University of Hertfordshire

School of Computer Science

BSc Computer Science (Software Engineering)

Module: Machine Learning and Neural Computing

Coursework

Data Classification

16040776,

Level 6

Academic Year 2018-19

Task 1- Data Exploration

Task 1 set up the data variables that were to be used later in following tasks. Firstly, the data was imported using pandas both the test and training data.

After which the data from two of the features specifically NUM_OPERANDS and NUM_OPERATORS were plotted against each other.

The data was split using the '.iloc' method which utilises a mechanism of splitting the data based on rows and columns. The main data to be used was extracted by selecting all columns and only the first 12 features. The labels use was weather the data was non-defective -1 or defective 1.

Both data sets were normalised using 'StandarScaler()' using only the training data as the parameter.

PCA analysis was then performed on the scaled training data set followed by a scree plot that captured the variances of each principal component. It was observed that PC1 had the greatest variance of above 70% following which all PCA's fell below 10% decreasing more after each principal component.

The test set was then projected onto the same PCA space provided by the training set.

After which both plots of PCA 1 and 2 for the test and training data was plotted in a subplot.

Critical analysis

The training dataset that has had PCA analysis performed on it shows that the data is closer towards pc2 than pc1 the same is also the case with the test set that has had PCA performed upon it. Another commonality between the two is that the darker purple plots are more clustered together in both test and train subplots closer to pc2 whereas the yellow plots also group in the same areas as the darker plots however are much more dispersed in comparison along pc1. This is significant as pc1 has greater variance than 2.

Task 2

In task 2 the training data was split into two the training data set and a validation set. In some instances, it would also have to be split to create a test set however that is already provided and so is not necessary. 30% of the original training data set was used for the validations set. The training set had 280 datapoints and the validation set had 120.

Both data sets training and validation were normalised using 'StandardScaler()' with the training set used as the parameter similarly to task 1. It should be noted that the original split training data was

used via '.iloc' and not the training data that was previously normalised and upon which principle component analysis was carried out upon.

Task 3- the Most Suitable Parameters

In task 3 a support vector machine was used on the training set from task 2. From the specification it was stated that the kernel to be used was the Gaussian radial basis or 'rbf'. Various hyperparameters were tested by adjusting the cost and gamma values the model created is then validated using the scaled validation set. 6 combination of the values were tested to find the most optimal [C=50, γ=0.01], [C=50, γ=10], [C=5, γ=1], [C=100, γ=0.01], [C=100, γ=10] and [C=100, γ=1]. The most optimal was found to be the first with a cost of 50 and gamma of 0.01 which gave an accuracy of 0.725.

A classification report is created using the most optimal model that was validated.

Task 4- Non-linear Classification

The optimal combination of parameters of the cost and gamma from task 3 are used on the test data that has been normalised and scaled using the training data as the parameter as was done in task 1. It should be noted again that this was the original test and training data.

After which a model is created using the training dataset and is validated via training data as opposed to a validation set as was done in task 3 this resulted in an accuracy of 0.57 or 57%.

Finally a function was created to determine the number of false positives in the prediction in comparison the actual data were those that were actually labelled as non-defective or -1 are incorrectly predicted as defective or 1. The number of false positives identified with this false positives function was 4. To confirm that this was the correct value a confusion matrix was used to identify the number of true positives, false positives, false negatives, and true negatives which helped to corroborate the number of false positives being 4. It should be noted that the confusion matrix also identified that there were 42 true positives, 39 false negatives, and 15 true negatives which added up to the total number of data points in the test data of 100.

Appendix: Task 1

In [251]:

```
import pandas as pd

"""load the testing and training data set using pandas"""

testing_data = pd.read_csv('Desktop/Data Classification-cw/tes
tingSet-1.csv')
training_data = pd.read_csv('Desktop/Data Classification-cw/tr
ainingSet-1.csv')
```

In [252]:

```
testing_data
```

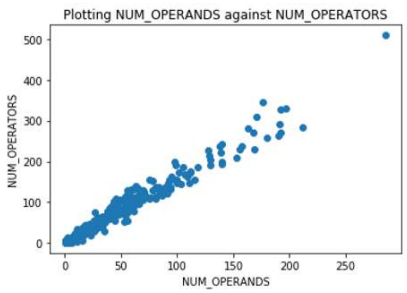
Out[252]:

	LOC_BLANK	BRANCH_COUNT	LOC_CODE_AND_COMMENT	LO
0	1	5	0	
1	0	1	0	
2	0	7	0	
3	0	1	0	
4	0	1	0	
...	...	...	...	
95	0	3	0	
96	0	1	0	
97	0	1	0	
98	0	1	0	
99	0	1	0	

100 rows x 14 columns

In [254]:

```
import matplotlib.pyplot as plt
import numpy as np
#plot comparing NUM_OPERANDS to NUM_OPERATORS
fig = plt.figure()

plt.title("Plotting NUM_OPERANDS against NUM_OPERATORS")
plt.xlabel("NUM_OPERANDS")
plt.ylabel("NUM_OPERATORS")
plt.scatter(training_data.iloc[:, 8],training_data.iloc[:, 9],
marker = 'o')
plt.show()
```



In [255]:

```
"""Splitting data"""
train_inputs = training_data.iloc[:, 0:13]
train_labels = training_data.iloc[:,13]

test_inputs = testing_data.iloc[:, 0:13]
test_labels = testing_data.iloc[:,13]
```

In [256]:

```
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
#StandardScaler using parameters frpm the training data
scaler = preprocessing.StandardScaler().fit(train_inputs)

#Normalising the data
scaled_data_training = scaler.transform(train_inputs)
scaled_data_test = scaler.transform(test_inputs)
```

In [257]:

```
import numpy as np

from sklearn.decomposition import PCA

pca = PCA()
train = pca.fit_transform(scaled_data_training)
```

In [258]:

```
pca.explained_variance_
```

Out[258]:

```
array([9.62022196e+00, 1.01743247e+00, 8.77802044e
-01, 5.30464425e-01,
       4.24033994e-01, 2.33567446e-01, 1.80840247e
-01, 6.63390402e-02,
       3.91313540e-02, 2.88607814e-02, 1.09915563e
-02, 1.91247185e-03,
       9.83665542e-04])
```

In [259]:

```
pca.explained_variance_ratio_
```

Out[259]:

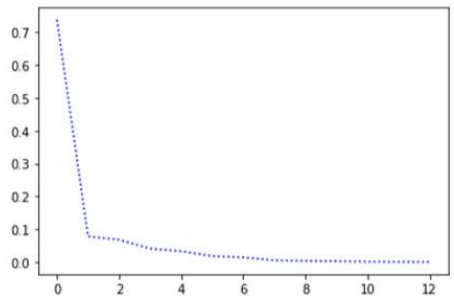
```
array([7.38167031e-01, 7.80683757e-02, 6.73544261e
-02, 4.07029434e-02,
       3.25364546e-02, 1.79218098e-02, 1.38760113e
-02, 5.09024559e-03,
       3.00257890e-03, 2.21450996e-03, 8.43390568e
-04, 1.46745436e-04,
       7.54774137e-05])
```

In [260]:

```
plt.plot(pca.explained_variance_ratio_, color='blue', linestyl
e='dotted')
```

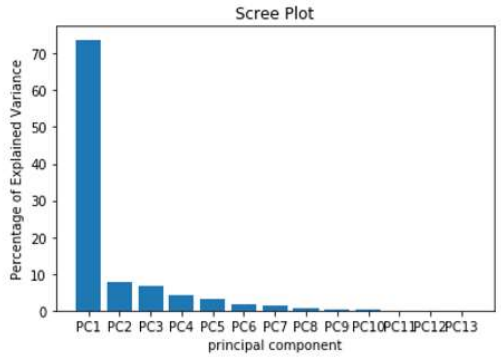
Out[260]:

```
[<matplotlib.lines.Line2D at 0x1a1d1b7c50>]
```



In [261]:

```
per_var = np.round(pca.explained_variance_ratio_*100, decimals
= 1)
labels = ['PC' + str(n) for n in range(1, len(per_var)+1)]

plt.bar(x=range(1,len(per_var)+1), height=per_var, tick_label=
labels)
plt.ylabel('Percentage of Explained Variance')
plt.xlabel('principal component')
plt.title('Scree Plot')
plt.show()
```

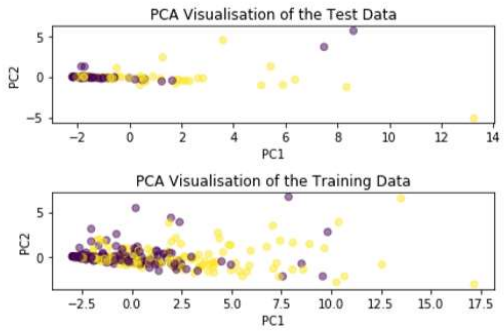


In [262]:

```
test = pca.fit_transform(scaled_data_test)
```

In [263]:

```
plt.subplot(211)
plt.title("PCA Visualisation of the Test Data")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.scatter(test[:, 0],test[:, 1],c=test_labels, marker = 'o',
alpha = 0.5)

plt.subplot(212)
plt.title("PCA Visualisation of the Training Data")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.scatter(train[:, 0],train[:, 1],c=train_labels, marker = '
o', alpha =0.5)

plt.tight_layout()
plt.show()
```



Appendix: Task 2

In [266]:

```
"""Task 2"""
from sklearn.model_selection import train_test_split

#Xtrain, Xtest, ytrain, ytest = train_test_split(train_inputs,
train_labels, test_size=0.3, random_state=42)

SXtrain, Xvalid, Sytrain, vtest = train_test_split(train_input
s, train_labels, test_size=0.30,
                                                   random_state=4
2)
```

In [267]:

```
len(SXtrain)
```

Out[267]:

```
280
```

In [268]:

```
len(Xvalid)
```

Out[268]:

```
120
```

In [269]:

```
scaler2 = preprocessing.StandardScaler().fit(SXtrain)

"""Scale the training set"""
scaled_trnX = scaler2.transform(SXtrain)
"""Scale the validation set"""
scaled_valX = scaler2.transform(Xvalid)
```

Appendix: Task 3

In [270]:

```
"""Task 3"""

from sklearn.svm import SVC
from sklearn import metrics

svc1 = SVC(kernel='rbf', class_weight='balanced', C=50, gamma=
0.01)
model1 = svc1.fit(scaled_trnX, Sytrain)

vyfit1 = model1.predict(scaled_valX)

print('Accuracy:', metrics.accuracy_score(vtest, vyfit1))
```

```
Accuracy: 0.725
```

In [271]:

```
svc2 = SVC(kernel='rbf', class_weight='balanced', C=50, gamma=
10)
model2 = svc2.fit(scaled_trnX, Sytrain)

vyfit2 = model2.predict(scaled_valX)

print('Accuracy:', metrics.accuracy_score(vtest, vyfit2))
```

```
Accuracy: 0.7083333333333334
```

In [272]:

```
svc3 = SVC(kernel='rbf', class_weight='balanced', C=5, gamma=1
)
model3 = svc3.fit(scaled_trnX, Sytrain)

vyfit3 = model3.predict(scaled_valX)

print('Accuracy:', metrics.accuracy_score(vtest, vyfit3))
```

```
Accuracy: 0.7083333333333334
```

In [273]:

```
svc4 = SVC(kernel='rbf', class_weight='balanced', C=100, gamma
=0.01)
model4 = svc4.fit(scaled_trnX, Sytrain)

vyfit4 = model4.predict(scaled_valX)

print('Accuracy:', metrics.accuracy_score(vtest, vyfit4))
```

```
Accuracy: 0.7166666666666667
```

In [274]:

```
svc5 = SVC(kernel='rbf', class_weight='balanced', C=100, gamma
=10)
model5 = svc5.fit(scaled_trnX, Sytrain)

vyfit5 = model5.predict(scaled_valX)

print('Accuracy:', metrics.accuracy_score(vtest, vyfit5))
```

```
Accuracy: 0.7083333333333334
```

In [275]:

```
svc6 = SVC(kernel='rbf', class_weight='balanced', C=100, gamma
=1)
model6 = svc6.fit(scaled_trnX, Sytrain)

vyfit6 = model6.predict(scaled_valX)

print('Accuracy:', metrics.accuracy_score(vtest, vyfit6))
```

```
Accuracy: 0.6833333333333333
```

In [276]:

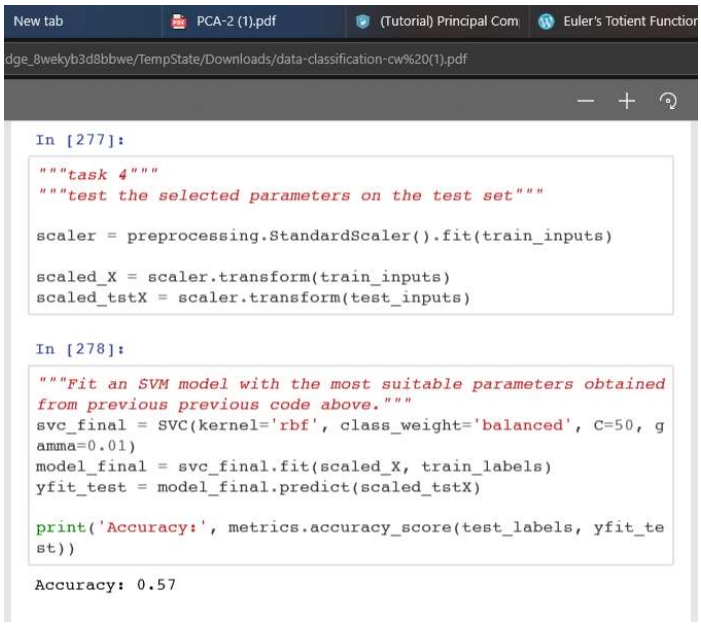
```
from sklearn.metrics import classification_report
print(classification_report(vtest, vyfit1,
                            train_labels))
```

```
              precision    recall  f1-score   supp
ort

           1       0.93      0.45      0.60
56
          -1       0.67      0.97      0.79
64
           1       0.93      0.45      0.60
56
          -1       0.67      0.97      0.79
64
```

...

```
64
          -1       0.67      0.97      0.79
64

    accuracy                           0.73     24
032
   macro avg       0.79      0.71      0.70     24
032
weighted avg       0.79      0.73      0.70     24
032
```

Appendix: Task 4



In [277]:

```
"""task 4"""
"""test the selected parameters on the test set"""

scaler = preprocessing.StandardScaler().fit(train_inputs)

scaled_X = scaler.transform(train_inputs)
scaled_tstX = scaler.transform(test_inputs)
```

In [278]:

```
"""Fit an SVM model with the most suitable parameters obtained
from previous previous code above."""
svc_final = SVC(kernel='rbf', class_weight='balanced', C=50, g
amma=0.01)
model_final = svc_final.fit(scaled_X, train_labels)
yfit_test = model_final.predict(scaled_tstX)

print('Accuracy:', metrics.accuracy_score(test_labels, yfit_te
st))
```

```
Accuracy: 0.57
```

In [283]:

```
from sklearn.metrics import confusion_matrix

confusion_matrix(test_labels, yfit_test)
```

Out[283]:

```
array([[42,  4],
       [39, 15]])
```

In [284]:

```
"""function to find false positives, those that were actually
labeled as non-defective or -1,
   but are incorrectly predicted as defective or 1"""
def false_positives(actual_val, predicted_val):

    false_positives = 0

    for i in range(len(actual_val)):
        if actual_val[i] == -1 and predicted_val[i] == 1:
            false_positives +=1

    return false_positives
```

In [285]:

```
false_positives(test_labels, yfit_test)
```

Out[285]:

```
4
```

In [ ]:

Entirety of the code can be found attached alongside this document as a pdf.