SMAI Assignment 1

Mitesh Shah - 20172010

Problem 1:

Q1) Implement Voted Perceptron:

Here is the complete code for the voted perceptron

# 20172010_q1_1.py

```
# @Author - Mitesh Shah
# @Roll Number - 20172010
import numpy as np
import csv
import sys

def readIonoSphere():
	f = open("ionosphere.csv","r")
	dataraw = list(csv.reader(f))
	y = []
	temp = map(float,dataraw[0][:-1])
	x = np.array(temp)
	if dataraw[0][-1].strip() == 'b':
		y.append(1)
	else:
		y.append(-1)
	for i in xrange(1,len(dataraw)):
		temp = map(float,dataraw[i][:-1])
		x = np.vstack((x,np.array(temp,dtype=float)))
		if dataraw[i][-1].strip() == 'b':
			y.append(1)
		else:
			y.append(-1)
	y = np.array(y)
	return x,y

def readBreastCancer():
	f = open("breast-cancer-wisconsin.csv","r")
	dataraw1 = list(csv.reader(f))
	dataraw = []
	# Remove the '?' Values
	for row in dataraw1:
		if "?" in row:
			Continue
		else:
			dataraw.append(row)
	"""
	# To Store the "?" Removed Data
	f = open("newbcd.csv","w")
	for row in dataraw:
		f.write(','.join(row)+"\n")
	f.close()
	"""
	y = []
	temp = map(float,dataraw[0][1:-1])
	x = np.array(temp)
	if dataraw[0][-1] == '2':
		y.append(1)
	else:
		y.apppend(-1)
	for i in xrange(1,len(dataraw)):
		temp = map(float,dataraw[i][1:-1])
		x = np.vstack((x,np.array(temp,dtype=float)))
		if dataraw[i][-1] == '2':
			y.append(1)
		else:
			y.append(-1)
	y = np.array(y)
	return x,y

def VotedPerceptron(x,y,epochs):
	resvectors = []
	w = np.zeros(len(x[0]))
	b = 0.0
	c = 0
	for j in xrange(epochs):
		for i in xrange(len(x)):
			if (np.dot(w,x[i])+b)*y[i] <= 0:
				t = (w,b,c)
				resvectors.append(t)
				w = w + y[i]*x[i]
				b = b + y[i]
				c = 1
			else:
				c += 1
		t = (w,b,c)	#Last tuple was not Appended
		resvectors.append(t)
	return resvectors

if __name__ == "__main__":
	print "Select your Dataset."
	print "Enter 1. for Ionosphere Dataset"
	print "Enter 2. for Breast Cancer Dataset"
	choice = input("Enter your choice: ")
	if choice == 1:
		traindata,trainlabels = readIonoSphere()
	elif choice == 2:
		traindata,trainlabels = readBreastCancer()
	else:
		print "Wrong Choice Entered!"
		sys.exit(1)
	epochs = input("Enter the Number of Epochs: ")
	resTuples = VotedPerceptron(traindata,trainlabels,epochs)
```

```
        print "Perceptron Trained."
        print len(resTuples)
```

Output:

```
Select your Dataset.
Enter 1. for Ionosphere Dataset
Enter 2. for Breast Cancer Dataset
Enter your choice: 1
Enter the Number of Epochs: 10
Perceptron Trained.
558
```

Q2) Read the UCI Datasets:

Here is the complete code. Here we make the following assumptions:

1. In Ionosphere data, we take 'b' as +1 class and 'g' as -1 class.
2. In the Breast Cancer data, we take '2' as the +1 class and '4' as -1 class.
3. In the Breast Cancer data, we remove the first column (which is the sample id) because it isn't required for classification.

The rows containing '?' are removed from the Breast Cancer Dataset. There were 699 rows initially and 16 contained '?' and thus the final set contains 683 rows.

```
#20172010_q1_2.py
def readIonoSphere():
        f = open("ionosphere.csv","r")
        dataraw = list(csv.reader(f))
        y = []
        temp = map(float,dataraw[0][:-1])
        x = np.array(temp)
        if dataraw[0][-1].strip() == 'b':
                y.append(1)
        else:
                y.append(-1)
        for i in xrange(1,len(dataraw)):
                temp = map(float,dataraw[i][:-1])
                x = np.vstack((x,np.array(temp,dtype=float)))
                if dataraw[i][-1].strip() == 'b':
                        y.append(1)
                else:
                        y.append(-1)
        y = np.array(y)
        return x,y

def readBreastCancer():
        f = open("breast-cancer-wisconsin.csv","r")
        dataraw1 = list(csv.reader(f))
        dataraw = []
        # Remove the '?' Values
```

```
    for row in dataraw1:
        if "?" in row:
            continue
        else:
            dataraw.append(row)
    """
    # To Store the "?" Removed Data
    f = open("newbcd.csv","w")
    for row in dataraw:
        f.write(','.join(row)+"\n")
    f.close()
    """
    y = []
    temp = map(float,dataraw[0][1:-1])
    x = np.array(temp)
    if dataraw[0][-1] == '2':
        y.append(1)
    else:
        y.apppend(-1)
    for i in xrange(1,len(dataraw)):
        temp = map(float,dataraw[i][1:-1])
        x = np.vstack((x,np.array(temp,dtype=float)))
        if dataraw[i][-1] == '2':
            y.append(1)
        else:
            y.append(-1)
    y = np.array(y)
    return x,y
```

Output (for the Breast Cancer Dataset):

```
Data:
[[ 5.  1.  1. ...  3.  1.  1.]
 [ 5.  4.  4. ...  3.  2.  1.]
 [ 3.  1.  1. ...  3.  1.  1.]
 ...
 [ 5. 10. 10. ...  8. 10.  2.]
 [ 4.  8.  6. ... 10.  6.  1.]
 [ 4.  8.  8. ... 10.  4.  1.]]
683 Rows
```

Q3) Run the Voted Perceptron and Vanilla Perceptron on Both Datasets:

Here is the complete code:

```
# 20172010_q1_3.py
# @Author - Mitesh Shah
# @Roll Number - 20172010
import numpy as np
import csv
import sys

def readIonoSphere():
	f = open("ionosphere.csv","r")
	dataraw = list(csv.reader(f))
	y = []
	temp = map(float,dataraw[0][:-1])
	x = np.array(temp)
	if dataraw[0][-1].strip() == 'b':
		y.append(1)
	else:
		y.append(-1)
	for i in xrange(1,len(dataraw)):
		temp = map(float,dataraw[i][:-1])
		x = np.vstack((x,np.array(temp,dtype=float)))
		if dataraw[i][-1].strip() == 'b':
			y.append(1)
		else:
			y.append(-1)
	y = np.array(y)
	return x,y

def readBreastCancer():
	f = open("breast-cancer-wisconsin.csv","r")
	dataraw1 = list(csv.reader(f))
	dataraw = []
	# Remove the '?' Values
	for row in dataraw1:
		if "?" in row:
			continue
		else:
			dataraw.append(row)
	"""
	# To Store the "?" Removed Data
	f = open("newbcd.csv","w")
	for row in dataraw:
		f.write(','.join(row)+"\n")
	f.close()
	"""
	y = []
	temp = map(float,dataraw[0][1:-1])
	x = np.array(temp)
	if dataraw[0][-1] == '2':
		y.append(1)
	else:
		y.apppend(-1)
	for i in xrange(1,len(dataraw)):
		temp = map(float,dataraw[i][1:-1])
		x = np.vstack((x,np.array(temp,dtype=float)))
		if dataraw[i][-1] == '2':
			y.append(1)
		else:
```

```
                y.append(-1)
        y = np.array(y)
        return x,y

def VotedPerceptron(x,y,epochs):
        resvectors = []
        w = np.zeros(len(x[0]))
        b = 0.0
        c = 0
        for j in xrange(epochs):
                for i in xrange(len(x)):
                        if (np.dot(w,x[i])+b)*y[i] <= 0:
                                t = (w,b,c)
                                resvectors.append(t)
                                w = w + y[i]*x[i]
                                b = b + y[i]
                                c = 1
                        else:
                                c += 1
                t = (w,b,c)       #Last Tuple not appended to the result
                resvectors.append(t)
        return resvectors

def VanillaPerceptron(x,y,epochs):
        w = np.zeros(len(x[0]))
        b = 0.0
        for j in xrange(epochs):
                for i in xrange(len(x)):
                        if (np.dot(w,x[i])+b)*y[i] <= 0:
                                w = w + y[i]*x[i]
                                b = b + y[i]
                        else:
                                continue
        return w,b

if __name__ == "__main__":
        print "Select your Dataset."
        print "Enter 1. for Ionosphere Dataset"
        print "Enter 2. for Breast Cancer Dataset"
        choice = input("Enter your choice: ")
        if choice == 1:
                traindata,trainlabels = readIonoSphere()
        elif choice == 2:
                traindata,trainlabels = readBreastCancer()
        else:
                print "Wrong Choice Entered!"
                sys.exit(1)
        print "Enter 1. for Vanilla Perceptron."
        print "Enter 2. for Voted Perceptron."
        choice1 = input("Enter your choice: ")
        epochs = input("Enter the Number of Epochs: ")
        if choice1 == 1:
                w,b = VanillaPerceptron(traindata,trainlabels,epochs)
```

```
            print "Perceptron Trained."
            print "Weight Vector:\n",w
            print "Bias Value:\n",b
      else:
            resTuples = 
VotedPerceptron(traindata,trainlabels,epochs)
            print "Perceptron Trained."
            print resTuples
```

Output:

(Vanilla Perceptron)

```
Select your Dataset.
Enter 1. for Ionosphere Dataset
Enter 2. for Breast Cancer Dataset
Enter your choice: 1
Enter 1. for Vanilla Perceptron.
Enter 2. for Voted Perceptron.
Enter your choice: 1
Enter the Number of Epochs: 1
Perceptron Trained.
Weight Vector:
[-5.        0.       -5.0664   -3.71917 -5.64762 -2.383    -2.36511 -
1.71709
  0.47418  0.34674  2.36411 -0.14239  0.37059 -3.43505 -2.12339
0.20832
 -0.83621 -2.86692 -0.297   -1.13098 -0.32262  4.22784 -2.30984 -
1.57868
 -3.783   -4.52635  3.34429  0.3641  -1.82399 -2.30116 -2.10731
1.35383
  1.54326  1.34888]
Bias Value:
9.0
```

(Voted Perceptron)

```
Select your Dataset.
Enter 1. for Ionosphere Dataset
Enter 2. for Breast Cancer Dataset
Enter your choice: 1
Enter 1. for Vanilla Perceptron.
Enter 2. for Voted Perceptron.
Enter your choice: 2
Enter the Number of Epochs: 1
Perceptron Trained.
[(array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0.,

       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0.]), 0.0, 0), (array([-1.      ,  0.      , -0.99539,  0.05889, -
0.85243, -0.02306, -0.83398,  0.37708, -1.      , -0.0376 , -0.85243,
0.17755, -0.59755,  0.44945, -0.60536,  0.38223, -0.84356,  0.38542,
-0.58212,  0.32192, -0.56971,  0.29674, -0.36946,  0.47357, -
0.56811,  0.51171, -0.41078,  0.46168, -0.21266,  0.3409 , -0.42267,
0.54487, -0.18641,  0.453  ]), -1.0, 1) …
… (array([-5.      ,  0.      , -5.0664 , -3.71917, -5.64762, -2.383
,  -2.36511, -1.71709,  0.47418,  0.34674,  2.36411, -0.14239,
0.37059, -3.43505, -2.12339,  0.20832, -0.83621, -2.86692, -0.297  ,
-1.13098, -0.32262,  4.22784, -2.30984, -1.57868, -3.783  , -
4.52635,  3.34429,  0.3641 , -1.82399, -2.30116, -2.10731,  1.35383,
1.54326,  1.34888]), 9.0, 106)]
```

Q4) Compare Cross Validation Accuracies of Vanilla Perceptron and Voted Perceptron on both Datasets:

Here is the complete code:

```
# 20172010_q1_4.py
# @Author - Mitesh Shah
# @Roll Number - 20172010
import numpy as np
import csv
import sys
from random import seed
from random import randrange
import matplotlib.pyplot as plt

def readIonoSphere():
	f = open("ionosphere.csv","r")
	dataraw = list(csv.reader(f))
	"""
	y = []
	temp = map(float,dataraw[0][:-1])
	x = np.array(temp)
	if dataraw[0][-1].strip() == 'b':
		y.append(1)
	else:
		y.append(-1)
	for i in xrange(1,len(dataraw)):
		temp = map(float,dataraw[i][:-1])
		x = np.vstack((x,np.array(temp,dtype=float)))
		if dataraw[i][-1].strip() == 'b':
			y.append(1)
		else:
			y.append(-1)
	y = np.array(y)
	return x,y
	"""
	return dataraw

def readBreastCancer():
	f = open("breast-cancer-wisconsin.csv","r")
	dataraw1 = list(csv.reader(f))
	dataraw = []
	# Remove the '?' Values
	for row in dataraw1:
		if "?" in row:
			continue
		else:
			dataraw.append(row)
	return dataraw

def VotedPerceptron(x,y,epochs):
	resvectors = []
	w = np.zeros(len(x[0]))
	b = 0.0
	c = 0
	for j in xrange(epochs):
		for i in xrange(len(x)):
			if (np.dot(w,x[i])+b)*y[i] <= 0:
				t = (w,b,c)
				resvectors.append(t)
				w = w + y[i]*x[i]
				b = b + y[i]
				c = 1
			else:
				c += 1
		t = (w,b,c)
		resvectors.append(t)
	return resvectors

def VanillaPerceptron(x,y,epochs):
	w = np.zeros(len(x[0]))
	b = 0.0
	for j in xrange(epochs):
		for i in xrange(len(x)):
			if (np.dot(w,x[i])+b)*y[i] <= 0:
				w = w + y[i]*x[i]
				b = b + y[i]
			else:
				continue
	return w,b

def k_fold_cross_validation(dataset,folds=10):		#Default
Value of 10 Fold
	#seed(1)			# Can comment this line to get
different data everytime
	dataset_split = []
	dataset_copy = list(dataset)
	fold_size = len(dataset)//folds  #Using Integer Division
	for i in range(folds):
		fold = []
		while len(fold)<fold_size:
			index = randrange(len(dataset_copy))
			fold.append(dataset_copy.pop(index))
		dataset_split.append(fold)
	return dataset_split
```

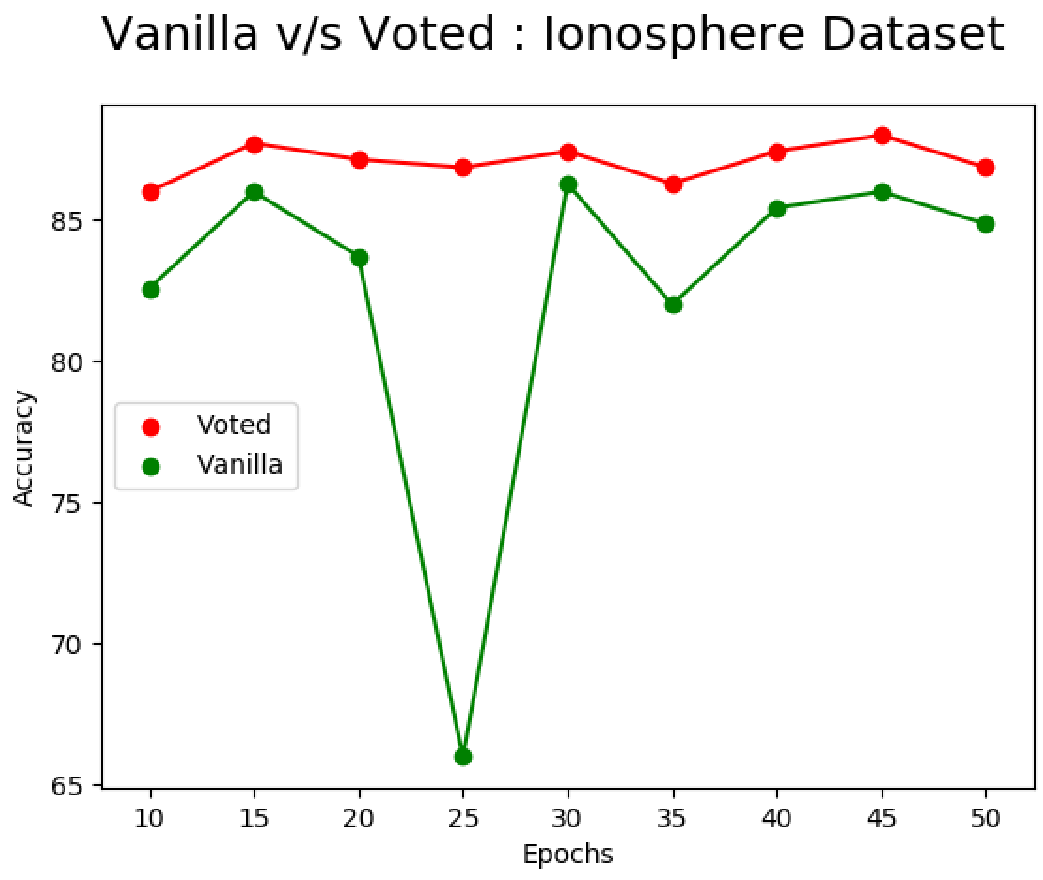
```python
def sign(y):
	if y >= 0:
		return 1
	else:
		return -1

def testVoted(x,y,resTuples):
	total = len(x)
	miss = 0
	for i in xrange(len(x)):
		val = 0
		for ele in resTuples:
			val += ele[2]*sign(np.dot(ele[0],x[i])+ele[1])
		if sign(val) != y[i]:
			miss += 1
		else:
			continue
	correct = total - miss
	accuracy = float(correct)/total
	return accuracy

def testVanilla(x,y,w,b):
	total = len(x)
	miss = 0
	for i in xrange(len(x)):
		if sign(np.dot(w,x[i])+b) != y[i]:
			miss += 1
		else:
			continue
	correct = total - miss
	accuracy = float(correct)/total
	return accuracy

if __name__ == "__main__":
	Num_Epochs = range(10,55,5)
	Vanilla_Acc = []
	Voted_Acc = []
	x = readIonoSphere()  # Can change the dataset here.
	for epoch in Num_Epochs:
		print "Epochs : ",epoch
		validatedSet = k_fold_cross_validation(x)
		votedScore = []
		vanillaScore = []
		for fold in validatedSet:
			training = list(validatedSet)
			training.remove(fold)
			# Generate Training Data and Labels
			training_data = []
			training_labels = []
			for data in training:
				for row in data:
					training_data.append(map(float,row[:-
1]))
```

```
                    if row[-1].strip() == 'b':
                        training_labels.append(1)
                    else:
                        training_labels.append(-1)
            training_data = np.array(training_data)
            training_labels = np.array(training_labels)
            # Generate Testing Data and Labels
            testing_data = []
            testing_labels = []
            for row in fold:
                testing_data.append(map(float,row[:-1]))
                if row[-1].strip() == 'b':
                    testing_labels.append(1)
                else:
                    testing_labels.append(-1)
            resTuples =
VotedPerceptron(training_data,training_labels,epoch)

    votedScore.append(testVoted(testing_data,testing_labels,resTu
ples))
            w,b =
VanillaPerceptron(training_data,training_labels,epoch)

    vanillaScore.append(testVanilla(testing_data,testing_labels,w
,b))
        print "Voted Perceptron Validation Scores: "
        for ele in votedScore:
            print "%0.2f" %(ele),
        print
        print "Average Accuracy in Voted Perceptron: ",
        avgVot = (float(sum(votedScore))/len(votedScore)) * 100
        print "%0.2f" %(avgVot)
        print "Vanilla Perceptron Validation Scores: "
        for ele in vanillaScore:
            print "%0.2f" %(ele),
        print
        print "Average Accuracy in Vanilla Perceptron: ",
        avgVan = (float(sum(vanillaScore))/len(vanillaScore)) *
100
        print "%0.2f" %(avgVan)
        Voted_Acc.append(avgVot)
        Vanilla_Acc.append(avgVan)
        print "-----------------------------------------------------
------------"
    # Plotting the Data
    v1 = plt.scatter(Num_Epochs,Voted_Acc,c = "red",label =
"Voted")
    v2 = plt.scatter(Num_Epochs,Vanilla_Acc,c = "green",label =
"Vanilla")
    v1 = plt.plot(Num_Epochs,Voted_Acc,c = "red")
    v2 = plt.plot(Num_Epochs,Vanilla_Acc,c = "green")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend(loc = "center left")
    plt.suptitle("Vanilla v/s Voted : Ionosphere Dataset",
fontsize = 18)
    plt.show()
```

Output (Ionosphere Dataset):

```
Epochs :  10
Voted Perceptron Validation Scores:
0.86 0.94 0.71 0.86 0.80 0.80 0.94 0.94 0.86 0.89
Average Accuracy in Voted Perceptron:  86.00
Vanilla Perceptron Validation Scores:
0.86 0.89 0.63 0.83 0.80 0.77 0.89 0.89 0.83 0.89
Average Accuracy in Vanilla Perceptron:  82.57
------------------------------------------------------------
Epochs :  15
Voted Perceptron Validation Scores:
0.94 0.89 0.89 0.94 0.83 0.86 0.91 0.89 0.71 0.91
Average Accuracy in Voted Perceptron:  87.71
Vanilla Perceptron Validation Scores:
0.91 0.86 0.86 0.97 0.80 0.86 0.89 0.89 0.69 0.89
Average Accuracy in Vanilla Perceptron:  86.00
------------------------------------------------------------
Epochs :  20
Voted Perceptron Validation Scores:
0.91 0.94 0.86 0.89 0.91 0.86 0.91 0.86 0.80 0.77
Average Accuracy in Voted Perceptron:  87.14
Vanilla Perceptron Validation Scores:
0.91 1.00 0.77 0.89 0.74 0.86 0.89 0.83 0.69 0.80
Average Accuracy in Vanilla Perceptron:  83.71
------------------------------------------------------------
Epochs :  25
Voted Perceptron Validation Scores:
0.91 0.86 0.89 0.86 0.83 0.89 0.80 0.89 0.94 0.83
Average Accuracy in Voted Perceptron:  86.86
Vanilla Perceptron Validation Scores:
0.51 0.74 0.49 0.54 0.80 0.83 0.77 0.51 0.57 0.83
Average Accuracy in Vanilla Perceptron:  66.00
------------------------------------------------------------
Epochs :  30
Voted Perceptron Validation Scores:
0.77 0.89 0.77 0.97 0.80 0.86 0.83 0.94 0.94 0.97
Average Accuracy in Voted Perceptron:  87.43
Vanilla Perceptron Validation Scores:
0.80 0.86 0.77 0.97 0.80 0.83 0.80 0.94 0.89 0.97
Average Accuracy in Vanilla Perceptron:  86.29
------------------------------------------------------------
Epochs :  35
Voted Perceptron Validation Scores:
0.80 0.86 0.89 0.91 0.74 0.94 0.86 0.91 0.83 0.89
Average Accuracy in Voted Perceptron:  86.29
Vanilla Perceptron Validation Scores:
0.80 0.71 0.83 0.74 0.80 0.89 0.86 0.86 0.86 0.86
Average Accuracy in Vanilla Perceptron:  82.00
------------------------------------------------------------
Epochs :  40
```

```
Voted Perceptron Validation Scores:
0.83 0.97 0.77 0.77 0.77 0.91 0.94 0.91 0.94 0.91
Average Accuracy in Voted Perceptron:  87.43
Vanilla Perceptron Validation Scores:
0.86 1.00 0.77 0.69 0.74 0.91 0.94 0.86 0.86 0.91
Average Accuracy in Vanilla Perceptron:  85.43
------------------------------------------------------------
Epochs :  45
Voted Perceptron Validation Scores:
0.89 0.86 0.91 0.89 0.86 0.86 0.91 0.97 0.89 0.77
Average Accuracy in Voted Perceptron:  88.00
Vanilla Perceptron Validation Scores:
0.86 0.86 0.94 0.83 0.83 0.86 0.86 0.97 0.86 0.74
Average Accuracy in Vanilla Perceptron:  86.00
------------------------------------------------------------
Epochs :  50
Voted Perceptron Validation Scores:
0.83 0.80 0.80 0.86 0.91 0.83 0.91 0.91 0.91 0.91
Average Accuracy in Voted Perceptron:  86.86
Vanilla Perceptron Validation Scores:
0.86 0.83 0.80 0.77 0.94 0.71 0.91 0.89 0.89 0.89
Average Accuracy in Vanilla Perceptron:  84.86
------------------------------------------------------------
```

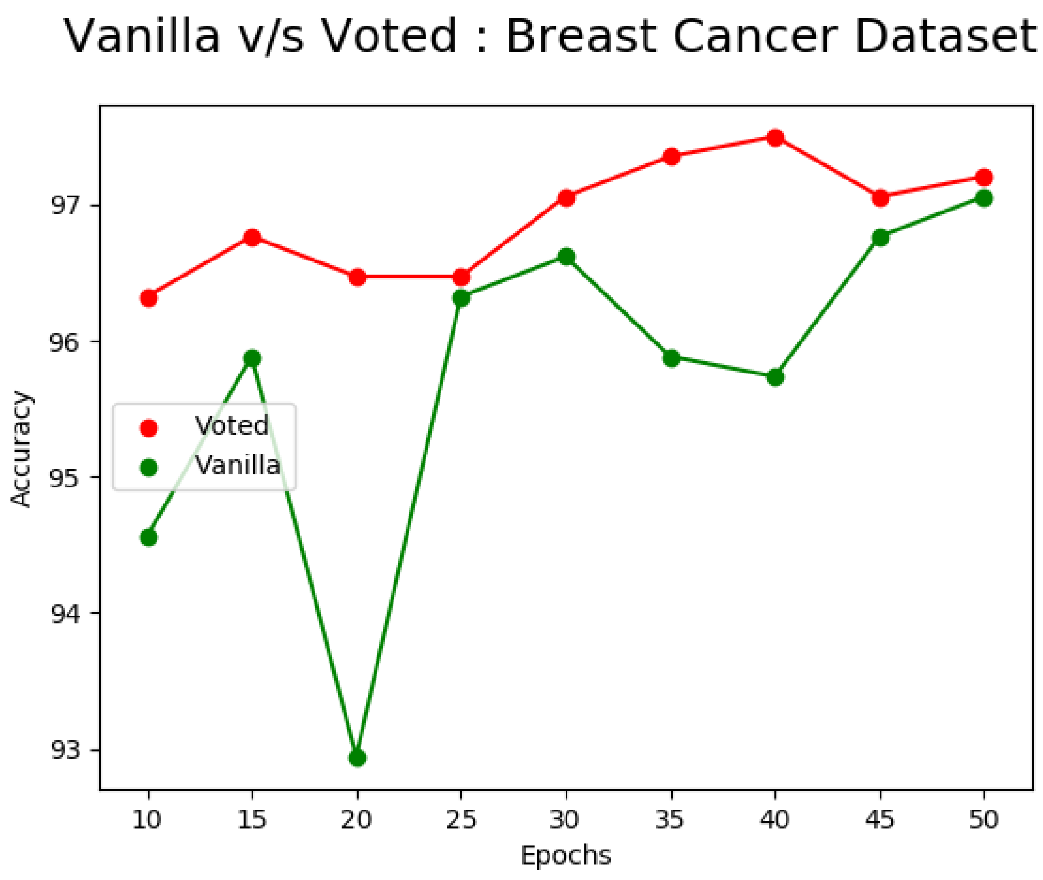


Output (Breast Cancer Dataset):

```
Epochs :  10
Voted Perceptron Validation Scores:
0.96 0.97 0.91 0.99 0.99 0.97 0.97 0.96 0.97 0.96
```

```
Average Accuracy in Voted Perceptron:  96.32
Vanilla Perceptron Validation Scores:
0.88 0.96 0.93 1.00 0.94 0.97 0.96 0.97 0.94 0.91
Average Accuracy in Vanilla Perceptron:  94.56
------------------------------------------------------------
Epochs :  15
Voted Perceptron Validation Scores:
0.99 0.94 0.97 0.99 0.93 0.96 0.99 0.94 1.00 0.99
Average Accuracy in Voted Perceptron:  96.76
Vanilla Perceptron Validation Scores:
0.99 0.91 0.93 1.00 0.91 0.94 0.99 0.94 1.00 0.99
Average Accuracy in Vanilla Perceptron:  95.88
------------------------------------------------------------
Epochs :  20
Voted Perceptron Validation Scores:
0.96 0.94 1.00 0.99 0.99 0.94 0.97 0.96 0.93 0.99
Average Accuracy in Voted Perceptron:  96.47
Vanilla Perceptron Validation Scores:
0.91 0.94 0.99 0.99 0.99 0.93 1.00 0.96 0.94 0.66
Average Accuracy in Vanilla Perceptron:  92.94
------------------------------------------------------------
Epochs :  25
Voted Perceptron Validation Scores:
0.94 0.97 0.94 0.97 0.97 0.91 1.00 1.00 0.97 0.97
Average Accuracy in Voted Perceptron:  96.47
Vanilla Perceptron Validation Scores:
1.00 0.94 0.96 0.96 0.99 0.91 0.99 0.97 0.97 0.96
Average Accuracy in Vanilla Perceptron:  96.32
------------------------------------------------------------
Epochs :  30
Voted Perceptron Validation Scores:
0.94 0.96 1.00 0.97 0.94 1.00 0.96 1.00 0.97 0.97
Average Accuracy in Voted Perceptron:  97.06
Vanilla Perceptron Validation Scores:
0.93 0.94 0.97 0.99 0.94 1.00 0.94 1.00 0.99 0.97
Average Accuracy in Vanilla Perceptron:  96.62
------------------------------------------------------------
Epochs :  35
Voted Perceptron Validation Scores:
0.99 0.97 1.00 0.96 0.97 0.96 0.99 1.00 0.96 0.96
Average Accuracy in Voted Perceptron:  97.35
Vanilla Perceptron Validation Scores:
0.94 0.91 0.96 0.97 0.97 0.96 0.99 0.97 0.96 0.97
Average Accuracy in Vanilla Perceptron:  95.88
------------------------------------------------------------
Epochs :  40
Voted Perceptron Validation Scores:
0.99 0.97 0.96 0.99 0.94 0.97 1.00 0.99 0.99 0.97
Average Accuracy in Voted Perceptron:  97.50
Vanilla Perceptron Validation Scores:
0.99 0.96 0.94 0.93 0.93 0.97 0.96 0.96 0.99 0.97
Average Accuracy in Vanilla Perceptron:  95.74
------------------------------------------------------------
Epochs :  45
Voted Perceptron Validation Scores:
1.00 0.96 0.94 1.00 0.96 0.99 1.00 0.96 0.97 0.94
```

```
Average Accuracy in Voted Perceptron:  97.06
Vanilla Perceptron Validation Scores:
1.00 0.94 0.97 1.00 0.94 0.97 0.99 0.96 0.97 0.94
Average Accuracy in Vanilla Perceptron:  96.76
------------------------------------------------------------
Epochs :  50
Voted Perceptron Validation Scores:
0.99 0.97 0.97 0.93 0.96 0.99 0.99 0.99 0.99 0.97
Average Accuracy in Voted Perceptron:  97.21
Vanilla Perceptron Validation Scores:
0.99 0.99 0.97 0.93 0.96 0.97 0.97 0.99 1.00 0.96
Average Accuracy in Vanilla Perceptron:  97.06
------------------------------------------------------------
```



<u>Comment on Both Approaches:</u>

As evident from the graphs, Voted Perceptron inherently performs better than the Vanilla Perceptron for both the datasets and gives a better accuracy. Thus voting gives better accuracy than vanilla, although storing the extra (w,b,c) values uses some extra storage and takes time to predict the class when their number is very large!

Problem 2:

Q1) Implement Linear Classifier using Least Square Approach:

Here is the complete code for the linear classifier using least square approach. (Used data of table 1 as sample data)

```
#20172010_q2_1.py
import numpy as np
from TranDataQ2 import td,labels  #Contains the Datasets of Table1
y = np.array(labels)
for i in range(len(td)):
      td[i].append(1)           #Creating Augmented Training Vector
x = np.array(td)
#x contains training data
#y contains the labels
xt = np.matrix(x).getT()    #Calculates X^T
xm = np.matrix(x)      #Converts X to Matrix
a = xt*xm       #Calculates X^TX
a = a.getI()          #Calculates (X^TX)^-1
a = a*xt              #Calculates (X^TX)^-1X
yt = np.matrix(y).getT()    #Calculates Y^T
w = a*yt        #Calculates w = (X^TX)^-1XY^T
print w
```

Output:

```
[[ 0.375     ]
 [ 0.33423913]
 [-0.57880435]]
```

Q2) Implement Linear Classifier using Fisher’s Linear Discriminant:

Here is the complete code for the linear classification using Fisher’s Linear Discriminant. (Used data of table 1 as sample data)

```
#20172010_q2_2.py
import numpy as np
import matplotlib.pyplot as plt

def vanillaPerceptron(epochs,x,y):
      # x is the training data
      # y is labels
      w = np.zeros(len(x[0]))
      b = 0
```

```
	for i in xrange(epochs):
		for j in xrange(len(x)):
			if (np.dot(w,x[j])+b)*y[j] <= 0:
				w = w + y[j]*x[j]
				b = b + y[j]
			else:
				continue
	return w,b

class1 = [[3,3],[3,0],[2,1],[0,2]]
class2 = [[-1,1],[0,0],[-1,-1],[1,0]]
m1 = np.mean(class1,axis=0)
m2 = np.mean(class2,axis=0)
s1 = np.dot((class1-m1).T,(class1-m1))
s2 = np.dot((class2-m2).T,(class2-m2))
s = s1 + s2
w = np.dot(np.linalg.inv(s),(m1-m2))
w = w / np.linalg.norm(w)
x1 = []
y1 = []
x2 = []
y2 = []
for val in class1:
	x1.append(val[0])
	y1.append(val[1])
for val in class2:
	x2.append(val[0])
	y2.append(val[1])
plt.scatter(x1,y1,c="red",label="Class 1")
plt.scatter(x2,y2,c="green",label="Class 2")
# Projecting the Points on the Line
c1d1 = np.dot(class1,w)
projClass1 = []
for value in c1d1:
	projClass1.append(np.dot(value,w))
c2d2 = np.dot(class2,w)
projClass2 = []
for value in c2d2:
	projClass2.append(np.dot(value,w))
# Plotting the Discriminant Line
slope = w[1]/w[0]
plt.plot([-2,4],[-2*slope,4*slope],color='k',linestyle='-
',linewidth=1)
# Plotting the Projected Points and Projections
px1 = []
py1 = []
px2 = []
py2 = []
for pt in projClass1:
	px1.append(pt[0])
	py1.append(pt[1])
for pt in projClass2:
	px2.append(pt[0])
	py2.append(pt[1])
plt.plot(px1,py1,"rx",label="C1 Projection")
plt.plot(px2,py2,"gx",label="C2 Projection")
for i in range(len(px1)):
	plt.plot([x1[i],px1[i]],[y1[i],py1[i]],"r--")
	plt.plot([x2[i],px2[i]],[y2[i],py2[i]],"g--")
# Preparing Data for Perceptron
traindata = []
labels = []
for row in projClass1:
	traindata.append(row)
	labels.append(1)
for row in projClass2:
	traindata.append(row)
	labels.append(-1)
# Classify using Perceptron
wp,b = vanillaPerceptron(10,traindata,labels)
print wp
# Plotting the Classifier Line
xc = range(-2,4)
yc = [(x*(-wp[0]/wp[1])-(b/wp[1])) for x in xc]
plt.plot(xc,yc,"c--",label="Classifier")
plt.legend(loc="best")
plt.show()
```

<u>Output:</u>

```
[2.10797998 1.87885173]
```

Q3) Plot Data Points on Table 1 and find Classifiers using both approaches:

Here is the complete code for the comparison of both the classifiers on data of table 1.

```
#20172010_q2_3.py
import numpy as np
import matplotlib.pyplot as plt
from TranDataQ2 import td,labels #Import Table 1 for Least Square
Classifier

def vanillaPerceptron(epochs,x,y):
	# x is the training data
	# y is labels
	w = np.zeros(len(x[0]))
	b = 0
	for i in xrange(epochs):
		for j in xrange(len(x)):
			if (np.dot(w,x[j])+b)*y[j] <= 0:
				w = w + y[j]*x[j]
```

```
                        b = b + y[j]
                  else:
                        continue
      return w,b

class1 = [[3,3],[3,0],[2,1],[0,2]]
class2 = [[-1,1],[0,0],[-1,-1],[1,0]]
# Least Square Classifier
y = np.array(labels)
for i in range(len(td)):
      td[i].append(1)
x = np.array(td)
xt = np.matrix(x).getT()    #Calculates X^T
xm = np.matrix(x)                 #Converts X to Matrix
a = xt*xm                          #Calculates X^T x X
a = a.getI()                      #Calculates (X^TX)^-1
a = a*xt                          #Calculates (X^TX)^-1 X
yt = np.matrix(y).getT()    #Calculates Y^T
wlsq = a*yt                                 #Calculates w = (X^TX)^-
1XY^T
# Plotting the Data Points
x1 = []
y1 = []
x2 = []
y2 = []
for val in class1:
      x1.append(val[0])
      y1.append(val[1])
for val in class2:
      x2.append(val[0])
      y2.append(val[1])
plt.scatter(x1,y1,c="red",label="Class 1")
plt.scatter(x2,y2,c="green",label="Class 2")
# Plotting the Least Square Classfier
x = []
y = []
ymax = 4
xmax = 4
x.append(0)
y.append(float(-1*wlsq[2])/wlsq[1])
y.append(0)
x.append(float(-1*wlsq[2])/wlsq[0])
x.append(xmax)
y.append(float((-1*wlsq[0]*xmax)-wlsq[2])/wlsq[1])
y.append(ymax)
x.append(float((-1*wlsq[1]*ymax)-wlsq[2])/wlsq[0])
plt.plot(x,y,c="black",label="Least Square Classifier")
# Fisher's Linear Discriminant
m1 = np.mean(class1,axis=0)
m2 = np.mean(class2,axis=0)
s1 = np.dot((class1-m1).T,(class1-m1))
s2 = np.dot((class2-m2).T,(class2-m2))
s = s1 + s2
```

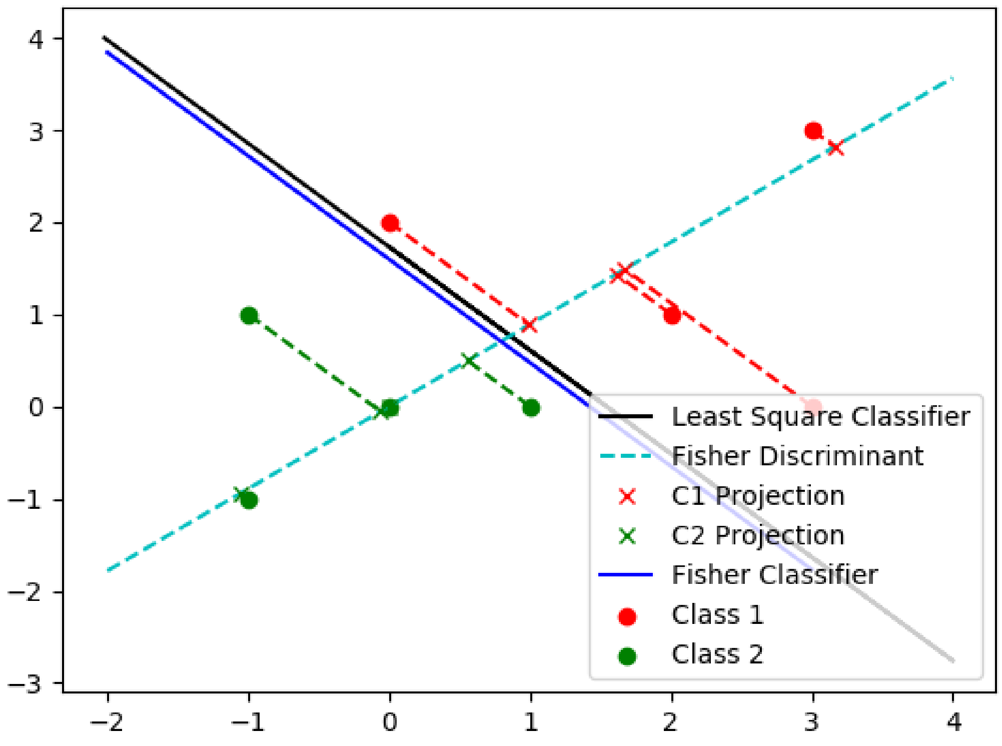
```
w = np.dot(np.linalg.inv(s),(m1-m2))
w = w / np.linalg.norm(w)
# Projecting the Points on the Line
c1d1 = np.dot(class1,w)
projClass1 = []
for value in c1d1:
      projClass1.append(np.dot(value,w))
c2d2 = np.dot(class2,w)
projClass2 = []
for value in c2d2:
      projClass2.append(np.dot(value,w))
# Plotting the Discriminant Line
slope = w[1]/w[0]
plt.plot([-2,4],[-2*slope,4*slope],"c--",label="Fisher
Discriminant")
# Plotting the Projected Points and Projections
px1 = []
py1 = []
px2 = []
py2 = []
for pt in projClass1:
      px1.append(pt[0])
      py1.append(pt[1])
for pt in projClass2:
      px2.append(pt[0])
      py2.append(pt[1])
plt.plot(px1,py1,"rx",label="C1 Projection")
plt.plot(px2,py2,"gx",label="C2 Projection")
for i in range(len(px1)):
      plt.plot([x1[i],px1[i]],[y1[i],py1[i]],"r--")
      plt.plot([x2[i],px2[i]],[y2[i],py2[i]],"g--")
# Preparing Data for Perceptron
traindata = []
labels = []
for row in projClass1:
      traindata.append(row)
      labels.append(1)
for row in projClass2:
      traindata.append(row)
      labels.append(-1)
# Classify using Perceptron
wp,b = vanillaPerceptron(10,traindata,labels)
# Plotting the Classifier Line
xc = range(-2,4)
yc = [(x*(-wp[0]/wp[1])-(b/wp[1])) for x in xc]
plt.plot(xc,yc,c="blue",label="Fisher Classifier")
plt.legend(loc="best")
plt.show()
```

For reference, here is the code in “TranDataQ2.py”

```
import numpy as np
td = [[3,3],[3,0],[2,1],[0,2],[-1,1],[0,0],[-1,-1],[1,0]]
```

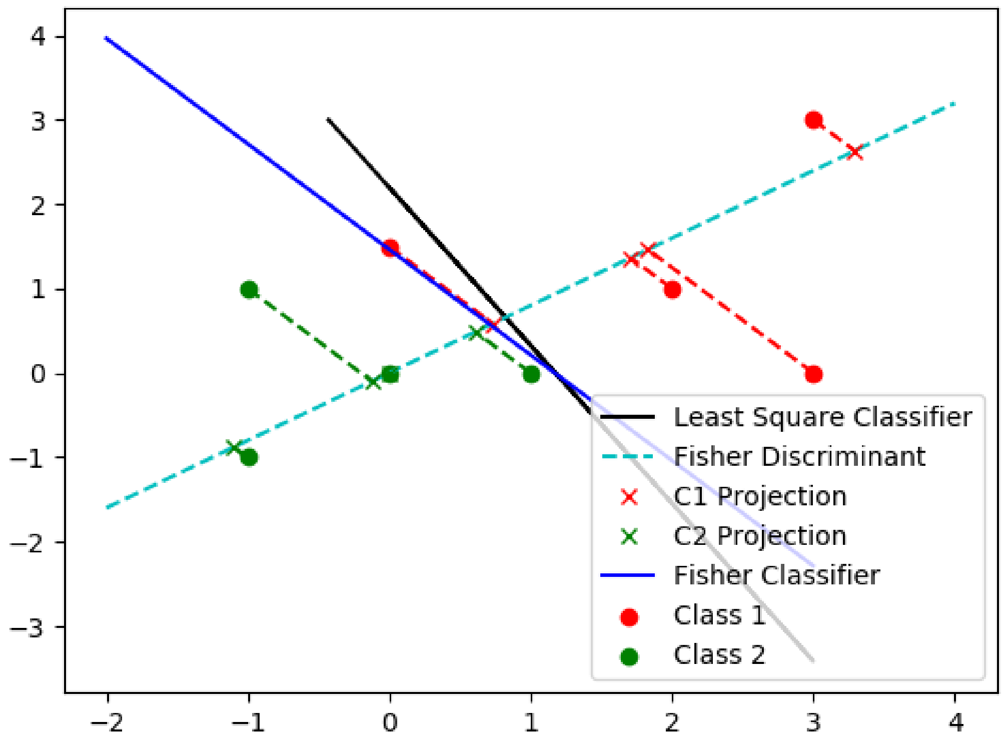
```
labels = [1,1,1,1,-1,-1,-1,-1]
```

Output:



Q4) Plot Data Points on Table 2 and find Classifiers using both approaches:

The code for this part is same as the code above (for Q3), just the data set changes. Here is the plot we get:



Q5) Comment on both the approaches:

Clearly from the above two graphs, we can see that the least square classifier is able to classify data of table 1 but can't classify data of table 2. However, with Fisher's Linear Discriminant, we are able to linearly separate both datasets. Thus it is evident from this that:

1. Least Squared is not guaranteed to classify even if the data is linearly separable as evident in table 2. It tries to minimize errors so it doesn't give priority to misclassification.
2. Fisher's Discriminant maximizes the distance between classes (and one dimensional projection was best possible separation) and thus running the perceptron algorithm on the projected data gave us a classifier that can linearly separate both classes.

Problem 3: Latent Semantic Analysis

Q2) Find the class of documents using One Vs Rest Perceptron using the TF-IDF Matrix and Dimensionality Reduction using SVD:

Here we calculate the TF-IDF Matrix after splitting the training data in 80-20% ratio, and implemented a Multiclass Perceptron. Here I didn't apply SVD as the class information is stored according to the rows of TF-IDF Matrix and thus when we re-create the matrix after applying SVD, the rows may get jumbled and thus we lose the class information. So I trained the perceptron without applying SVD and got an accuracy of about 93.45 %. (Also applying SVD on the TF-IDF Matrix ran for 1 hour and then the laptop hanged and didn't recover back!)

Here is the complete code.

```
#20172010_q3_a.py
import numpy as np
import sys
import os
import platform
import random
import math
import glob
import string

docCountTrain = 0
docCountTest = 0
wordDictTrain = {}
wordDictTest = {}
stopWords = []
classMappingTrain = {}      #Contains Class Label Mapping for
Document Numbers of Training Data
classMappingTest = {} #Contains Class Label Mapping for Document
Numbers of Testing Data
DM = ""          #Forward Slash or Training

# Checking if data directories are provided or not
if(len(sys.argv) < 3):
	print "Training/Testing Data Directories Not Provied."
	sys.exit(1)

train_directory = sys.argv[1]
test_directory = sys.argv[2]

# Getting the forward slash or backward slash depending upon OS
os = platform.system()
if os.lower().startswith('win'):
      DM = '\\'
else:
      DM = '/'

# Appending a slash added at the end if not added
if (train_directory[-1] != DM):
      train_directory += DM
if (test_directory[-1] != DM):
      test_directory += DM

# Helper Methods

# Dictionary Builder to be used later to create BOW of Training
Data
def dictBuilderTraining(content):
      global wordDictTrain,docCountTrain
      words = content.split()
      for word in words:
            word = word.lower()
            word = word.translate(None,string.punctuation)
      #Removes Punctuation Marks
            if word in stopWords:
                  pass
            elif word in wordDictTrain:
                  wordDictTrain[word].append(docCountTrain)
            else:
                  wordDictTrain[word] = [docCountTrain]
      docCountTrain = docCountTrain + 1

# Dictionary Builder to be used later to create BOW of Testing
Data
def dictBuilderTesting(content):
      global wordDictTest,docCountTest
      words = content.split()
      for word in words:
            word = word.lower()
            word = word.translate(None,string.punctuation)
      #Remove Punctuation Marks
            if word in stopWords:
                  pass
            elif word in wordDictTest:
                  temp = wordDictTest[word]
                  temp.append(docCountTest)
                  wordDictTest[word] = temp
      docCountTest = docCountTest + 1

# Create the Bag of Words using the Dictionary Structure
def createBOW(wordDict,docCount):
      keys = wordDict.keys()
      keys.sort()
      mat = np.zeros([len(keys),docCount])
      count = 0
      for key in keys:
            for dno in wordDict[key]:
                  mat[count,dno] = mat[count,dno] + 1
            count = count + 1
      return mat

# Build the TFIDF using the Bag of Words
def buildTFIDF(BOW):
      wordsPerDoc = np.sum(BOW,axis=0)
      docsPerWord = np.sum(np.asarray(BOW > 0,'i'),axis=1)
      rows, cols = BOW.shape
      for i in xrange(rows):
            for j in xrange(cols):
                  BOW[i,j] = (BOW[i,j]/wordsPerDoc[j]) * 
math.log(float(cols)/(1+docsPerWord[i]))
      TFIDF = BOW.T    # Transpose it to get into proper structure 
(i.e. docs * words)
      return TFIDF

# Find the training label associated with the document Number
def findTrainingLabel(docNumber):
      for key in classMappingTrain.keys():
            low,high = classMappingTrain[key]
            if low <= docNumber and docNumber <= high:
                  return int(key)


def findTestingLabel(docNumber):
      for key in classMappingTest.keys():
            low,high = classMappingTest[key]
            if low <= docNumber and docNumber <= high:
                  return int(key)

# Main Program

# Get the list of stopwords
f = open('.'+DM+'stopwords.txt','r')
for line in f:
      stopWords.append(line.strip())

# Generate TF-IDF for Training Data
train_dirs = glob.glob(train_directory+'*'+DM)
for i in xrange(len(train_dirs)):
      docs = glob.glob(train_dirs[i]+'*.txt')
      dname = train_dirs[i].split(DM)[-2]
      if (i==0):
            classMappingTrain[dname] = 
(docCountTrain,docCountTrain+len(docs))
      else:
            classMappingTrain[dname] = 
(docCountTrain+1,docCountTrain+len(docs))
      for doc in docs:
            desc = open(doc)
            content = desc.read()
            content = content.replace('\n','')
            dictBuilderTraining(content)

BOWTrain = createBOW(wordDictTrain,docCountTrain)
# BOW is in transpose form i.e. Rows are Words and Columns are
Document Numbers (or words * docs)
TFIDFTrain = buildTFIDF(BOWTrain)


# Can Store the TF-IDF Matrix in a file for the Training Data if
data is fixed to avoid recalculation.
f = open('TF-IDF.txt','w')
for row in TFIDFTrain:
      f.write(str(list(row)))
      f.write('\n')
f.close()


wordVocab = list(wordDictTrain.keys())      # To keep same
vocabulary for Training and Testing Data
# Create Empty Lists in Testing Dictionary Structure
for word in wordVocab:
      wordDictTest[word] = []

# Generate TF-IDF for Testing Data!
test_dirs = glob.glob(test_directory+'*'+DM)
for i in xrange(len(test_dirs)):
      docs = glob.glob(test_dirs[i]+'*.txt')
      dname = test_dirs[i].split(DM)[-2]
      if (i==0):
            classMappingTest[dname] =
(docCountTest,docCountTest+len(docs))
      else:
            classMappingTest[dname] =
(docCountTest+1,docCountTest+len(docs))
      for doc in docs:
            desc = open(doc)
            content = desc.read()
            content = content.replace('\n','')
            dictBuilderTesting(content)
BOWTest = createBOW(wordDictTest,docCountTest)
TFIDFTest = buildTFIDF(BOWTest)

print "TFIDF Matrices Created."
"""
# SVD Code Here
"""

# The Multi-Class Perceptron
classes = range(len(classMappingTrain.keys()))
noFeatures = len(TFIDFTrain[0])
noClasses = len(classes)
# Set the Epochs Here
```

```
epochs = 500    # Arbitary value got by trying different values of Epochs vs. Accuracies
w = np.zeros([noClasses,noFeatures+1]) #Weights for the Multiclass Perceptron

# Training the Multi-Class Perceptron
print "Training the Perceptron..."
for _ in xrange(epochs):
    for i in xrange(len(TFIDFTrain)):
        feature = np.hstack([TFIDFTrain[i],1])
        label = findTrainingLabel(i)
        max_act, predClass = 0,0
        for c in classes:
            curr_act = np.dot(feature,w[c])
            if curr_act > max_act:
                max_act = curr_act
                predClass = c
        if predClass != label:
            w[label] = w[label] + feature
            w[predClass] = w[predClass] - feature


# Testing Code
print "Testing the Perceptron..."
total = len(TFIDFTest)
correct = 0
for i in xrange(len(TFIDFTest)):
    test_vector = np.hstack([TFIDFTest[i],1])
    actual_class = findTestingLabel(i)
    max_act, predClass = 0,0
    for c in classes:
        curr_act = np.dot(test_vector,w[c])
        if curr_act > max_act:
            max_act = curr_act
            predClass = c
    if actual_class == predClass:
        correct = correct + 1
miss = total - correct
accuracy = (float(correct)/total)*100
print "Total Predicted: ",total
print "Correct Predicted: ",correct
print "Accuracy: %0.2f" %(accuracy)
```

Output:

```
TFIDF Matrices Created.
Training the Perceptron...
Testing the Perceptron...
Total Predicted:  352
Correct Predicted:  329
Accuracy: 93.47
```

Q3) Find the class of documents using Cosine Similarity:

Here we calculate the TF-IDF Matrix and compare the cosine similarity values of the tf-idf vectors of the testing documents and the training documents and pick out the top 10 most relevant documents. Then we find the class of those top 10 relevant documents and compare it with the actual class of the document and check if it was predicted correctly or not.

Difference from Submission Format:

TL;DR : The input is taken in the same way as for the above question.

Due to lack of time, rather than taking path of a single test document, I am taking the path of the folder containing the document in the same way as was for the above question. (i.e. test-> <class_no> -> files) and it uses the class number directory to get the class of the document being tested (and not from the command line!) Rest of the code works in normal way only.

By using this, I achieved an accuracy of 92-94%

Here is the complete code:

```
#20172010_q3_b.py
import numpy as np
import sys
import os
import platform
import random
import math
import glob
import string

docCountTrain = 0
docCountTest = 0
wordDictTrain = {}
wordDictTest = {}
stopWords = []
classMappingTrain = {}      #Contains Class Label Mapping for
Document Numbers of Training Data
classMappingTest = {} #Contains Class Label Mapping for Document
Numbers of Testing Data
DM = ""          #Forward Slash or Training

# Checking if data directories are provided or not
if(len(sys.argv) < 3):
     print "Training/Testing Data Directories Not Provied."
     sys.exit(1)

train_directory = sys.argv[1]
test_directory = sys.argv[2]

# Getting the forward slash or backward slash depending upon OS
os = platform.system()
if os.lower().startswith('win'):
     DM = '\\'
```

```
else:
      DM = '/'

# Appending a slash added at the end if not added
if (train_directory[-1] != DM):
      train_directory += DM
if (test_directory[-1] != DM):
      test_directory += DM

# Helper Methods

# Dictionary Builder to be used later to create BOW of Training
Data
def dictBuilderTraining(content):
      global wordDictTrain,docCountTrain
      words = content.split()
      for word in words:
            word = word.lower()
            word = word.translate(None,string.punctuation)
      #Removes Punctuation Marks
            if word in stopWords:
                  pass
            elif word in wordDictTrain:
                  wordDictTrain[word].append(docCountTrain)
            else:
                  wordDictTrain[word] = [docCountTrain]
      docCountTrain = docCountTrain + 1

# Dictionary Builder to be used later to create BOW of Testing
Data
def dictBuilderTesting(content):
      global wordDictTest,docCountTest
      words = content.split()
      for word in words:
            word = word.lower()
            word = word.translate(None,string.punctuation)
      #Remove Punctuation Marks
            if word in stopWords:
                  pass
            elif word in wordDictTest:
                  temp = wordDictTest[word]
                  temp.append(docCountTest)
                  wordDictTest[word] = temp
      docCountTest = docCountTest + 1

# Create the Bag of Words using the Dictionary Structure
def createBOW(wordDict,docCount):
      keys = wordDict.keys()
      keys.sort()
      mat = np.zeros([len(keys),docCount])
      count = 0
      for key in keys:
            for dno in wordDict[key]:
                    mat[count,dno] = mat[count,dno] + 1
               count = count + 1
          return mat

# Build the TFIDF using the Bag of Words
def buildTFIDF(BOW):
          wordsPerDoc = np.sum(BOW,axis=0)
          docsPerWord = np.sum(np.asarray(BOW > 0,'i'),axis=1)
          rows, cols = BOW.shape
          for i in xrange(rows):
               for j in xrange(cols):
                    BOW[i,j] = (BOW[i,j]/wordsPerDoc[j]) *
math.log(float(cols)/(1+docsPerWord[i]))
          TFIDF = BOW.T    # Transpose it to get into proper structure
(i.e. docs * words)
          return TFIDF

# Find the training label associated with the document Number
def findTrainingLabel(docNumber):
          for key in classMappingTrain.keys():
               low,high = classMappingTrain[key]
               if low <= docNumber and docNumber <= high:
                    return int(key)


def findTestingLabel(docNumber):
          for key in classMappingTest.keys():
               low,high = classMappingTest[key]
               if low <= docNumber and docNumber <= high:
                    return int(key)

# Main Program

# Get the list of stopwords
f = open('.'+DM+'stopwords.txt','r')
for line in f:
          stopWords.append(line.strip())

# Generate TF-IDF for Training Data
train_dirs = glob.glob(train_directory+'*'+DM)
for i in xrange(len(train_dirs)):
          docs = glob.glob(train_dirs[i]+'*.txt')
          dname = train_dirs[i].split(DM)[-2]
          if (i==0):
               classMappingTrain[dname] =
(docCountTrain,docCountTrain+len(docs))
          else:
               classMappingTrain[dname] =
(docCountTrain+1,docCountTrain+len(docs))
          for doc in docs:
               desc = open(doc)
               content = desc.read()
               content = content.replace('\n','')
```

```
        dictBuilderTraining(content)

BOWTrain = createBOW(wordDictTrain,docCountTrain)
# BOW is in transpose form i.e. Rows are Words and Columns are
Document Numbers (or words * docs)
TFIDFTrain = buildTFIDF(BOWTrain)

"""
# Can Store the TF-IDF Matrix in a file for the Training Data if
data is fixed to avoid recalculation.
f = open('TF-IDF.txt','w')
for row in TFIDFTrain:
    f.write(str(list(row)))
    f.write('\n')
f.close()
"""

wordVocab = list(wordDictTrain.keys())      # To keep same
vocabulary for Training and Testing Data
# Create Empty Lists in Testing Dictionary Structure
for word in wordVocab:
    wordDictTest[word] = []

# Generate TF-IDF for Testing Data!
test_dirs = glob.glob(test_directory+'*'+DM)
for i in xrange(len(test_dirs)):
    docs = glob.glob(test_dirs[i]+'*.txt')
    dname = test_dirs[i].split(DM)[-2]
    if (i==0):
        classMappingTest[dname] =
(docCountTest,docCountTest+len(docs))
    else:
        classMappingTest[dname] =
(docCountTest+1,docCountTest+len(docs))
    for doc in docs:
        desc = open(doc)
        content = desc.read()
        content = content.replace('\n','')
        dictBuilderTesting(content)
BOWTest = createBOW(wordDictTest,docCountTest)
TFIDFTest = buildTFIDF(BOWTest)

print "TFIDF Matrices Created."

classes = range(len(classMappingTrain.keys()))

# Finding the Cosine Similarity
total = len(TFIDFTest)
correct = 0
for i in xrange(len(TFIDFTest)):
    CosValues = []
    print "Predicting Row: ",i
    for j in xrange(len(TFIDFTrain)):
            val = 
np.dot(TFIDFTrain[j],TFIDFTest[i])/(np.linalg.norm(TFIDFTrain[j])*
np.linalg.norm(TFIDFTest[i]))
            # Mapping 0-1 Value to 0-100 for better comparision
            CosValues.append(val*100)
      # Finding Indices of Top 10 Scores
      indices = range(len(CosValues))
      top10 = sorted(indices,key = lambda i:CosValues[i],reverse = 
True)[:10]
      # Classes of the Top 10 Documents
      CTop10 = [findTrainingLabel(x) for x in top10]
      # Finding the Most Occuring Label
      ClassOcc = {}
      for c in classes:
            ClassOcc[c] = 0
      for ele in CTop10:
            ClassOcc[ele] += 1
      MostOcc = 0
      predClass = 0
      for k in ClassOcc:
            if MostOcc < ClassOcc[k]:
                  MostOcc = ClassOcc[k]
                  predClass = k
      actualClass = findTestingLabel(i)
      if actualClass == predClass:
            correct = correct + 1
miss = total - correct
accuracy = (float(correct)/total)*100
print "Total Predicted: ",total
print "Correctly Predicted: ",correct
print "Accuracy: %0.2f" %(accuracy)
```

Output:

```
TFIDF Matrices Created.
Predicting Row:  0
Predicting Row:  1
Predicting Row:  2
Predicting Row:  3
Predicting Row:  4
Predicting Row:  5
Predicting Row:  6
Predicting Row:  7
.
.
.
.
Predicting Row:  340
Predicting Row:  341
Predicting Row:  342
Predicting Row:  343
Predicting Row:  344
Predicting Row:  345
Predicting Row:  346
Predicting Row:  347
Predicting Row:  348
Predicting Row:  349
Predicting Row:  350
Predicting Row:  351
Total Predicted:  352
Correctly Predicted:  331
Accuracy: 94.03
```

Q4) SVD vs. Accuracy:

Since applying SVD might jumble the rows in TF-IDF matrix and I lose my class information with it, I didn’t apply SVD. Moreover when I tried applying SVD on my system, it tried running it for 1 hour and then the laptop completely hanged.