396	-2.743522	-0.947296	-0.822308	-1.577084	-0.002302	-0.808842

Finding the Covariance Matrix:

```
# covariance matrix

X = df_mcs.to_numpy()
Xt = X.transpose()

XtX = np.dot(Xt, X)
df_XtX = pd.DataFrame(XtX)
df_XtX/1999.0
```

	0	1	2	3	4	5	6
0	1.000000	-0.003298	0.050814	0.921025	-0.244932	0.718429	0.686955
1	-0.003298	1.000000	0.282211	0.014604	0.200469	0.074002	0.048336
2	0.050814	0.282211	1.000000	0.078292	0.618984	-0.091622	-0.073141
3	0.921025	0.014604	0.078292	1.000000	-0.236021	0.706039	0.673294
4	-0.244932	0.200469	0.618984	-0.236021	1.000000	-0.416049	-0.361621
5	0.718429	0.074002	-0.091622	0.706039	-0.416049	1.000000	0.723172
6	0.686955	0.048336	-0.073141	0.673294	-0.361621	0.723172	1.000000

Calculating Eigen Values and Eigen Vectors:

```
from numpy.linalg import eig

Evalue, Evector = eig(XtX)
print('E-value:\n', Evalue)
print('E-vector:\n', Evector)
```

```
E-value:
 [6787.68429243 3421.80053101 1741.77047985  155.95960672  741.76231514
  512.47144992  631.55132495]
E-vector:
 [[-0.48948817  0.16468976  0.19791006 -0.71630001  0.41057236  0.10352193
   0.04169002]
 [ 0.00534144  0.42971465 -0.87398505 -0.01296522  0.20900505  0.07854757
   0.03802633]
 [ 0.07713055  0.6714255   0.24129382 -0.03120517 -0.32943421  0.16369837
  -0.59052363]
 [-0.48438445  0.18311678  0.19312126  0.69580645  0.43485981  0.14512339
  -0.01528574]
 [ 0.27160625  0.54881495  0.27858707  0.02857353  0.035565   -0.33069007
   0.66029099]
 [-0.47977075  0.02894363 -0.13116326  0.02263037 -0.25987816 -0.81040806
  -0.16422083]
 [-0.4645917   0.04293096 -0.08153934  0.0172405  -0.64908839  0.41126438
   0.43001331]]
```

Principle Components:

```
# principle components

pcs = Evector.transpose()

p1 = pcs[0]
p2 = pcs[1]
p3 = pcs[2]
p4 = pcs[3]
p5 = pcs[4]
p6 = pcs[5]
p7 = pcs[6]


print("p1: ", p1)
print("p2: ", p2)
print("p3: ", p3)
print("p4: ", p4)
```

```
print("p5: ", p5)
print("p6: ", p6)
print("p7: ", p7)
```

```
p1:  [-0.48948817  0.00534144  0.07713055 -0.48438445  0.27160625 -0.47977075
 -0.4645917 ]
p2:  [0.16468976 0.42971465 0.6714255  0.18311678 0.54881495 0.02894363
 0.04293096]
p3:  [ 0.19791006 -0.87398505  0.24129382  0.19312126  0.27858707 -0.13116326
 -0.08153934]
p4:  [-0.71630001 -0.01296522 -0.03120517  0.69580645  0.02857353  0.02263037
  0.0172405 ]
p5:  [ 0.41057236  0.20900505 -0.32943421  0.43485981  0.035565   -0.25987816
 -0.64908839]
p6:  [ 0.10352193  0.07854757  0.16369837  0.14512339 -0.33069007 -0.81040806
  0.41126438]
p7:  [ 0.04169002  0.03802633 -0.59052363 -0.01528574  0.66029099 -0.16422083
  0.43001331]
```

Calculating Scores:

We get the transformed data with scores calculated from all 7 principle components .

```
t = np.dot(X,Evector)
t
```

```
array([[ 0.71493855, -1.68148729, -0.83541376, ..., -0.60798115,
         0.017157  ,  0.10716066],
       [ 1.34999078, -1.88910661, -0.52237044, ..., -0.25842381,
         0.0910107 , -0.10824651],
       [ 1.00767508, -2.39399143,  0.17843458, ..., -0.51743149,
         0.31794851, -0.52826882],
       ...,
       [ 2.44169006,  1.37971051,  1.64808107, ..., -0.61268396,
        -0.32302525, -0.03658382],
       [ 2.31371957,  1.05903344,  1.33261272, ..., -0.19150391,
        -0.01715813,  1.19923501],
       [ 2.13180815,  2.7631949 , -0.46381358, ...,  0.11438007,
         0.19273188,  0.04801259]])
```

```
df_pca = pd.DataFrame(t, columns = cols[0:-1])
```

```
# df_pca = pd.concat([df_pca, op], axis = 1)
df_pca
```

	A	B	C	D	E	F	G
0	0.714939	-1.681487	-0.835414	-0.006798	-0.607981	0.017157	0.107161
1	1.349991	-1.889107	-0.522370	0.289725	-0.258424	0.091011	-0.108247
2	1.007675	-2.393991	0.178435	-0.155456	-0.517431	0.317949	-0.528269
3	0.647267	-3.351619	-0.045527	0.058306	0.037637	1.063579	0.678377
4	0.720425	-3.021070	1.450805	0.108942	-0.536329	-0.396719	-0.960414

Logistic Regression with PCA

Figuring out the best number of Principle Components for Logistic Regression:

We will calculate the Accuracy for model for each each number of principle components.

From the Output below, it is significant that the model starts to overfit if we use more than 3 principle components. So, we will build a model with first 3 scores.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix

for i in range(7):

    print( "no. of principle components: ", i+1)
    X_train, X_test, y_train, y_test = train_test_split(df_pca.iloc[:,:
i+1], op, test_size = 0.25)
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)

    y_pred = logreg.predict(X_test)
    print('Accuracy of logistic regression classifier on test set: {}'.
format(logreg.score(X_test, y_test)))
    print()
```

```
no. of principle components:  1
Accuracy of logistic regression classifier on test set: 0.816

no. of principle components:  2
Accuracy of logistic regression classifier on test set: 0.916

no. of principle components:  3
Accuracy of logistic regression classifier on test set: 0.966

no. of principle components:  4
Accuracy of logistic regression classifier on test set: 0.958

no. of principle components:  5
Accuracy of logistic regression classifier on test set: 0.974

no. of principle components:  6
Accuracy of logistic regression classifier on test set: 0.98

no. of principle components:  7
Accuracy of logistic regression classifier on test set: 0.99
```

Building logistic Regression model:

Principle components = 3

Calculating Accuracy, training and testing time, and Confusion matrix.

```
start = time.time()

X_train, X_test, y_train, y_test = train_test_split(df_pca.iloc[:,:3],
op, test_size = 0.25)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

stop = time.time()
print("Training Time: ", stop - start)

start = time.time()

y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on test set: {:.2f}'.
format(logreg.score(X_test, y_test)))

stop = time.time()
print("Testing Time: ", stop - start)
```

```
confusion_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", pd.DataFrame(confusion_matrix))
```

```
Training Time:  0.032458305358886720
Accuracy of logistic regression classifier on test set: 0.97
Testing Time:  0.0037679672241210938
Confusion Matrix:
      0    1    2    3
0  116    0    0    1
1    0  127    5    0
2    2    2  119    1
3    1    0    4  122
```

Decision Tree with PCA

Figuring out the best number of Principle Components for Decision Tree:

We will calculate the Accuracy for model for each each number of principle components.

From the Output below, it is significant that the model starts to overfit if we use more than 3 principle components. So, we will build a model with first 3 scores.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,classification_report,confus
ion_matrix

for i in range(7):

    print( "no. of principle components: ", i+1)
    X_train, X_test, y_train, y_test = train_test_split(df_pca.iloc[:,:
i+1], op, test_size = 0.25)
    clf_model = DecisionTreeClassifier(criterion="gini")
    clf_model.fit(X_train,y_train)

    y_pred = clf_model.predict(X_test)

    print('Accuracy of Decision Tree classifier on test set:',end='')
    print(accuracy_score(y_test,y_pred))
    print()
```

```
no. of principle components:  1
Accuracy of decision Tree classifier on test set:0.734

no. of principle components:  2
Accuracy of decision Tree classifier on test set:0.886

no. of principle components:  3
Accuracy of decision Tree classifier on test set:0.954

no. of principle components:  4
Accuracy of decision Tree classifier on test set:0.948

no. of principle components:  5
Accuracy of decision Tree classifier on test set:0.936

no. of principle components:  6
Accuracy of decision Tree classifier on test set:0.952

no. of principle components:  7
Accuracy of decision Tree classifier on test set:0.94
```

Building Decision Tree model:

Principle components = 3

Calculating Accuracy, training and testing time, and Confusion matrix.

```
start = time.time()

X_train, X_test, y_train, y_test = train_test_split(df_pca.iloc[:,:3],
op, test_size = 0.25)
clf_model = DecisionTreeClassifier(criterion="gini")
clf_model.fit(X_train,y_train)

stop = time.time()
print("Training Time: ", stop - start)

start = time.time()
y_pred = clf_model.predict(X_test)

print('Accuracy of Decision Tree classifier on test set:',end='')
print(accuracy_score(y_test,y_pred))

stop = time.time()
print("Testing Time: ", stop - start)

confusion_matrix = confusion_matrix(y_test, y_pred)
```

```
print("Confusion Matrix:\n",pd.DataFrame(confusion_matrix))
```

```
Training Time:  0.009244680404663086
Accuracy of Decision Tree classifier on test set:0.938
Testing Time:  0.004150629043579101б
Confusion Matrix:
      0    1    2    3
0  129    0    5    0
1    1  116    8    0
2    2    5  103    4
3    0    0    6  121
```

Feature Selection

```
features = pd.DataFrame(df)
removed_index = []
features.head()
```

	A	B	C	D	E	F	G
0	-64	-56	-61	-66	-71	-82	-81
1	-68	-57	-61	-65	-71	-85	-85
2	-63	-60	-60	-67	-76	-85	-84
3	-61	-60	-68	-62	-77	-90	-80
4	-63	-65	-60	-63	-77	-81	-87

Selecting Features

The following code will find the indexes for best to worst attributes.

The method used is Backward Search.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix


for i in range(7):
  accuracy = []
  for i in range(7):

      if i in removed_index:
          print("Already Removed... Hence, Accuracy = 0")
          accuracy.append(0)
          continue

  #     np.array(unused.iloc[:,i]).reshape(-1,1)
```

```
      x = features.drop(features.columns[i],axis=1)
      X_train, X_test, y_train, y_test = train_test_split(x, op, test_s
ize = 0.25)
      logreg = LogisticRegression()
      logreg.fit(X_train, y_train)


      y_pred = logreg.predict(X_test)
      accuracy.append(logreg.score(X_test, y_test))
      print('Accuracy of logistic regression classifier on test set: {}
'.format(accuracy[i]))

  max_index = np.argmax(accuracy)
  print()
  removed_index.append(max_index)
  print(removed_index)
```

```
significant_features = removed_index[::-1]
significant_features
```

```
[4, 0, 6, 3, 5, 2, 1]
```

Arranging DataFrame according to significance:

	E	A	G	D	F	C	B
0	-71	-64	-81	-66	-82	-61	-56
1	-71	-68	-85	-65	-85	-61	-57
2	-76	-63	-84	-67	-85	-60	-60
3	-77	-61	-80	-62	-90	-68	-60
4	-77	-63	-87	-63	-81	-60	-65

Logistic Regression with Feature Selection

Figuring out the best number of Features to be used for Logistic Regression:

We will calculate the Accuracy for model for each each number of features.

From the Output below, it is significant that the model starts to overfit if we use more than 2 features. So, we will build a model with first 2 significant features.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix

for i in range(7):

    print( "no. of principle components: ", i+1)
    X_train, X_test, y_train, y_test = train_test_split(df_fs.iloc[:,:i
+1], op, test_size = 0.25)
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)

    y_pred = logreg.predict(X_test)
    print('Accuracy of logistic regression classifier on test set: {}'.
format(logreg.score(X_test, y_test)))
    print()
```

```
no. of principle components:  1
Accuracy of logistic regression classifier on test set: 0.594

no. of principle components:  2
Accuracy of logistic regression classifier on test set: 0.97

no. of principle components:  3
Accuracy of logistic regression classifier on test set: 0.94

no. of principle components:  4
Accuracy of logistic regression classifier on test set: 0.958

no. of principle components:  5
Accuracy of logistic regression classifier on test set: 0.96

no. of principle components:  6
Accuracy of logistic regression classifier on test set: 0.982

no. of principle components:  7
Accuracy of logistic regression classifier on test set: 0.976
```

Building logistic Regression model:

Features = 2

Calculating Accuracy, training and testing time, and Confusion matrix.

```
start = time.time()

X_train, X_test, y_train, y_test = train_test_split(df_fs.iloc[:,:2], o
p, test_size = 0.25)
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

stop = time.time()
print("Training Time: ", stop - start)

start = time.time()

y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on test set: {:.2f}'.
format(logreg.score(X_test, y_test)))

stop = time.time()
```

```
print("Testing Time: ", stop - start)

confusion_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", pd.DataFrame(confusion_matrix))
```

```
Training Time:  0.10279059410095215
Accuracy of logistic regression classifier on test set: 0.97
Testing Time:  0.005642890930175781
Confusion Matrix:
      0    1    2    3
0  112    0    0    0
1    0  125    4    0
2    2    5  122    0
3    2    0    0  128
```

Decision Tree with Feature Selection

Figuring out the best number of Features to be used for Logistic Regression:

We will calculate the Accuracy for model for each each number of features.

From the Output below, it is significant that the model starts to overfit if we use more than 2 features. So, we will build a model with first 2 significant features.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,classification_report,confus
ion_matrix

for i in range(7):

    print( "no. of principle components: ", i+1)
    X_train, X_test, y_train, y_test = train_test_split(df_fs.iloc[:,:i
+1], op, test_size = 0.25)
    clf_model = DecisionTreeClassifier(criterion="gini")
    clf_model.fit(X_train,y_train)

    y_pred = clf_model.predict(X_test)

    print('Accuracy of Decision Tree classifier on test set:',end='')
    print(accuracy_score(y_test,y_pred))
    print()
```

```
no. of principle components:  1
Accuracy of Decision Tree classifier on test set:0.656

no. of principle components:  2
Accuracy of Decision Tree classifier on test set:0.95

no. of principle components:  3
Accuracy of Decision Tree classifier on test set:0.954

no. of principle components:  4
Accuracy of Decision Tree classifier on test set:0.966

no. of principle components:  5
Accuracy of Decision Tree classifier on test set:0.976

no. of principle components:  6
Accuracy of Decision Tree classifier on test set:0.964

no. of principle components:  7
Accuracy of Decision Tree classifier on test set:0.978
```

Building decision tree model:

Principle components = 2

Calculating Accuracy, training and testing time, and Confusion matrix.

```
start = time.time()

X_train, X_test, y_train, y_test = train_test_split(df_pca.iloc[:,:3],
op, test_size = 0.25)
clf_model = DecisionTreeClassifier(criterion="gini")
clf_model.fit(X_train,y_train)

stop = time.time()
print("Training Time: ", stop - start)

start = time.time()
y_pred = clf_model.predict(X_test)

print('Accuracy of Decision Tree classifier on test set:',end='')
print(accuracy_score(y_test,y_pred))

stop = time.time()
```

```
print("Testing Time: ", stop - start)

confusion_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n",pd.DataFrame(confusion_matrix))
```

```
Training Time:  0.011095523834228516
Accuracy of Decision Tree classifier on test set:0.972
Testing Time:  0.0043239593505859375
Confusion Matrix:
      0    1    2    3
0  132    0    2    0
1    0  120    7    0
2    0    2  126    0
3    1    0    2  108
```

Results:

<table>
<tr><th>Logistic Regression</th><th>PCA</th><th>Feature Selection</th></tr>
<tr><td>No. of Features</td><td>3</td><td>2</td></tr>
<tr><td>Accuracy</td><td>0.972</td><td>0.976</td></tr>
<tr><td>Training time</td><td>0.324</td><td>0.102</td></tr>
<tr><td>Testing time</td><td>0.003</td><td>0.004</td></tr>
<tr><td>Confusion Matrix</td><td>
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>0</td><td>116</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>127</td><td>5</td><td>0</td></tr>
<tr><td>2</td><td>2</td><td>2</td><td>119</td><td>1</td></tr>
<tr><td>3</td><td>1</td><td>0</td><td>4</td><td>122</td></tr>
</table>
</td><td>
Confusion Matrix:
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>0</td><td>112</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>125</td><td>4</td><td>0</td></tr>
<tr><td>2</td><td>2</td><td>5</td><td>122</td><td>0</td></tr>
<tr><td>3</td><td>2</td><td>0</td><td>0</td><td>128</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Decision Tree</th><th>PCA</th><th>Feature Selection</th></tr>
<tr><td>No. of Features</td><td>3</td><td>2</td></tr>
<tr><td>Accuracy</td><td>0.938</td><td>0.972</td></tr>
<tr><td>Training time</td><td>0.009</td><td>0.011</td></tr>
<tr><td>Testing time</td><td>0.004</td><td>0.005</td></tr>
<tr><td>Confusion Matrix</td><td>
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>0</td><td>129</td><td>0</td><td>5</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>116</td><td>8</td><td>0</td></tr>
<tr><td>2</td><td>2</td><td>5</td><td>103</td><td>4</td></tr>
<tr><td>3</td><td>0</td><td>0</td><td>6</td><td>121</td></tr>
</table>
</td><td>
<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th></tr>
<tr><td>0</td><td>132</td><td>0</td><td>2</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>120</td><td>7</td><td>0</td></tr>
<tr><td>2</td><td>0</td><td>2</td><td>126</td><td>0</td></tr>
<tr><td>3</td><td>1</td><td>0</td><td>2</td><td>108</td></tr>
</table>
</td></tr>
</table>

Conclusion:

We used a dataset with 7 attributes in order to predict the room for the given signal strengths. In total 2000 data points were used (1500 train data and 500 test data).

We had to apply PCA and Feature Selection on data. The two classifiers we used were Logistic Regression and Decision Tree.

For the PCA, the dataset was mean centered and scaled. Then using the Covariance matrix and, Eigen Values and Vectors, The Principle components were calculated. The scores were calculated for all the principle components using Eigen Vector (principle components) and the data.

Next, to determine the number of Principle components to be used, we built the models for all values of n (n = no. of Principle components). On looking at the accuracies, we could tell after which component the model was overfitting.

For both Logistic Regression and Decision Tree, the model was overfitting after 3rd component. So we built both the models for n = 3. The results (training time, testing time, accuracy and confusion matrix was noted.

For the Feature Selection, we determined the best to worst Features using the backward search technique. The data was arranged according to the best to worst columns.

Next, to determine the number of Features to be used, we built the models for all values of n (n = no. of features). On looking at the accuracies, we could tell after how many features the model was overfitting.

For both Logistic Regression and Decision Tree, the model was overfitting after 2 features. So we built both the models for n = 2. The results (training time, testing time, accuracy and confusion matrix was noted.

From the results (*given in results.docx*), it can be said that Feature

Selection gives better results than PCA for both the classifiers. The aim of PCA is dimensionality reduction. We remove the attributes with low variance. This will result in losing information. Also, the PCA doesn't consider target variable for this approach. On the flip side, in Feature Selection, we are removing the attribute based on accuracy considering the target value.