NEURAL NETWORKS
ASSIGNMENT 3

TASLEEM SULTHANA
REGD NO 18MCMT17
MTECH CS

INTRODUCTION:

Assignment is to solve real world problems using SVM
Given dataset is arcene cancer data consisting of train ,validation data and their labels.

Data Set	File Name
Train_data	arcene_train.data
Train_lables	arcene_train.labels
Valid_data	arcene_valid.data
Valid_lables	arcene_valid.labels

The information abot the above data is as follows:
Arcene Cancer Data set

- Number of features : 10,000
- Number of Training Data : 100
- Number of Testing Data(Validation Set is used) : 100

IMPLEMENTATION:

The set of imports we are using for the implementation of the svm on the given data are:

“import numpy as np

from sklearn import svm,metrics

from sklearn.preprocessing import StandardScaler

from sklearn import preprocessing

from sklearn.decomposition import PCA

from sklearn.model_selection import KFold,GridSearchCV

from sklearn.metrics import accuracy_score

import pandas as pd

import warnings

warnings.filterwarnings('ignore')”

READING THE DATA FILES:

```
numberOfFeaures=10000
trainData=validData=trainLables=validLables=np.array([])
fileNames=['arcene_train.data','arcene_valid.data']
for file in fileNames:
    with open(file) as myFile:
        data=myFile.read().split(' ')
        while '\n' in data: data.remove('\n')
        data=np.array(data).reshape(int(len(data)/numberOfFeaures),numberOfFeaures)
        if file=='arcene_train.data':
            trainData=data
        if file=='arcene_valid.data':
            validData=data
fileNames=['arcene_train.labels','arcene_valid.labels']
for file in fileNames:
     with open(file) as myFile:
        data=myFile.read().split('\n')
        while '' in data: data.remove('')
        if file=='arcene_train.labels':
             trainLables=np.array(data)
        if file=='arcene_valid.labels':
             validLables=np.array(data)
print("Train Data Shape : ",trainData.shape)
print("Train Lables Shape : ",trainLables.shape)
print("Valid Lables Shape : ",validLables.shape)
print("Valid Data Shape : ",validData.shape)
```

OUTPUT: Train Data Shape : (100, 10000)

```
Train Lables Shape :  (100,)
Valid Lables Shape :  (100,)
Valid Data Shape :  (100, 10000)
```

K-DIGIT PCA:

To perform PCA we need to standarize the data and perform 10 and 100 component pca for 1000 dataset,and also on the valid dataset

code:

```
def getPca10And100(train,test,k1,k2):
    train = preprocessing.MinMaxScaler().fit_transform(train)
    test = preprocessing.MinMaxScaler().fit_transform(test)
    scaler = StandardScaler()
    train = scaler.fit_transform(train)
    test = scaler.transform(test)
    train = preprocessing.MinMaxScaler().fit_transform(train)
    test = preprocessing.MinMaxScaler().fit_transform(test)
    pca10 = PCA(n_components=k1)
    pca10.fit(train)
    pcaTrain10 = pca10.transform(train)
    pcaTest10 = pca10.transform(test)

    pca100 = PCA(n_components=k2)
    pca100.fit(train)
    pcaTrain100 = pca100.transform(train)
    pcaTest100 = pca100.transform(test)
    return [pcaTrain10,pcaTrain100],[pcaTest10,pcaTest100]
```

PERFORM SVM,OTHER FUNCTION:

getAccuracy: Returns the accuracy by counting the number of correct outputs

code:

```
def getAccuracy(predict,actual):
    correct=0
    for i in range(actual.shape[0]):
        if predict[i]==actual[i]:
            correct+=1
    return correct/actual.shape[0]
```

metricsCM: Gives us various metrics of Confusion Matrix such as, Precision, Recall/Sensitivity, Specificity and Accuracy.

```
class metricsCM():
    def __init__(self,cm):
        self.cm=cm
        self.TP=self.getTP()
        self.TN=self.getTN()
        self.FP=self.getFP()
        self.FN=self.getFN()
        self.tot=self.TP+self.TN+self.FP+self.FN
    def errorRate(self):
        return np.around(((self.FP+self.FN)/self.tot),decimals=3)
   def accuracy(self):
        return np.around((self.TP+self.TN)/self.tot,decimals=3)
 def precision(self):
        return np.around((self.TP/(self.TP+self.FP)),decimals=3)
    def recall(self):
        return np.around((self.TP/(self.TP+self.FN)),decimals=3)
    def specificity(self):
```

```
        return np.around((self.TN/(self.FP+self.TN)),decimals=3)
    def getTP(self):
        return np.diag(self.cm)
    def getFP(self):
        FP = []
        for i in range(2):
            FP.append(sum(self.cm[:,i]) - self.cm[i,i])
            return np.array(FP)
    def getFN(self):
        FN = []
        for i in range(2):
            FN.append(sum(self.cm[i,:]) - self.cm[i,i])
        return np.array(FN)
    def getTN(self):
        TN = []
        for i in range(2):
            temp = np.delete(self.cm, i, 0)   # delete ith row
            temp = np.delete(temp, i, 1)  # delete ith column
            TN.append(sum(sum(temp)))
        return np.array(TN)
```

CrossValidate: Performs cross validation and returns Mean Error Matrix, Mean SUpport Vectors Matrix, index of PCA Data Sets and index of Classifiers at which minimum cross validation error has occured.

Code:

```
def performCrossValidation(lables,dataSets):
    numberOfFolds=5
    errorTrack=[]
    minError=99999
    minClassifierIndex=minPcaDataIndex=-1
```

```
    classifierIndex=pcaIndex=0

meanErrorOutputSamples=np.zeros(len(classifiers)*len(pcaDataSets)).reshape(len(classifiers),len(pcaDataSets))

meanOfSupportVectors=np.zeros(len(classifiers)*len(pcaDataSets)).reshape(len(classifiers),len(pcaDataSets))
    for classifier in classifiers:
        pcaIndex=0
        for pcaData in dataSets:
            kfold = KFold(numberOfFolds,False, 1)
            error=[]
            supportVectors=[]
            for train, test in kfold.split(pcaData):
                trainingSet=pcaData[train]
                testingSet=pcaData[test]
                lablesTrain=lables[train]
                lablesTest=lables[test]
                classifier.fit(trainingSet, lablesTrain)
                predicted=classifier.predict(testingSet)
                accuracy=getAccuracy(predicted,lablesTest)
                err=1-accuracy
                error.append(err)
                supportVectors.append(np.sum(classifier.n_support_))
                cm=metrics.confusion_matrix(lablesTest, predicted)
                measuresCm=measuresOfConMatrix(cm)
                acc=measuresCm.accuracy()
                spe=measuresCm.specificity()
                print("Kernel : ",classifier.kernel,", K-Digit PCA :",pcaData.shape[1])
```

```
            print("Support Vectors : ",classifier.n_support_," : Total number of support vectors : ",np.sum(classifier.n_support_))
            print("Confusion matrix:\n%s" % cm)
            print("Classification report for classifier %s:\n%s\n"
                  % (classifier, metrics.classification_report(lablesTest, predicted)))
            print("\t\tAccuracy\tSpecificity")
            print("\t-1\t","%.2f\t"%acc[0],"\t%.2f"%spe[0])
            print("\t 1\t","%.2f\t"%acc[1],"\t%.2f"%spe[1])
            print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")
        meanError=np.mean(np.array(error))
        meanSupportVectos=np.mean(np.array(supportVectors))
        meanErrorOutputSamples[classifierIndex][pcaIndex]=meanError
        meanOfSupportVectors[classifierIndex][pcaIndex]=meanSupportVectos
        print("Mean Cross Validation Error : %0.2f" % meanError)
        errorTrack.append(meanError)
        if minError > meanError:
            minError=meanError
            minClassifierIndex=classifierIndex
            minPcaDataIndex=pcaIndex
        pcaIndex+=1
        print("------------------------------------------------------------------------------")
        print("------------------------------------------------------------------------------")
        print("------------------------------------------------------------------------------")
        print("------------------------------------------------------------------------------")
        print("------------------------------------------------------------------------------")
    classifierIndex+=1
return minClassifierIndex,minPcaDataIndex,meanErrorOutputSamples,meanOfSupportVectors
```

Grid search model selection:This process is done in order to select the appropriate C and Gamma values so that the number of support vectors reduces and GridSearchCV gives the best combination of kernel,gamma and C values**.**

Code:

```
pcaDataSets,pcaValidationSets=getPca10And100(trainData,validData,10,100)
#Create a dictionary of possible parameters
params_grid = {'C': [0.001,0.005, 0.01,0.05, 0.1,0.5, 1, 5,10,50, 100,1000],
        'gamma': [0.0001, 0.001, 0.01, 0.1],
        'kernel':['linear','rbf'] }

#Create the GridSearchCV object
clf = GridSearchCV(svm.SVC(class_weight='balanced'), params_grid,cv=5)

for i in range(len(pcaDataSets)):
   #Fit the data with the best possible parameters
   clf.fit(pcaDataSets[i], trainLables)
   #Print the best estimator with it's parameters
   print(clf.best_score_)
   print(clf.best_estimator_)
   bestClassifier=clf.best_estimator_
   bestPcaData=pcaDataSets[i]
   pcaValidationSet=pcaValidationSets[i]
   bestClassifier.fit(bestPcaData, trainLables)
   predicted=bestClassifier.predict(bestPcaData)
   trainAccuracy=getAccuracy(predicted,trainLables)
   predicted=bestClassifier.predict(pcaValidationSet)
   testAccuracy=getAccuracy(predicted,validLables)
   cancerCm=metrics.confusion_matrix(validLables, predicted)
   measuresCancer=measuresOfConMatrix(cancerCm)
   cancerAcc=measuresCancer.accuracy()
   cancerSpe=measuresCancer.specificity()
   marginCount=0
   nonMarginCount=0
   for dualCoeff in (bestClassifier.dual_coef_[0]):
      if(abs(dualCoeff)<bestClassifier.C):
         marginCount+=1
      elif(abs(dualCoeff)==bestClassifier.C):
         nonMarginCount+=1

   print("Kernel : ",bestClassifier.kernel,", K-Digit PCA :",bestPcaData.shape[1])
```

```
    print("Support Vectors : ",bestClassifier.n_support_," : Total number of support vectors :
",np.sum(bestClassifier.n_support_))
    print("No. of Margin SVs : ",marginCount,", No. of Non Margin SVs : ",nonMarginCount)
    print("Train Data set Accuracy : %0.2f" % (trainAccuracy*100),"%")
    print("Validation Data set Accuracy : %0.2f" % (testAccuracy*100),"%")
    print("Confusion matrix:\n%s" % cancerCm)
    print("Classification report for classifier %s:\n%s\n"
          % (bestClassifier, metrics.classification_report(validLables, predicted)))
    print("\t\tAccuracy\tSpecificity")
    print("\t-1\t","%.2f\t"%cancerAcc[0],"\t%.2f"%cancerSpe[0])
    print("\t 1\t","%.2f\t"%cancerAcc[1],"\t%.2f"%cancerSpe[1])
    print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++")
```

Implemetation:

code:

```
classifiers=getClassifiers()

pcaDataSets,pcaValidationSets=getPca10And100(trainData,validData,10,100)

minClassifierIndex,minPcaDataIndex,meanErrorCancer,meanSupportVectorsCancer=performCrossVali
dation(trainLables,pcaDataSets)
```

training and selection with best model and implementing:

Now we choose the combination of SVM classifier with PCA dataset that has given us the minimum cross validation error in order to test the validation error.

It is observed that SVM Classifier with Linear Kernel and 100 digit PCA has given the least **cross validation error of 14%**

Below is the code that chooses the best and performs training on the entire Arcene dataset and then Tests the Validation Set.

Accuracy on Test Set : 86%

code:

```
bestClassifier=classifiers[minClassifierIndex]
```

```
bestPcaData=pcaDataSets[minPcaDataIndex]
pcaValidationSet=pcaValidationSets[minPcaDataIndex]

bestClassifier.fit(bestPcaData, trainLables)
predicted=bestClassifier.predict(bestPcaData)
trainAccuracy=getAccuracy(predicted,trainLables)
predicted=bestClassifier.predict(pcaValidationSet)
testAccuracy=getAccuracy(predicted,validLables)
cancerCm=metrics.confusion_matrix(validLables, predicted)
measuresCancer=measuresOfConMatrix(cancerCm)
cancerAcc=measuresCancer.accuracy()
cancerSpe=measuresCancer.specificity()
print(bestClassifier.dual_coef_[0].shape)
marginCount=0
nonMarginCount=0
for dualCoeff in (bestClassifier.dual_coef_[0]):
    if(abs(dualCoeff)<bestClassifier.C):
        marginCount+=1
    elif(abs(dualCoeff)==bestClassifier.C):
        nonMarginCount+=1
print("Kernel : ",bestClassifier.kernel,", K-Digit PCA :",bestPcaData.shape[1])
print("Support Vectors : ",bestClassifier.n_support_," : Total number of support vectors :
",np.sum(bestClassifier.n_support_))
print("No. of Margin SVs : ",marginCount,", No. of Non Margin SVs : ",nonMarginCount)
print("Train Data set Accuracy : %0.2f" % (trainAccuracy*100),"%")
print("Validation Data set Accuracy : %0.2f" % (testAccuracy*100),"%")
print("Confusion matrix:\n%s" % cancerCm)
print("Classification report for classifier %s:\n%s\n"
      % (bestClassifier, metrics.classification_report(validLables, predicted)))
```

```
print("\t\tAccuracy\tSpecificity")
print("\t-1\t","%.2f\t"%cancerAcc[0],"\t%.2f"%cancerSpe[0])
print("\t 1\t","%.2f\t"%cancerAcc[1],"\t%.2f"%cancerSpe[1])
print("+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")
```

GENERALISATION ERROR:

Generalization Error for Arcene Dataset:

In the below code block it is observed that the all the SVM classifiers follow satisfy the inequality for Arcene Dataset

$$E[\![OutSampleError]\!] <= E[\![NumberofSVs]\!] N-1$$

Code:

```
generalizationError(len(classifiers),len(pcaDataSets),meanErrorCancer,meanSupportVectorsCancer,pcaDataSets[0].shape[0])
```

OUTPUT:

```
E[Out_Sample Error]    <=     E[[Number of SVs]]/(N-1)
-----------------------------------------------------
|       0.34            <=       0.5191919191919192     |
-----------------------------------------------------
|       0.14            <=       0.7535353535353535     |
-----------------------------------------------------
|       0.32            <=       0.7939393939393938     |
-----------------------------------------------------
|       0.34            <=       0.8080808080808081     |
-----------------------------------------------------
```

SVM USING ANOTHER DATA SET:

Telco Customer Churn SVM

Reading Data from CSV:

Here the data is present in csv format. It is read using Pandas Library.

The dataset is divided into 80% Traing data and 20% Testing Data

CODE:

```
data = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")

df = pd.DataFrame(data)

df=pd.get_dummies(df,columns=['gender','Partner','Dependents','PhoneService','MultipleLines','InternetService','OnlineSecurity','OnlineBackup','DeviceProtection','TechSupport','StreamingTV','StreamingMovies','Contract','PaperlessBilling','PaymentMethod'])

df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

df.dropna(inplace=True)

df.replace(('Yes', 'No'), (1, -1), inplace=True)

df1 = df.drop(['Churn'], axis=1)

churnTrain=df1.iloc[:5634,1:]

churnTest=df1.iloc[5634:,1:]

churnTrainLables=df['Churn'][:5634]

churnTestLables=df['Churn'][5634:]
```

Grid Search Model Selection:

This process is done in order to select the appropriate C and Gamma values so that the number of support vectors reduces and GridSearchCV gives the best combination of kernel,gamma and C values

CODE:

```
pcaDataChurn,pcaValidationChurn=getPca10And100(churnTrain,churnTest,10,18)

#Create a dictionary of possible parameters

params_grid = {'C': [0.001, 0.01, 0.1,1,10,100],

        'gamma': [0.001, 0.01, 0.1],

        'kernel':['linear','rbf'] }


#Create the GridSearchCV object

clf = GridSearchCV(svm.SVC(class_weight='balanced'), params_grid)


for i in range(len(pcaDataChurn)):

   #Fit the data with the best possible parameters

   clf.fit(pcaDataChurn[i], churnTrainLables)
```

```
    #Print the best estimator with it's parameters
    print(clf.best_score_)
    print(clf.best_estimator_)
    bestClassifier=clf.best_estimator_
    bestPcaData=pcaDataChurn[i]
    pcaValidationSet=pcaValidationChurn[i]
    bestClassifier.fit(bestPcaData, churnTrainLables)
    predicted=bestClassifier.predict(bestPcaData)
    trainAccuracy=getAccuracy(predicted,churnTrainLables.values)
    predicted=bestClassifier.predict(pcaValidationSet)
    testAccuracy=getAccuracy(predicted,churnTestLables.values)
    cancerCm=metrics.confusion_matrix(churnTestLables, predicted)
    measuresCancer=measuresOfConMatrix(cancerCm)
    cancerAcc=measuresCancer.accuracy()
    cancerSpe=measuresCancer.specificity()
    marginCount=0
    nonMarginCount=0
    for dualCoeff in (bestClassifier.dual_coef_[0]):
        if(abs(dualCoeff)<bestClassifier.C):
            marginCount+=1
        elif(abs(dualCoeff)==bestClassifier.C):
            nonMarginCount+=1
    print("Kernel : ",bestClassifier.kernel,", K-Digit PCA :",bestPcaData.shape[1])
    print("Support Vectors : ",bestClassifier.n_support_," : Total number of support vectors :
",np.sum(bestClassifier.n_support_))
    print("No. of Margin SVs : ",marginCount,", No. of Non Margin SVs : ",nonMarginCount)
    print("Train Data set Accuracy : %0.2f" % (trainAccuracy*100),"%")
    print("Validation Data set Accuracy : %0.2f" % (testAccuracy*100),"%")
    print("Confusion matrix:\n%s" % cancerCm)
```

```
    print("Classification report for classifier %s:\n%s\n"
          % (bestClassifier, metrics.classification_report(churnTestLables, predicted)))
    print("\t\tAccuracy\tSpecificity")
    print("\t-1\t","%.2f\t"%cancerAcc[0],"\t%.2f"%cancerSpe[0])
    print("\t 1\t","%.2f\t"%cancerAcc[1],"\t%.2f"%cancerSpe[1])
    print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")
```

Cross Validation:

Performs Cross Validation and Prints the same details as done in earlier data set

CODE:

```
classifiers=getClassifiers()
pcaDataChurn,pcaValidationChurn=getPca10And100(churnTrain,churnTest,10,18)
minClassifierChurn,minPcaDataChurn,meanErrorChurn,meanSupportVectorsChurn=performCr
ossValidation(churnTrainLables.values,pcaDataChurn)
```

Training and Testing with best SVM model after Cross Validation

Minimum Cross Validation Error: **21%**

Best SVM Chosen : **Kernel - Linear, K-Digit PCA - 10**

Accuracy on Test Set : 78.40%

CODE:

```
bestClassifierChurn=classifiers[minClassifierChurn]
bestPcaDataChurn=pcaDataChurn[minPcaDataChurn]
pcaValidationSetChurn=pcaValidationChurn[minPcaDataChurn]
bestClassifierChurn.fit(bestPcaDataChurn, churnTrainLables)
predicted=bestClassifierChurn.predict(bestPcaDataChurn)
trainAccuracy=getAccuracy(predicted,churnTrainLables.values)
predicted=bestClassifierChurn.predict(pcaValidationSetChurn)
testAccuracy=getAccuracy(predicted,churnTestLables.values)


churnCm=metrics.confusion_matrix(churnTestLables, predicted)
measuresChurn=measuresOfConMatrix(churnCm)
churnAcc=measuresChurn.accuracy()
```

```
churnSpe=measuresChurn.specificity()

print("Kernel : ",bestClassifierChurn.kernel,", K-Digit PCA :",bestPcaDataChurn.shape[1])

print("Support Vectors : ",bestClassifierChurn.n_support_," : Total number of support vectors : ",np.sum(bestClassifierChurn.n_support_))

print("No. of Margin SVs : ",marginCount,", No. of Non Margin SVs : ",nonMarginCount)

print("Train Data set Accuracy : %0.2f" % (trainAccuracy*100),"%")

print("Validation Data set Accuracy : %0.2f" % (testAccuracy*100),"%")

print("Confusion matrix:\n%s" % churnCm)


print("Classification report for classifier %s:\n%s\n"

    % (bestClassifierChurn, metrics.classification_report(churnTestLables, predicted)))

print("\t\tAccuracy\tSpecificity")

print("\t-1\t","%.2f\t"%churnAcc[0],"\t%.2f"%churnSpe[0])

print("\t 1\t","%.2f\t"%churnAcc[1],"\t%.2f"%churnSpe[1])


print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++")
```

Generalization Error for TELCOM CUSTOMER CHURN Dataset:

All SVMs for Telco Customer Churn Dataset satisfy the below inequality

$$E[\![OutSampleError]\!] <= E[\![NumberofSVs]\!] N-1$$

CODE:

```
generalizationError(len(classifiers),len(pcaDataChurn),meanErrorChurn,meanSupportVectorsChurn,pcaDataChurn[0].shape[0])
```

OUTPUT:

```
E[Out_Sample Error]    <=     E[[Number of SVs]]/(N-1)
-----------------------------------------------------
|       0.21            <=       0.39080418959701757    |
-----------------------------------------------------
|       0.21            <=       0.38469731936800994    |
-----------------------------------------------------
|       0.21            <=       0.38267353097816437    |
-----------------------------------------------------
|       0.22            <=       0.3834546422865258     |
```

Observations:

1) The generalization error inequality $E[OutSampleError] <= E[NumberofSVs]N-1$ is satisfied by both the datasets

2) If the number of support vectors is the upper bound of the inequality becomes less and thus the error of out samples also gets reduced.

3) Normalization just before PCA gives better accuracy

4) In K-Didigit PCA the best results were achieved at K=100 in Arcene Cancer Data set and K=18 in Telco Custoer Churn Dataset

5) In both the model selection methods GridSearch and K-Fold Cross validation, the later has given better results of accuracy than the former. Grid Search was used to find out the best combination of C and gamma that can give better accuracy.

6) Using K-Fold Cross Validation the accyracies are

- Arcene Data Set : 86% on test set and 100% on train set
- Telco Customer Data Set : 78.33% on test set and 79.14%