STATISTICAL METHODS IN AI
ASSIGNMENT6:
SVM & Decision Trees

Name – Megha Agarwal
Roll No – **201506511**
Course – M.Tech CSIS(PG1)
Dated : 17 April, 2016

AIM:

The aim of this assignment is to experiment with Support Vector Machine (SVM) and Decision Tree (DT) techniques we learned in the class on real world problems. :

A. Support Vector Machine (SVM) Classifier

B. Decision Tree (DT) Classifier

CODE : A) *Support Vector Machines*

ARCRENE – DATA SET:

```
import csv
from random import shuffle
import numpy as np
import sklearn
from sklearn.decomposition import PCA
from sklearn import svm
from sklearn import cross_validation
from sklearn.metrics import *
from sklearn import datasets
from sklearn.metrics import confusion_matrix

name_file='arcene_train.data'
lables_name='arcene_train.labels'

with open(name_file,'rb') as file_name:
        datal=list(csv.reader(file_name,delimiter=' '))
with open(lables_name,'rb') as f:
        labels=list(csv.reader(f))

labels=np.array(labels)
labels=np.ravel(labels).astype(int)
for i in range(0,len(datal)):
        del datal[i][-1]
        for x in range(0,len(datal[i])):
                datal[i][x]=float(datal[i][x])

print "Principal Componenets Analysis : Components 10"
pca_output = PCA(n_components=10).fit(datal)
dataset_pca_out=pca_output.transform(datal)

print "*****SVM with Linear Kernels*********"
clf_op=svm.SVC(kernel='linear')

count = 1
print
print "------Fold No------", count
clf_op.fit(dataset_pca_out[:60],labels[:60])#Teseting
result=clf_op.predict(dataset_pca_out[80:])     #Prediction
conf=confusion_matrix(labels[80:], result, labels=[1, -1])
print "Confusion Matrix: "
print conf
acc=accuracy_score(labels[80:], result)

print "Accuracy: ", acc*100, "%"
```

```
prec=precision_score(labels[80:], result)
print "Precision Score:", prec
recall=recall_score(labels[80:], result)
print "Recall Score:", recall
count+=1
print "-----------------------"


print
print "------Fold No------", count
clf_op.fit(dataset_pca_out[:60],labels[:60])#Teseting
result=clf_op.predict(dataset_pca_out[0:20])   #Prediction
conf=confusion_matrix(labels[0:20], result, labels=[1, -1])
print "Confusion Matrix: "
print conf
acc=accuracy_score(labels[0:20], result)

print "Accuracy: ", acc*100, "%"

prec=precision_score(labels[0:20], result)
print "Precision Score:", prec
recall=recall_score(labels[0:20], result)
print "Recall Score:", recall
count+=1
print "-----------------------"



print
print "------Fold No------", count
clf_op.fit(dataset_pca_out[40:],labels[40:])#Teseting
result=clf_op.predict(dataset_pca_out[20:40]) #Prediction
conf=confusion_matrix(labels[20:40], result, labels=[1, -1])
print "Confusion Matrix: "
print conf
acc=accuracy_score(labels[20:40], result)

print "Accuracy: ", acc*100, "%"

prec=precision_score(labels[20:40], result)
print "Precision Score:", prec
recall=recall_score(labels[20:40], result)
print "Recall Score:", recall
count+=1
print "-----------------------"



print
print "------Fold No------", count
```

```
clf_op.fit(dataset_pca_out[40:],labels[40:])#Teseting
result=clf_op.predict(dataset_pca_out[40:60]) #Prediction
conf=confusion_matrix(labels[40:60], result, labels=[1, -1])
print "Confusion Matrix: "
print conf
acc=accuracy_score(labels[40:60], result)

print "Accuracy: ", acc*100, "%"

prec=precision_score(labels[40:60], result)
print "Precision Score:", prec
recall=recall_score(labels[40:60], result)
print "Recall Score:", recall
count+=1
print "------------------------"



print
print "------Fold No------", count
clf_op.fit(dataset_pca_out[30:90],labels[30:90])#Teseting
result=clf_op.predict(dataset_pca_out[60:80]) #Prediction
conf=confusion_matrix(labels[60:80], result, labels=[1, -1])
print "Confusion Matrix: "
print conf
acc=accuracy_score(labels[60:80], result)

print "Accuracy: ", acc*100, "%"

prec=precision_score(labels[60:80], result)
print "Precision Score:", prec
recall=recall_score(labels[60:80], result)
print "Recall Score:", recall
count+=1
print "------------------------"
```

IRIS – DATA SET:

Data Loading:

```
datairis=datasets.load_iris()
datal=datairis.data
labels=datairis.target
print "Principal Componenets Analysis : Components 2"
pca_output = PCA(n_components=2).fit(datal)
dataset_pca_out=pca_output.transform(datal)
```

Further cross fold is properly applied.

SAMPLE OUTPUT:

Arcrene Data Set :

Principal Components : 10, "LINEAR":

```
Principal Componenets Analysis : Components 10
*****SVM with Linear Kernels*********

------Fold No------ 1
Confusion Matrix:
[[7 4]
 [3 6]]
Accuracy:  65.0 %
Precision Score: 0.7
Recall Score: 0.636363636364
------------------------

------Fold No------ 2
Confusion Matrix:
[[ 4  3]
 [ 3 10]]
Accuracy:  70.0 %
Precision Score: 0.571428571429
Recall Score: 0.571428571429
------------------------

------Fold No------ 3
Confusion Matrix:
[[5 3]
 [6 6]]
Accuracy:  55.0 %
Precision Score: 0.454545454545
Recall Score: 0.625
------------------------

------Fold No------ 4
Confusion Matrix:
[[7 2]
 [6 5]]
Accuracy:  60.0 %
Precision Score: 0.538461538462
Recall Score: 0.777777777778
------------------------

------Fold No------ 5
Confusion Matrix:
[[5 4]
```

[4 7]]
Accuracy: 60.0 %
Precision Score: 0.555555555556
Recall Score: 0.555555555556

Principal Components : 100,   "LINEAR":

Principal Componenets Analysis : Components 100
*****SVM with Linear Kernels*********

------Fold No------ 1
Confusion Matrix:
[[7 4]
 [0 9]]
Accuracy: 80.0 %
Precision Score: 1.0
Recall Score: 0.636363636364

------Fold No------ 2
Confusion Matrix:
[[7 0]
 [0 13]]
Accuracy: 100.0 %
Precision Score: 1.0
Recall Score: 1.0

------Fold No------ 3
Confusion Matrix:
[[7 1]
 [3 9]]
Accuracy: 80.0 %
Precision Score: 0.7
Recall Score: 0.875

------Fold No------ 4
Confusion Matrix:
[[9 0]
 [0 11]]
Accuracy: 100.0 %
Precision Score: 1.0
Recall Score: 1.0

------Fold No------ 5
Confusion Matrix:
[[9 0]
 [0 11]]
Accuracy: 100.0 %
Precision Score: 1.0
Recall Score: 1.0

Principal Components : 10, "RBF - Kernel":

 Principal Componenets Analysis : Components 10
*****SVM with RBF Kernels*********

------Fold No------ 1
Confusion Matrix:
[[0 11]
 [0 9]]
Accuracy: 45.0 %

------Fold No------ 2
Confusion Matrix:
[[0 7]
 [0 13]]
Accuracy: 65.0 %

------Fold No------ 3
Confusion Matrix:
[[8 0]
 [0 12]]
Accuracy: 100.0 %

------Fold No------ 4
Confusion Matrix:
[[9 0]
 [0 11]]
Accuracy: 100.0 %

------Fold No------ 5
Confusion Matrix:
[[9 0]
 [0 11]]

Accuracy: 100.0 %
Precision Score:

Principal Components : 100,   "RBF-Kernel":

Principal Componenets Analysis : Components 100
*****SVM with RBF Kernels*********

------Fold No------ 1
Confusion Matrix:
[[0 11]
 [0 9]]
Accuracy: 45.0 %

------Fold No------ 2
Confusion Matrix:
[[0 7]
 [0 13]]
Accuracy: 65.0 %

------Fold No------ 3
Confusion Matrix:
[[8 0]
 [0 12]]
Accuracy: 100.0 %

------Fold No------ 4
Confusion Matrix:
[[9 0]
 [0 11]]
Accuracy: 100.0 %

------Fold No------ 5
Confusion Matrix:
[[9 0]
 [0 11]]
Accuracy: 100.0 %

IRIS Data Set :

Principal Components : 2, “LINEAR”:

*****SVM with Linear Kernels : IRIS DATA SET*********

```
------Fold No------ 1
Confusion Matrix:
[[30  0  0]
 [ 0  0  0]
 [ 0  0  0]]
Accuracy:  100.0 %
Precision Score: 0.0
Recall Score: 0.0
-----------------------

------Fold No------ 2
Confusion Matrix:
[[20  0  0]
 [ 0 10  0]
 [ 0  0  0]]
Accuracy:  100.0 %
Precision Score: 1.0
Recall Score: 1.0
-----------------------

------Fold No------ 3
Confusion Matrix:
[[ 0  0  0]
 [ 0 30  0]
 [ 0  0  0]]
Accuracy:  100.0 %
Precision Score: 1.0
Recall Score: 1.0
-----------------------

------Fold No------ 4
Confusion Matrix:
[[ 0  0  0]
 [ 0 10  0]
 [ 0  5 15]]
Accuracy:  83.3333333333 %
Precision Score: 0.666666666667
Recall Score: 1.0
-----------------------

------Fold No------ 5
Confusion Matrix:
[[ 0  0  0]
```

[0 0 0]
 [0 11 19]]
Accuracy: 63.3333333333 %

Principal Components : 2,   "RBF-Kernel":

------Fold No------ 1
Confusion Matrix:
[[30 0 0]
 [0 0 0]
 [0 0 0]]
Accuracy: 100.0 %
Precision Score: 0.0
Recall Score: 0.0

------Fold No------ 2
Confusion Matrix:
[[20 0 0]
 [0 10 0]
 [0 0 0]]
Accuracy: 100.0 %
Precision Score: 1.0
Recall Score: 1.0

------Fold No------ 3
Confusion Matrix:
[[0 0 0]
 [0 30 0]
 [0 0 0]]
Accuracy: 100.0 %
Precision Score: 1.0
Recall Score: 1.0

------Fold No------ 4
Confusion Matrix:
[[0 0 0]
 [0 10 0]
 [0 6 14]]
Accuracy: 80.0 %
Precision Score: 0.625
Recall Score: 1.0

------Fold No------ 5
Confusion Matrix:
```
[[ 0  0  0]
 [ 0  0  0]
 [ 0 11 19]]
```
Accuracy: 63.3333333333 %

CODE : B) *DECISION TREES*

Hayes-Roth Data Set :

```
import random
import operator
from math import *
def split(data, axis, val):
    newData = []
    for i in range(len(data)):
        if data[i][axis] != val:
            pass
        else:
            reducedFeat = data[i][:axis]
            temp = axis
            temp = temp +1
            reducedFeat.extend(data[i][temp:])
            newData.append(reducedFeat)
    return newData

def  lodadata(file_name):
    dataset = open(file_name,"r")
    data = []
    lines = dataset.readlines()
    data_with_normalisation = []
    for i in range(len(lines)):
        data.append(lines[i].split(","))
    temp = []
    for x in range(0,len(data)):
        len_data = len(data[x])
        for y in range(0,len_data):
            if y > 0:
                t = float(data[x][y])
                data[x][y]=t
                temp.append(t)
        data_with_normalisation.append(temp)
        temp = []
    return data_with_normalisation

def majority(classList):
    dictionary_class_count={}

    for i in range(len(classList)):
        if classList[i] in dictionary_class_count.keys():
            pass
        else:
            dictionary_class_count[classList[i]] = 0
        temp = dictionary_class_count[classList[i]]
        temp = temp+1
```

```
        dictionary_class_count[classList[i]] = temp
            value   =   sorted(dictionary_class_count.iteritems(),key=operator.itemgetter(1),
reverse=True)[0][0]
    return value

def entropy(data):
    labels={}
    count=0
    entropy = float(0)
    for i in range(len(data)):
        label=data[i][-1]
        count+=1
        if label in labels.keys():
            pass
        else:
            t = 0
            labels[label] = 0
        t+=1
        labels[label]=labels[label]+ 1

    for key in labels:
        length = len(data)
        value = float(labels[key])
        temp = value/length
        entropy=entropy-temp* log(temp,2)
    return entropy

def choose(data):
    temp = data[0]
    length=len(temp)
    bestInfoGain =  0.0
    features = length - 1
    baseEntropy = entropy(data)
    bestFeat = -1
    newEntropy = 0.0
    for i in range(features):
        uniqueVals = set([ex[i] for ex in data])
        newData_list = []
        for value in uniqueVals:
            newData = split(data, i, value)
            length = len(newData)
            length_data = len(data)
            length_data = float(length_data)
            temp = newEntropy
            newEntropy=(length/length_data) * entropy(newData)
            newEntropy=newEntropy+temp
            newData_list = []
        compare = baseEntropy - newEntropy
        if(bestInfoGain<compare):
            info = baseEntropy - newEntropy
```

```
            bestFeat = i
            bestInfoGain = info

        newEntropy = 0.0
    return bestFeat

def tree(data,labels):
    classList=[]
    for ex in data:
        classList.append(ex[-1])
    length = len(data[0])
    if length == 1:
        return majority(classList)
    if classList.count(classList[0]) == len(classList):
        return classList[0]

    bestFeat = choose(data)
    theTree = {labels[choose(data)]:{}}
    bestFeatLabel = labels[choose(data)]
    del(labels[bestFeat])
    uniqueVals = set([ex[bestFeat] for ex in data])
    for value in uniqueVals:
        split_value = split(data, bestFeat, value)
        theTree[bestFeatLabel][value] = tree(split_value, labels[:])
    return theTree

def cal_accuracy(test_data,dtree, correct):
    len_test = len(test_data)
    for index in xrange(len_test):
        temp = dtree
        while isinstance(temp,dict):
            key = temp.keys()
            t, temp_key_0 = key[0], key[0]
            temp_key=t-1
            value = test_data[index][t-1]
            temp = temp[temp_key_0]
            if value not in temp:
                ina = temp.keys()[0]
                temp = ina
            else:
                temp = temp[value]
        val = test_data[index][-1]
        if temp != val:
            pass
        else:
            t = correct
            t = t + 1
            correct = t
    ret = correct*100
    ret = float(ret)
```

```
    ret = ret/len_test
    return ret


def call_5_fold(data,index_start,index_end,total_accuracy,len_data_cf):
    training_data = []
    for fold in xrange(5):
        training_data=[]
        print
        print "******Fold No ", fold , "******"
        temp = index_end
        index_end+=len_data_cf
        index_start = temp
        test_data=[]
        for i in range(index_start, index_end):
            test_data.append(data[i])
        if index_start > 0:
            length = len(training_data)
            training_data.extend(data[0:index_start])
            temp2 = index_end
        temp2 = index_end
        if len(data) > temp2 :
            length = len(data)
            training_data.extend(data[index_end:length])

        dtree=tree(training_data,[1,2,3,4])
        accuracy=cal_accuracy(test_data,dtree,0)
        print dtree
        print 'Accuracy',
        print accuracy
        total_accuracy=total_accuracy+ accuracy
    return total_accuracy

if __name__ == '__main__':
    data=lodadata("hayes-roth.data")
    random.shuffle(data)
    length = len(data)
    length = length/5
    total_accuracy=call_5_fold(data,0,0,0,length)
    print "Average Accuracy= ",
    total_accuracy = total_accuracy/5
    print total_accuracy
    print
```

SAMPLE OUTPUT:

Hayes-Roth Data Set :

******Fold No 1 ******
{2: {1.0: {4: {1.0: 1.0, 2.0: {3: {1.0: 1.0, 2.0: 2.0, 3.0: {1: {1.0: 2.0, 2.0: 2.0, 3.0: 1.0}}, 4.0: 3.0}}, 3.0: {3: {1.0: 1.0, 3.0: 1.0, 4.0: 3.0}}, 4.0: 3.0}}, 2.0: {3: {1.0: {4: {1.0: 1.0, 2.0: 2.0, 3.0: {1: {1.0: 1.0, 2.0: 1.0, 3.0: 2.0}}, 4.0: 3.0}}, 2.0: 2.0, 3.0: {4: {2.0: 2.0, 3.0: 2.0, 4.0: 3.0}}, 4.0: 3.0}}, 3.0: {3: {1.0: 1.0, 2.0: {1: {1.0: {4: {1.0: 1.0, 2.0: 2.0}}, 2.0: 2.0, 3.0: {4: {1.0: 1.0, 2.0: 2.0}}}}, 3.0: 1.0}}, 4.0: 3.0}}
Accuracy 46.1538461538

******Fold No 2 ******
{3: {1.0: {2: {1.0: 1.0, 2.0: {4: {1.0: 1.0, 2.0: 2.0, 3.0: {1: {1.0: 2.0, 2.0: 1.0, 3.0: 1.0}}, 4.0: 3.0}}, 3.0: {4: {1.0: 1.0, 3.0: 1.0, 4.0: 3.0}}, 4.0: 3.0}}, 2.0: {4: {1.0: {2: {1.0: 1.0, 2.0: 2.0, 3.0: {1: {1.0: 1.0, 2.0: 1.0, 3.0: 2.0}}}}, 2.0: {1: {1.0: 2.0, 2.0: 3.0, 3.0: 2.0}}, 3.0: {2: {2.0: 2.0, 3.0: 2.0, 4.0: 3.0}}, 4.0: 3.0}}, 3.0: {4: {1.0: {1: {2.0: 3.0, 3.0: 1.0}}, 2.0: {2: {1.0: {1: {1.0: 2.0, 2.0: 1.0, 3.0: 1.0}}, 2.0: 2.0, 3.0: 2.0, 4.0: 3.0}}, 3.0: 2.0, 4.0: 3.0}}, 4.0: 3.0}}
Accuracy 53.8461538462

******Fold No 3 ******
{2: {1.0: {4: {1.0: {3: {2.0: 1.0, 3.0: 1.0, 4.0: 3.0}}, 2.0: {3: {1.0: 1.0, 2.0: 2.0, 3.0: {1: {1.0: 1.0, 2.0: 2.0, 3.0: 1.0}}, 4.0: 3.0}}, 3.0: {3: {3.0: 1.0, 4.0: 3.0}}, 4.0: 3.0}}, 2.0: {3: {1.0: {4: {1.0: 1.0, 2.0: 2.0, 3.0: {1: {1.0: 2.0, 2.0: 1.0, 3.0: 1.0}}, 4.0: 3.0}}, 2.0: {4: {1.0: 2.0, 3.0: 2.0, 4.0: 3.0}}, 3.0: 2.0, 4.0: 3.0}}, 3.0: {3: {1.0: {4: {1.0: 1.0, 3.0: 1.0, 4.0: 3.0}}, 2.0: {4: {1.0: {1: {1.0: 2.0, 2.0: 1.0, 3.0: 2.0}}, 2.0: 2.0, 3.0: 2.0, 4.0: 3.0}}, 3.0: {4: {1.0: 1.0, 2.0: 2.0}}, 4.0: 3.0}}, 4.0: 3.0}}
Accuracy 73.0769230769

******Fold No 4 ******
{4: {1.0: {3: {1.0: {2: {2.0: 1.0, 3.0: 1.0, 4.0: 3.0}}, 2.0: {2: {1.0: 1.0, 2.0: 2.0, 3.0: {1: {1.0: 1.0, 2.0: 1.0, 3.0: 2.0}}, 4.0: 3.0}}, 3.0: 1.0, 4.0: 3.0}}, 2.0: {2: {1.0: {3: {1.0: 1.0, 2.0: 2.0, 3.0: {1: {1.0: 1.0, 2.0: 2.0, 3.0: 1.0}}, 4.0: 3.0}}, 2.0: {3: {1.0: 2.0, 3.0: 2.0, 4.0: 3.0}}, 3.0: {3: {3.0: 2.0, 4.0: 3.0}}, 4.0: 3.0}}, 3.0: {2: {1.0: {3: {1.0: 1.0, 3.0: 1.0, 4.0: 3.0}}, 2.0: {3: {1.0: {1: {1.0: 2.0, 2.0: 1.0, 3.0: 1.0}}, 3.0: 2.0, 4.0: 3.0}}, 3.0: 2.0, 4.0: 3.0}}, 4.0: 3.0}}
Accuracy 76.9230769231

******Fold No 5 ******
{4: {1.0: {3: {1.0: {2: {2.0: 1.0, 4.0: 3.0}}, 2.0: {2: {1.0: 1.0, 2.0: 2.0, 3.0: {1: {1.0: 1.0, 2.0: 1.0, 3.0: 2.0}}, 4.0: 3.0}}, 3.0: {2: {1.0: 1.0, 3.0: 1.0, 4.0: 3.0}}, 4.0: 3.0}}, 2.0: {3: {1.0: {2: {1.0: 1.0, 2.0: 2.0}}, 2.0: {2: {1.0: 2.0, 3.0: 2.0, 4.0: 3.0}}, 3.0: {2: {1.0: {1: {1.0: 1.0, 2.0: 2.0, 3.0: 1.0}}, 3.0: 2.0}}, 4.0: 3.0}}, 3.0: {3: {1.0: {2: {1.0: 1.0, 2.0: {1: {1.0: 2.0, 2.0: 1.0, 3.0: 1.0}}, 3.0: 1.0}}, 2.0: {2: {2.0: 2.0, 4.0: 3.0}}, 3.0: 1.0, 4.0: 3.0}}, 4.0: 3.0}}
Accuracy 65.3846153846
Average Accuracy= 63.0769230769