SIT720 Assignment Two

Name: Jose Arturo Gil Alonso

Student ID: 218 659 676

MACHINE LEARNING

PART 1:

1.1

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import *
from decimal import *
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

In [2]:
```
#Read the training data
```

```
dtrain = pd.read_csv('train_wbcd.csv',delimiter=",",header=0).values
rows, cols = dtrain.shape
print("There are {} rows and  {} columns in the Training dataset".forma
t(rows, cols))

#Reading the testing data:
dtest = pd.read_csv('test_wbcd.csv',delimiter=",",header=0).values
rows, cols = dtest.shape
print("There are {} rows and  {} columns in the Testing dataset".format
(rows, cols))
```

```
There are 100 rows and  32 columns in the Training dataset
There are 20 rows and  32 columns in the Testing dataset
```

In [3]:

```
#Features in the Training dataset
trainingf= dtrain[:, 2:32].T
print ("Features in Training dataset:", len(trainingf))
```

```
Features in Training dataset: 30
```

In [4]:

```
# Features in the Testing dataset
testingf= dtest[:, 2:32].T
print ("Features in Testing dataset:", len(testingf))
```

```
Features in Testing dataset: 30
```

In [5]:

```
# Assign the training data set to Variables
train1 = dtrain[:,2:32]
train2 = dtrain[:,1:2]

# Assign the test data set to Variables
test1 = dtest[:,2:32]
test2 = dtest[:,1:2]
```

In [7]:

```
#Features with missing entries
dtrain_missf = sum(sum(pd.isnull(dtrain)))
print('Features that have entries missing  in training dataset:', dtrai
n_missf)
```

```
dtest_missf = sum(sum(pd.isnull(dtest)))
print ('Features that have entries missing  in testing dataset: ', dtes
t_missf)
```

```
Features that have entries missing  in training dataset: 2
Features that have entries missing  in testing dataset:  1
```

In [8]:

```
#Use of Median for filling features
me_dtrain = train1[~pd.isnull(train1[:,20])]
me_dtrain = np.median(me_dtrain[:,20])
train1[pd.isnull(train1[:,20])]=me_dtrain

me_dtest = test1[~pd.isnull(test1[:,20])]
me_dtest = np.median(me_dtest[:,20])
test1[pd.isnull(test1[:,20])]=me_dtest
```

It was used Median as the most accurate metric to replace the null values as with the mean as there are some 0 or null values, those are also taking into account to calculate the average, however, the median metric, does not take those null values into account to fill the gaps

In [9]:

```
#Data Normalization in Training and Testing datasets
Norm_dtra=preprocessing.normalize(train1)
Norm_dtest= preprocessing.normalize(test1)
```

1.2 Log Regression

In [12]:

```
# Logistic Regression training model using L1 Regularization
alpha = 0.1
logrm_L1 = LogisticRegression(C=1/alpha, penalty='l1')
logrm_L1.fit(train1, train2)
L1_prediction = logrm_L1.predict(test1)
import warnings
warnings.filterwarnings('ignore');
```

In [14]:

```
#Evaluate model
accuracy_m = accuracy_score(L1_prediction, test2)
print ("Model Accuracy: {}%".format(np.round(accuracy_m*100, decimals=2
)))

precision_m = precision_score(L1_prediction, test2,average ='weighted')
print ('Precision is: ', precision_m)

recall_m = recall_score(L1_prediction, test2,average ='weighted')
print ('Recall is: ',recall_m)

F1sc_m = 2 * (precision_m * recall_m) / (precision_m + recall_m)
print ('F1-score is :',F1sc_m)

confusion_m = confusion_matrix(L1_prediction, test2)
print ('Confusion matrix is : ',confusion_m)
```

```
Model Accuracy: 85.0%
Precision is:  0.8488095238095237
Recall is:  0.85
F1-score is : 0.8494043447792572
Confusion matrix is :  [[12  1]
 [ 2  5]]
```

In [15]:

```
#Logistic Regression training model using L2 Regularization
v_lambda = 0.1
logrm_L2 = LogisticRegression(C=1/v_lambda, penalty='l2')
logrm_L2.fit(train1, train2)
L2_prediction = logrm_L2.predict(test1)
```

In [17]:

```
#Evaluate model
accuracy_m2 = accuracy_score(L2_prediction, test2)
print ("Model Accuracy: {}%".format(np.round(accuracy_m2*100, decimals=
2)))

precision_m2 = precision_score(L2_prediction, test2,average ='weighted'
)
print ('Precision: ', precision_m2)
```

```
recall_m2 = recall_score(L2_prediction, test2,average ='weighted')
print ('Recall: ',recall_m2)

F1sc_m2 = 2 * (precision_m2 * recall_m2) / (precision_m2 + recall_m2)
print ('F1-score:',F1sc_m2)

confusion_m2 = confusion_matrix(L2_prediction, test2)
print ('Confusion matrix: ',confusion_m2)
```

```
Model Accuracy: 85.0%
Precision:  0.8488095238095237
Recall:  0.85
F1-score: 0.8494043447792572
Confusion matrix:  [[12  1]
 [ 2  5]]
```

1.3 Best hyper-parameter

In [18]:

```
#Filtering the warnings
import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning
)
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
from warnings import simplefilter
simplefilter(action='ignore', category=ConvergenceWarning)
simplefilter(action='ignore', category=FutureWarning)
```

In [20]:

```
# best value of alpha value and lambda value
fID=(218659676%3)
print(fID)
```

```
2
```

In [21]:

```
#Using for loop to perform the 10 random splits in L1
```

```
from sklearn.model_selection import train_test_split
def f1(alpha):
    precision_ml_array=[]
    for item in range(0,10):
        trainseries= np.concatenate((Norm_dtra,train2),axis=1)
        train, validation =train_test_split(trainseries, test_size=0.3)
        d_train1 = train[:,0:30]
        d_train2 = train[:,30:31]
        validt1 = train[:,0:30]
        validt2= train[:,30:31]
        logrm_L1 = LogisticRegression(C=alpha, penalty='l1')
        logrm_L1.fit(train1, train2)
        Log_reg = logrm_L1.predict(validt1)
        precision_ml = precision_score(validt2, Log_reg, average='weigh
ted')
        precision_ml_array.append(precision_ml)
    return np.mean(precision_ml_array)
```

In [22]:

```
#Hyper-parameter values
set1=[0.1,1,3,10,33,100,333,1000, 3333, 10000, 33333]
bestalpha1=[]
bestalpha1.append(f1(10))
bestalpha1.append(f1(1))
bestalpha1.append(f1(0.3333))
bestalpha1.append(f1(0.1))
bestalpha1.append(f1(0.30303))
bestalpha1.append(f1(0.01))
bestalpha1.append(f1(0.003003))
bestalpha1.append(f1(0.001))
bestalpha1.append(f1(0.00030003))
bestalpha1.append(f1(0.0001))
bestalpha1.append(f1(3e-5))

#best alpha value from set:
select_bestalpha1=set1[np.argmax(bestalpha1)]

print ('The precision score for 10 split alpha values:', bestalpha1)
print ('L1 Best alpha value:-' , select_bestalpha1)
```

```
import warnings
warnings.filterwarnings('ignore');
```

```
The precision score for 10 split alpha values: [0.988849847352965, 0.97
80885228358691, 0.9437066404926503, 0.9441266508353754, 0.9458654050139
834, 0.8850557872766048, 0.8818574955865381, 0.7133232446689235, 0.1747
7551020408164, 0.1881428571428571, 0.3450408163265306]
L1 Best alpha value:- 0.1
```

In [23]:

```
##Using for loop to perform the 10 random splits in L2
def f2(lambdav):
    precision_ml_array=[]
    for item in range(0,10):
        trainseries= np.concatenate((Norm_dtra,train2),axis=1)
        train, validation =train_test_split(trainseries, test_size=0.3)
        d_train1 = train[:,0:30]
        d_train2 =train[:,30:31]
        validt1 = train[:,0:30]
        validt2 = train[:,30:31]
        logrm_L2 = LogisticRegression(C=lambdav, penalty='l2')
        logrm_L2.fit(d_train1, d_train2)
        Log_reg = logrm_L2.predict(validt1)
        precision_ml = precision_score(validt2, Log_reg, average='weigh
ted')
        precision_ml_array.append(precision_ml)
    return np.mean(precision_ml_array)
```

In [24]:

```
#Hyper-parameter values
lambda_range=[0.001,0.003,0.01,0.03,0.1,0.3,1,3,10,33]
bestlambda1 =[]
bestlambda1.append(f2(10))
bestlambda1.append(f2(1))
bestlambda1.append(f2(0.3333))
bestlambda1.append(f2(0.1))
bestlambda1.append(f2(0.30303))
bestlambda1.append(f2(0.01))
bestlambda1.append(f2(0.003003))
bestlambda1.append(f2(0.001))
bestlambda1.append(f2(0.00030003))
```

```
bestlambda1.append(f2(0.0001))
bestlambda1.append(f2(3e-5))

# best lambda value from set:
select_bestlambda1=lambda_range[np.argmax(bestlambda1)]
print ('The precision score for 10 split lambda:', bestlambda1)
print ('Best lambda for L2:-' , select_bestlambda1)
import warnings
warnings.filterwarnings('ignore');
```

```
The precision score for 10 split lambda: [0.8861326162529567, 0.7022233
373226138, 0.43487969970596935, 0.3196530612244898, 0.3625510204081633,
0.33391836734693875, 0.3923027210884354, 0.35214285714285715, 0.3373877
5510204083, 0.34871428571428575, 0.34226530612244893]
Best lambda for L2:- 0.001
```

In [25]:

```
# Performance of Prediction evaluation on the test data for l1

log_m = LogisticRegression(C=1/select_bestalpha1,penalty='l1')

log_m.fit(Norm_dtra, train2)

predict = log_m.predict(Norm_dtest)

true_label=test2

print ("Accuracy: {}%".format(np.round(accuracy_score(true_label, predi
ct)*100, decimals=2)))

print ("Precision: ",precision_score(true_label,predict,average='weight
ed'))

print ("Confusion matrix:",confusion_matrix(true_label,predict))

#Top 5 features selected:

weightsf = log_m.coef_

weights_f=((np.argsort(weightsf))[0])[::-1]
```

```
print ("Top 5 features selected:",weights_f[0:5]+1)
```

```
Accuracy: 95.0%
Precision:  0.9533333333333334
Confusion matrix: [[14  0]
 [ 1  5]]
Top 5 features selected: [24 30 29  2  5]
```

In [26]:

```
#Prediction performance evaluation in the test data for l2

log_m2 = LogisticRegression(C=1/select_bestlambda1,penalty='l2')

log_m2.fit(Norm_dtra, train2)

prediction_value = log_m2.predict(Norm_dtest)

true_label=test2

print ("Accuracy {}%".format(np.round(accuracy_score(true_label, predic
tion_value)*100, decimals=2)))

print ("Precision: ",precision_score(true_label,prediction_value,averag
e='weighted'))

print ("Confusion matrix",confusion_matrix(true_label,prediction_value
))

#Top five selected features:
weightsf = log_m2.coef_

weights_f=((np.argsort(weightsf))[0])[::-1]

print ("The top 5 features selected in decreasing order of feature weig
hts.",weightsf[0:5]+1)
```

```
Accuracy 95.0%
Precision:  0.9533333333333334
Confusion matrix [[14  0]
 [ 1  5]]
```

```
The top 5 features selected in decreasing order of feature weights. [[

-7.56380959   4.92381582 -54.86674005 -17.21235271   3.82714468
    4.10778191   4.37454582   4.0939792    3.79219237   3.81635835
    3.55178802   4.05418679   3.51215518   0.96845829   3.88374047
    3.9365957    3.97964188   3.90546657   3.8658898    3.88927339
   -8.72071464  14.8160534  -45.31081598  17.97557248   3.85013976
    4.65583063   5.17228707   4.25804679   3.78985538   3.86648566]]
```

Part 2:

2.1.

In [28]:
```
#Reading the MNIST dataset
dtmt = pd.read_csv('reduced_mnist.csv',delimiter=',',header=0).values
rows, cols = dtmt.shape
print("The dataset named MNIST contains {} rows and  {} columns".format
(rows, cols))
```

```
The dataset named MNIST contains 2520 rows and  785 columns
```

In [29]:
```
# Printing features in the dataset
print ("The dataset contains {} features".format(len(dtmt.T[1:,:])))

# Printing the Unique labels
unique_labels = dtmt[:,0]
print ("The unique labels identified are: ",np.unique(unique_labels))
```

```
The dataset contains 784 features
The unique labels identified are:  [0 1 2 3 4 5 6 7 8 9]
```

In [30]:
```
# Splitting the data, 70% training and 30% testing. Fit a One-vs-Rest C
lassifier
from sklearn.model_selection import train_test_split
```

In [31]:
```
# Training 70% testing 30% dataset deviation
train_dt1 = dtmt[0:1764,1:]
train_dt2 = dtmt[0:1764,0:1]
test_dt1 = dtmt[1764:2520,1:]
test_dt2 = dtmt[1764:2520,0:1]
```

In [32]:
```
log_rm = LogisticRegression(C=1, penalty='l1')
log_rm.fit(train_dt1, train_dt2)
predict = log_rm.predict(test_dt1)
```

In [33]:
```
m_accu = accuracy_score(predict, test_dt2)
print ('The Accuracy is: {}% '.format(np.round(m_accu*100, decimals=1
)))
```

```
The Accuracy is: 83.1%
```

In [34]:
```
Precision = precision_score(predict, test_dt2,average ='weighted')
print ('The precision is: ', Precision)
```

```
The precision is:  0.8326798912105154
```

In [35]:
```
Recall = recall_score(predict, test_dt2,average ='weighted')
print ('The Recall is: ',Recall)
```

```
The Recall is:  0.8306878306878307
```

2.2.

In [36]:
```
#Log regression classifier
def f3(C):
    prec_val_score=[]
    for item in range(0,10):
        dtmtrain1 = dtmt[0:1764,:]
        data_train,data_value=train_test_split(dtmtrain1, test_size=0.3
)
        dtm_train1=data_train[:,1:786]
```

```
        dtm_train2=data_train[:,0:1]
        data_xvalidation=data_value[:,1:786]
        data_yvalidation=data_value[:,0:1]
        Logr_mdl= LogisticRegression(C=C,penalty='l1')
        Logr_mdl.fit(dtm_train1,dtm_train2)
        label_pred = Logr_mdl.predict(data_xvalidation)
        truelabel=data_yvalidation
        prec_val = precision_score(truelabel,  label_pred, average='wei
ghted')
        prec_val_score.append(prec_val)
    return np.mean(prec_val)
```

In [37]:

```
#Hyper-parameter values appending
alpha_set=[0.1,1,3,10,33,100,333,1000, 3333, 10000, 33333]
alphav=[]
alphav.append(f3(10))
alphav.append(f3(1))
alphav.append(f3(0.3333))
alphav.append(f3(0.1))
alphav.append(f3(0.30303))
alphav.append(f3(0.01))
alphav.append(f3(0.003003))
alphav.append(f3(0.001))
alphav.append(f3(0.00030003))
alphav.append(f3(0.0001))
alphav.append(f3(3e-5))

import warnings
warnings.filterwarnings('ignore');
```

In [39]:

```
best_alphav=alpha_set[np.argmax(alphav)]
print ("Precision score after 10 iterations taking random splits of alp
ha values:-",alphav)
print ("Regularisation of alpha value for L1:-",best_alphav)
```

```
Precision score after 10 iterations taking random splits of alpha value
s:- [0.8207594169571201, 0.8323981402067938, 0.8276129144992193, 0.8371
473543723654, 0.8299474827998615, 0.8648124682779728, 0.870622791856237
9, 0.8398396632319594, 0.8029388751966412, 0.6855097299179604, 0.533044
```

```
3407812605]
Regularisation of alpha value for L1:- 333
```

In [40]:

```
## Training data for validation

def flog(C):
    prec_val_score=[]
    for item in range(0,10):
        dtmtrain1 = dtmt[0:1764,:]
        data_train,data_value=train_test_split(dtmtrain1, test_size=0.3
)
        dtm_train1=data_train[:,1:786]
        dtm_train2=data_train[:,0:1]
        data_xvalidation=data_value[:,1:786]
        data_yvalidation=data_value[:,0:1]
        Logr_mdl= LogisticRegression(C=C,penalty='l1')
        Logr_mdl.fit(dtm_train1,dtm_train2)
        label_pred = Logr_mdl.predict(data_xvalidation)
        truelabel=data_yvalidation
        prec_val = precision_score(truelabel,  label_pred, average='wei
ghted')
        prec_val_score.append(prec_val)
    return np.mean(prec_val)
```

In [41]:

```
#Apppending  hyper-parameter values

alpha_set=[0.1,1,3,10,33,100,333,1000, 3333, 10000, 33333]
dtm_trainalpha=[]
dtm_trainalpha.append(flog(10))
dtm_trainalpha.append(flog(1))
dtm_trainalpha.append(flog(0.3333))
dtm_trainalpha.append(flog(0.1))
dtm_trainalpha.append(flog(0.30303))
dtm_trainalpha.append(flog(0.01))
dtm_trainalpha.append(flog(0.003003))
dtm_trainalpha.append(flog(0.001))
dtm_trainalpha.append(flog(0.00030003))
```

```
dtm_trainalpha.append(flog(0.0001))
dtm_trainalpha.append(flog(3e-5))
best_train_alpha=alpha_set[np.argmax(dtm_trainalpha)]
print ("L1 precision score for 10 random splits of each alpha values:-"
,dtm_trainalpha)
print ("Alpha value for L1 regularisation:-",best_train_alpha)
```

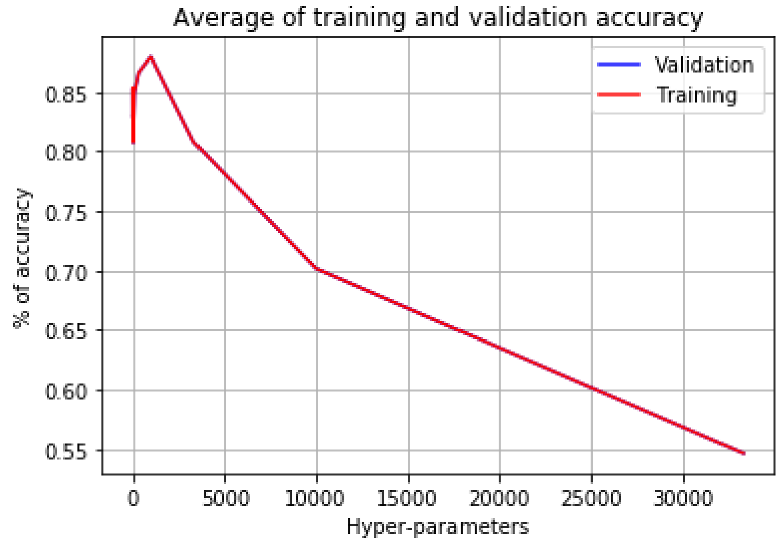
```
L1 precision score for 10 random splits of each alpha values:- [0.82913
2488813412, 0.8287061723687771, 0.8535590040606887, 0.8342723631920035,
0.8074046873351741, 0.84934977832183, 0.866354312143041, 0.879676388324
6758, 0.8077852869437667, 0.7017505143925317, 0.5462867773993325]
Alpha value for L1 regularisation:- 1000
```

In [42]:

```
# Graph
import matplotlib.pyplot as plt
plt.plot(alpha_set,dtm_trainalpha,color='b',label='Validation')
plt.plot(alpha_set,dtm_trainalpha,color='r',label='Training')
plt.xlabel('Hyper-parameters')
plt.ylabel('% of accuracy')
plt.title('Average of training and validation accuracy')
plt.grid()
plt.legend(loc="upper right")
```

Out[42]:

```
<matplotlib.legend.Legend at 0x21162bcf438>
```



2.2.2

In [43]:
```
#Confusion matrix

alpha_v= Decimal(best_alphav)
Model= LogisticRegression(C=(1/alpha_v),penalty='l1')
Model.fit(train_dt1,train_dt2)
Predictionlast = Model.predict(test_dt1)
print ("features considered non-zero are:",np.count_nonzero(Model.coef_
))
Conf_Matrix =confusion_matrix(test_dt2,Predictionlast)
print ("Confusion matrix is:",Conf_Matrix)
```

```
features considered non-zero are: 986
Confusion matrix is: [[71  0  1  0  0  0  1  1  1  0]
 [ 0 89  1  0  0  0  0  0  2  0]
 [ 0  2 70  0  2  0  0  1  4  2]
 [ 1  1  5 65  0  2  1  0  4  2]
 [ 1  3  1  0 60  1  2  0  2  6]
 [ 1  0  0  6  0 57  3  0  8  0]
 [ 2  1  3  0  0  0 62  0  0  0]
```

```
 [ 2  1  3  0  0  0 62  0  0  0]
 [ 1  3  0  0  3  1  0 59  1  4]
 [ 0  0  1  1  1  2  1  0 65  0]
 [ 0  0  1  1  5  0  0  2  1 55]]
```

In [44]:

```python
#Precision, recall and accuracy for each class.
Fpos = Conf_Matrix.sum(axis=0) - np.diag(Conf_Matrix)
FNeg = Conf_Matrix.sum(axis=1) - np.diag(Conf_Matrix)
TruePositive = np.diag(Conf_Matrix)
TrueNegative = Conf_Matrix.sum() - (Fpos + FNeg + TruePositive)
accuracy=[]
Precision=[]
recall=[]
for item in range(0,10):
    accuracy.append(float(TruePositive[item]+TrueNegative[item])/float(
TruePositive[item]+Fpos[item]+FNeg[item]+TrueNegative[item])*100)
    Precision.append(float(TruePositive[item])/float(TruePositive[item]
+Fpos[item]))
    recall.append(float(TruePositive[item])/float(TruePositive[item]+FN
eg[item]))
print ('Accuracy for class:',accuracy)
print ('Precision for Class:',Precision)
print ('Recall for Class:',recall)
```

```
Accuracy for class: [98.67724867724867, 98.28042328042328, 96.825396825
39682, 96.82539682539682, 96.42857142857143, 96.82539682539682, 98.1481
4814814815, 97.75132275132276, 96.16402116402116, 96.82539682539682]
Precision for Class: [0.922077922077922, 0.898989898989899, 0.843373493
9759037, 0.8904109589041096, 0.8450704225352113, 0.9047619047619048, 0.
8857142857142857, 0.9365079365079365, 0.7386363636363636, 0.79710144927
53623]
Recall for Class: [0.9466666666666667, 0.967391304347826, 0.86419753086
41975, 0.8024691358024691, 0.7894736842105263, 0.76, 0.911764705882352
9, 0.8194444444444444, 0.9154929577464789, 0.8461538461538461]
```

2.3. Underfitting or Overfitting

It can be seen the average training performance and validation are decreasing in the same direction. As it is known, an underfitting model is less flexible than the overfitting as it has high bias and low variance.

Thus, as it can be identified, the model does not have a clear overfitting or underfitting