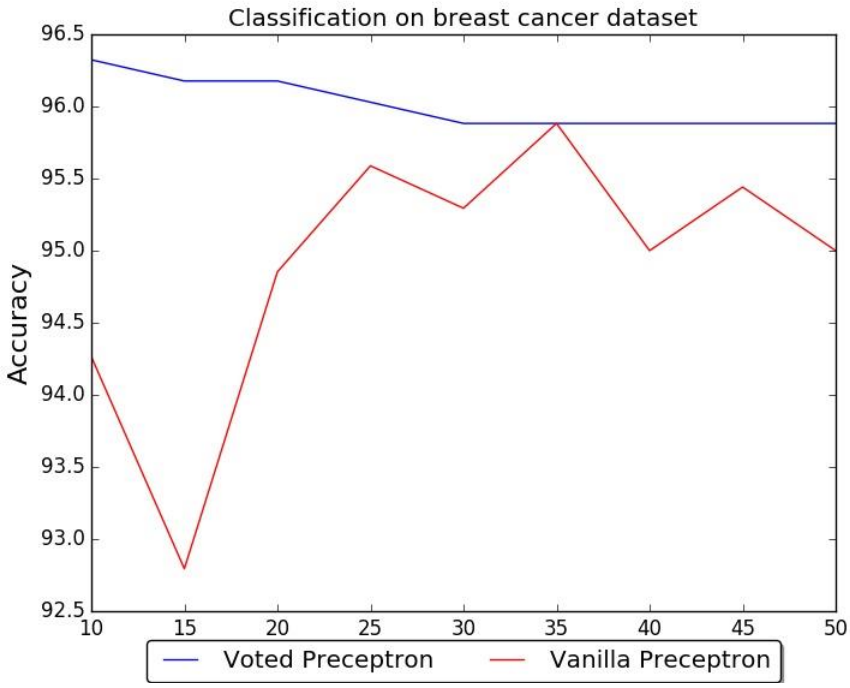
[CSE471: Statistical Methods in Artificial Intelligence (SMAI)]

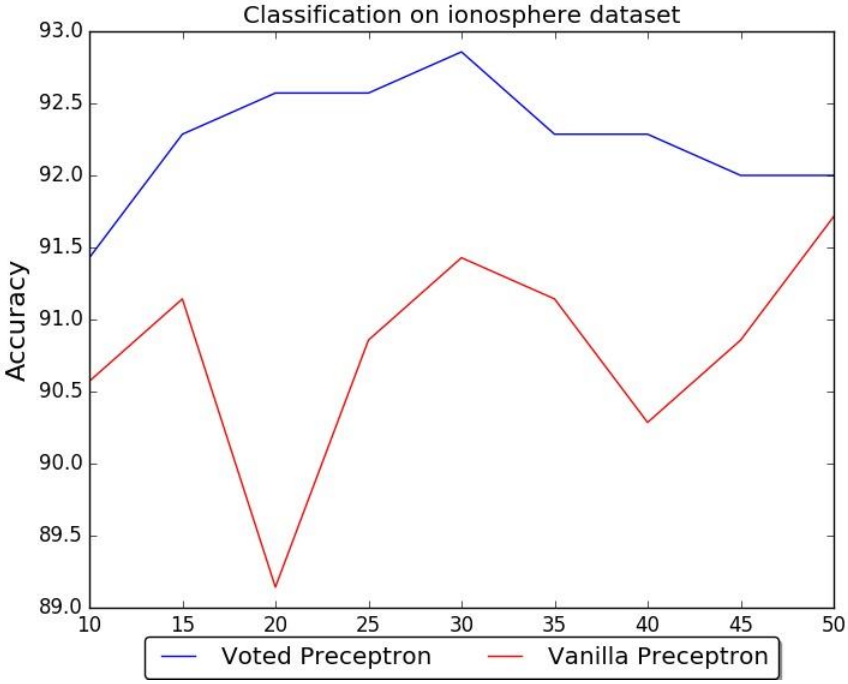
[Assignment-1]

Name: Ranajit Saha
Branch: M.Tech. CSE
Roll: 20172119

Q1: Compare the performance of Voted and Vanilla Perceptron Classification Models

Observation:





----Breast Cancer Dataset----

epoch = 10

Average Voted Perceptron Accuracy: 96.3235294118

Average Vanilla Perceptron Accuracy: 94.2647058824

epoch = 15

Average Voted Perceptron Accuracy: 96.1764705882

Average Vanilla Perceptron Accuracy: 92.7941176471

epoch = 20

Average Voted Perceptron Accuracy: 96.1764705882

Average Vanilla Perceptron Accuracy: 94.8529411765

epoch = 25

Average Voted Perceptron Accuracy: 96.0294117647

Average Vanilla Perceptron Accuracy: 95.5882352941

epoch = 30

Average Voted Perceptron Accuracy: 95.8823529412

Average Vanilla Perceptron Accuracy: 95.2941176471

epoch = 35

Average Voted Perceptron Accuracy: 95.8823529412

Average Vanilla Perceptron Accuracy: 95.8823529412

epoch = 40

Average Voted Perceptron Accuracy: 95.8823529412

Average Vanilla Perceptron Accuracy: 95.0

epoch = 45

Average Voted Perceptron Accuracy: 95.8823529412

Average Vanilla Perceptron Accuracy: 95.4411764706

epoch = 50

Average Voted Perceptron Accuracy: 95.8823529412

Average Vanilla Perceptron Accuracy: 95.0

----Ionosphere Dataset----
epoch = 10
Average Voted Perceptron Accuracy: 91.4285714286
Average Vanilla Perceptron Accuracy: 90.5714285714
epoch = 15
Average Voted Perceptron Accuracy: 92.2857142857
Average Vanilla Perceptron Accuracy: 91.1428571429
epoch = 20
Average Voted Perceptron Accuracy: 92.5714285714
Average Vanilla Perceptron Accuracy: 89.1428571429
epoch = 25
Average Voted Perceptron Accuracy: 92.5714285714
Average Vanilla Perceptron Accuracy: 90.8571428571
epoch = 30
Average Voted Perceptron Accuracy: 92.8571428571
Average Vanilla Perceptron Accuracy: 91.4285714286
epoch = 35
Average Voted Perceptron Accuracy: 92.2857142857
Average Vanilla Perceptron Accuracy: 91.1428571429
epoch = 40
Average Voted Perceptron Accuracy: 92.2857142857
Average Vanilla Perceptron Accuracy: 90.2857142857
epoch = 45
Average Voted Perceptron Accuracy: 92.0
Average Vanilla Perceptron Accuracy: 90.8571428571
epoch = 50
Average Voted Perceptron Accuracy: 92.0
Average Vanilla Perceptron Accuracy: 91.7142857143

Comments:

The accuracy of the prediction in case of voted perceptron is better than the vanilla perceptron. Also, the accuracy of the voted one is maintaining a value within very short range for different epochs, whereas the accuracy of vanilla varies a lot depending upon the epochs. We can see the accuracy of vanilla perceptron sometime dips too much, even after giving a very high accuracy for some epoch, if we increase the epoch further. But this is not the case with the voted perceptron, it is almost invariant with respect to the epochs after an epoch value. It seems the voted perceptron converges much earlier than vanilla.

Reason:

In perceptron training, we update the weights when the used weight fails to correctly classify the data point. In vanilla, whenever we find a misclassification, we update the hyperplane and train the model for other data points using the updated weight. But the problem is we don't keep track of how many examples has been correctly classified by the old hyperplane. So, when we move the hyperplane to a new position, it might be the case that the new hyperplane fails to correctly classify the points it was correctly classifying previously, and that number of misclassifications, say the set P, may be more than

the number of correctly classified examples. So, we lose the importance of the "more" correct classifier. When, we come back again for the next round of iteration, we once again have to adjust the hyperplane to make the set P to fall into the correct class. But, this way we cannot guarantee that the classifier accuracy will be improved with increased epoch, though it is guaranteed to converge at some point of time, obviously if the data is linearly classifiable.
On the other hand, the voted perceptron has the votes for each corresponding weight. This vote signifies which hyperplane classifies more examples correctly. So, when we predict the output we take the weighted value of the hyperplanes. This way the hyperplanes which were "more" correct and survives for a longer time has more contribution in predicting the output.

Code:

```python
import numpy as np
import matplotlib.pyplot as plt
from random import shuffle
from sklearn.metrics import accuracy_score

def K_Fold_Cross_Validation(X, Y, K):
    '''
    Divide the whole dataset into K-folds and
    perform testing for each one the fold
    while train the model using other K-1 folds
    '''
    Y = Y.reshape(Y.shape[0],1)
    sampleSize = int(X.shape[0] / K)
    startPoint = 0
    endPoint = startPoint + sampleSize
    X_cv = X[startPoint:endPoint,:]
    Y_cv = Y[startPoint:endPoint]
    X_train = X[0:startPoint, :]
    X_train = np.append(X_train, X[endPoint:,:],axis=0)
    Y_train = Y[0:startPoint]
    Y_train = np.append(Y_train, Y[endPoint:],axis=0)
    divided_dataset = [[X_train, X_cv, Y_train, Y_cv]]
    for i in range(K-1):
        startPoint += sampleSize
        endPoint=startPoint+sampleSize
        X_cv = X[startPoint:endPoint,:]
        Y_cv = Y[startPoint:endPoint]
        X_train = X[0:startPoint,:]
        X_train = np.append(X_train, X[endPoint:, :],axis=0)
        Y_train = Y[0:startPoint]
        Y_train = np.append(Y_train, Y[endPoint:],axis=0)
        divided_dataset = np.append(divided_dataset, np.array([[X_train, X_cv, Y_train,Y_cv]]),axis=0)
    return divided_dataset

def sign(val):
    return -1 if val < 0 else 1
```

```python
def votedPerceptron(X, Y, epoch):
    '''
    Train the voted perceptron model
    '''
    W = np.zeros((1, X.shape[1]))
    B = np.zeros((1, 1))
    C = np.zeros((1, 1))
    n = 0
    m = X.shape[0]
    number_of_features = X.shape[1]
    for round in range(epoch):
        for i in range(m):
            if Y[i] * (np.dot(W[n], X[i].reshape(number_of_features, 1)) + B[n]) <= 0:
                W = np.append(W, np.array([W[n] + Y[i] * X[i]]), axis=0)
                B = np.append(B, np.array([B[n] + Y[i]]), axis=0)
                C = np.append(C, [[1]], axis=0)
                n += 1
            else:
                C[n] += 1
    return W[1:, :], B[1:, :], C[1:, :]

def votedPerceptronPredict(X, Y, W, B, C):
    '''
    Predict the output of the voted perceptron
    using the voted weights found during training
    '''
    Y_predicted = np.zeros((1, 1))
    examples = Y.shape[0]
    shape_w = W.shape
    for e in range(examples):
        temp_val = np.dot(W, X[e].T).reshape(shape_w[0], 1) + B
        temp_val[temp_val>0] = 1
        temp_val[temp_val<0] = -1
        Y_predicted = np.append(Y_predicted, np.asarray([sign(np.dot(C.T, temp_val))]))
    Y_predicted = Y_predicted[1:]
    return Y_predicted.reshape(examples, 1)
```

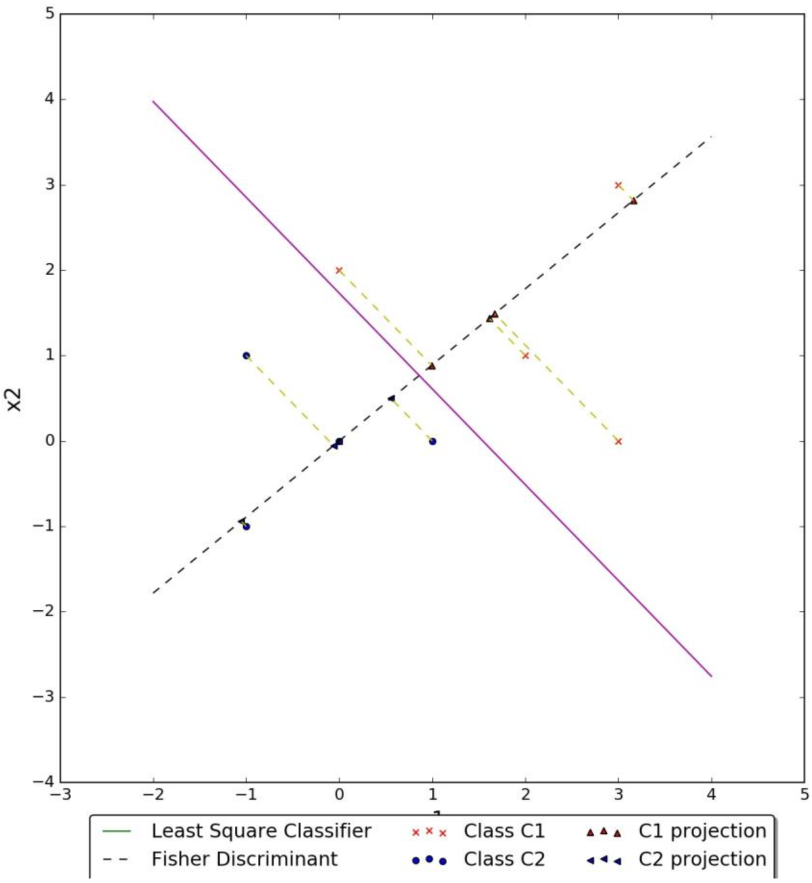
```python
def vanillaPerceptron(X, Y, epoch):
    '''
    Train the vanilla perceptron model
    '''
    bias = np.ones((X.shape[0],1))
    X = np.concatenate((X, bias),axis=1) # Append a column of 1s with X (for bias)
    number_of_features = X[0].shape[0]
    W = np.zeros((1, number_of_features))
    m = X.shape[0]
    for round in range(epoch):
        for i in range(m):
            if Y[i] * np.dot(W, X[i].reshape(number_of_features, 1)) <= 0:
                W = W + np.multiply(Y[i], X[i])
    return W

def vanillaPerceptronPredict(X, Y, W):
    '''
    Predict the output of the vanilla perceptron
    using the weights found during training
    '''
    bias = np.ones((X.shape[0],1))
    X = np.concatenate((X, bias),axis=1)
    Y_predicted = np.dot(X, W.T)
    Y_predicted[Y_predicted>0] = 1
    Y_predicted[Y_predicted<0] = -1
    return Y_predicted
```

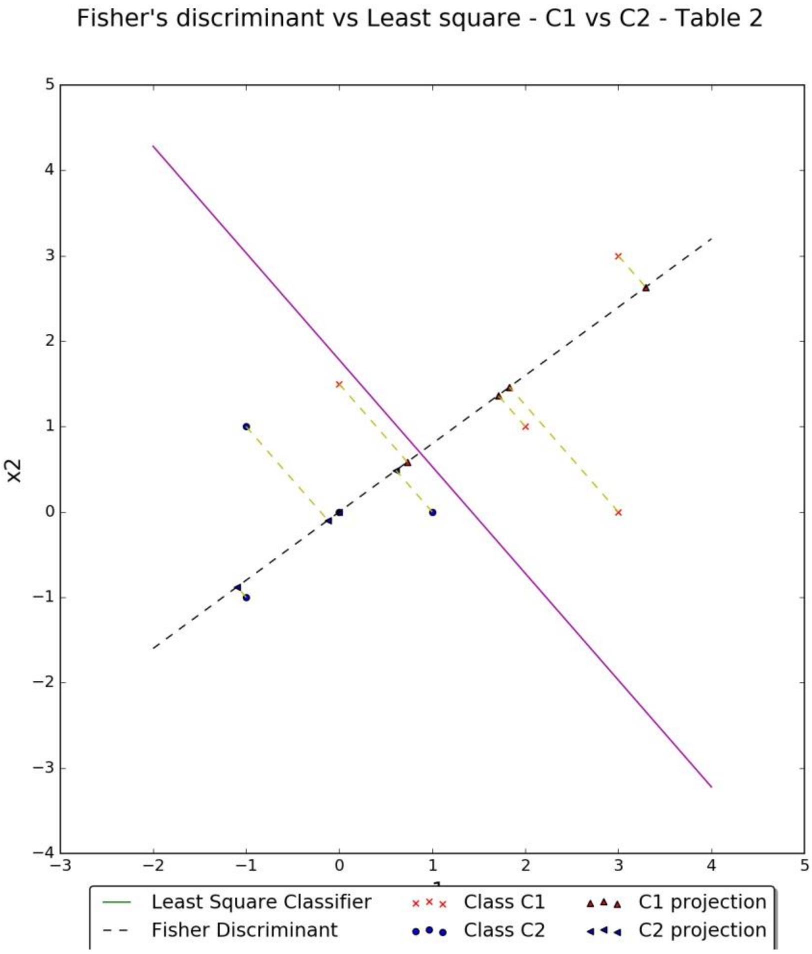
```python
def perceptron(X, Y, epochs):
    '''
    Compare two perceptron algorithms
    '''
    K = 10 # For K-fold validation
    accuracy_score_voted = []
    accuracy_score_vanilla = []
    divided_dataset = K_Fold_Cross_Validation(X,Y, K)
    for epoch in epochs:
        print("epoch = ", epoch)
        accuracy_vanilla = 0
        accuracy_voted = 0
        for ij in range(divided_dataset.shape[0]):
            X_train = divided_dataset[ij][0]
            X_cv = divided_dataset[ij][1]
            Y_train = divided_dataset[ij][2]
            Y_cv = divided_dataset[ij][3]
            W, B, C = votedPerceptron(X_train, Y_train, epoch)
            Y_predicted_Voted = votedPerceptronPredict(X_cv, Y_cv, W, B, C)
            accuracy_voted += accuracy_score(Y_cv, Y_predicted_Voted, normalize=True)
            W = vanillaPerceptron(X_train, Y_train, epoch)
            Y_predicted_Vanilla = vanillaPerceptronPredict(X_cv, Y_cv, W)
            accuracy_vanilla += accuracy_score(Y_cv, Y_predicted_Vanilla, normalize=True)
        print("Average Voted Perceptron Accuracy: ", accuracy_voted*100/K)
        print("Average Vanilla Perceptron Accuracy: ", accuracy_vanilla*100/K)
        accuracy_score_voted.append(accuracy_voted*100/K)
        accuracy_score_vanilla.append(accuracy_vanilla*100/K)
    return accuracy_score_voted, accuracy_score_vanilla
```

Q2. Comparison of classifiers – Least Square Method vs Fisher's LDA

Observation:

Fisher's discriminant vs Least square - C1 vs C2 - Table 1





Comment:

The classifiers learnt using least square method and Fisher's LDA are the same. Direction of discriminant gives the direction of projection of data points in which the separation between the classes is maximum.

Reasons:

Fisher's LDA gives the direction of projection on which the data is reduced to a lower dimension and those projected data points are efficiently separable. Then we can apply any classifier on the projected points to discriminate them. The motive of Fisher's LDA is to project the data in such a way that they are also classifiable in the projected lower dimensional space and that would give the same decision boundary. This is observed in the output.

Code:

```python
import numpy as np
import csv
import matplotlib.pyplot as plt

def LSC_binary_classification(X, Y):
    '''
    Find the weight using
    least square method
    as W = (X.T * X)^(-1) * X.T * Y
    '''
    bias = np.ones((X.shape[0],1))
    X_augmented = np.concatenate((bias, X),axis=1)
    W = np.dot(np.linalg.pinv(np.dot(X_augmented.T, X_augmented)), np.dot(X_augmented.T, Y))
    return W

def FisherLDA_binary_classification(X, Y):
    '''
    Find the direction of projection which
    maximises the class separation, the projected points
    and the weights of the classifier
    Direction of projection = S_W ^ -1 * (mean1 - mean0)
    '''
    dataset = np.concatenate((X, Y), axis=1)
    classes = np.unique(Y)
    class_mean = []
    num_of_dim = 1
    for c in classes:
        class_mean.append(((1 / len(Y[Y==c])) * np.sum(dataset[dataset[:, -1]==c], axis = 0))[:-1])
    X_class1 = dataset[dataset[:, -1]==classes[0]]
    X_class2 = dataset[dataset[:, -1]==classes[1]]
    zero_mean_X_1 = X_class1[:, :-1] - class_mean[0]
    zero_mean_X_2 = X_class2[:, :-1] - class_mean[1]
     # Using Fisher's LDA formula
    within_class_scatter = np.dot(zero_mean_X_1.T, zero_mean_X_1) + np.dot(zero_mean_X_2.T, zero_mean_X_2)
    W = np.dot(np.linalg.inv(within_class_scatter), (class_mean[1] - class_mean[0]))
    W = W / np.linalg.norm(W) # Unit vector along the direction of projection
    projected_data_scalar = np.dot(X, W) # Magnitude of the projected points
    projections = np.multiply(W.reshape(1,W.shape[0]),
        projected_data_scalar.reshape(projected_data_scalar.shape[0], 1)) # Magnitude * Unit vector gives the actual vector
    weight = LSC_binary_classification(projections, Y) # Classify the projected 1-D points using least square method
    return projections, W, weight

```

```python
def classify_LSC_LDA(X, Y, dataset_name):
    '''
    Compare two classifiers using
    1. Least square method
    2. Fisher LDA
    '''
    x_line = np.linspace(-2, 4)
    plt.figure(figsize=(10,10))
    plt.suptitle("Fisher's discriminant vs Least square - C1 vs C2 - " + dataset_name, fontsize = 18)
    plt.xlabel("x1", fontsize = 16)
    plt.ylabel("x2", fontsize = 16)
    plt.scatter(X[:4, 0], X[:4, 1], c='r', marker='x', label='Class C1')
    plt.scatter(X[4:, 0], X[4:, 1], c='b', marker='o', label='Class C2')
    # Plot Least Square Method
    LSC_weights = LSC_binary_classification(X, Y)
    boundary_LSC = -LSC_weights[0] / LSC_weights[2] - (LSC_weights[1] / LSC_weights[2]) * x_line
    plt.plot(x_line, boundary_LSC, color='g', label='Least Square Classifier')
    # Plot LDA
    projections, direction, LDA_weights = FisherLDA_binary_classification(X, Y)
    direction_line = (direction[1] / direction[0]) * x_line
    boundary_LDA = -LDA_weights[0] / LDA_weights[2] - (LDA_weights[1] / LDA_weights[2]) * x_line
    # Plot Projections
    plt.scatter(projections[:4, 0], projections[:4, 1], c='r', marker='^', label = "C1 projection")
    plt.scatter(projections[4:, 0], projections[4:, 1], c='b', marker='<', label = "C2 projection")
    for i in range(len(projections)):
        plt.plot([X[i][0], projections[i][0]], [X[i][1], projections[i][1]], 'y--')
    # Plot Discriminant and Boundary
    plt.plot(x_line, direction_line, '--', color='black', label = "Fisher Discriminant")
    plt.plot(x_line, boundary_LDA, color='m', label = "Fisher LDA")
    plt.legend(loc='upper center', bbox_to_anchor=(0.5, -0.03), fancybox=True, shadow=True, ncol=3)

    plt.savefig(dataset_name + '.jpg', dpi = 150)
    # plt.show()
    plt.clf()

#Table 1
X = np.array([[3, 3], [3, 0], [2, 1], [0, 2], [-1, 1], [0, 0], [-1, -1], [1, 0]])
Y = np.array([[1], [1], [1], [1], [-1], [-1], [-1], [-1]])
classify_LSC_LDA(X, Y, 'Table 1')

#Table 2
X = np.array([[3, 3], [3, 0], [2, 1], [0, 1.5], [-1, 1], [0, 0], [-1, -1], [1, 0]])
Y = np.array([[1], [1], [1], [1], [-1], [-1], [-1], [-1]])
classify_LSC_LDA(X, Y, 'Table 2')
```

Q3. Latent Semantic Analysis

- **Classify documents using one-vs-rest voted perceptron after applying PCA:**
  Observation:
  The accuracy of this method when we take 1110 singular values: 93.91634980988593

**Note: Value of k depends on how many train documents we take

When we vary the value of K, the following accuracies have been reported:

```
raja@Ranajit-Inspiron-5559:~/Raja/Sem2/SMAI/Assignment1/FinalSubmission$ python3 q3_a.py '../LSAdata (copy)/train/' '../LSAdata (copy)/test/'
Accuracy using k =  10
82.50950570342205
Accuracy using k =  60
82.12927756653993
Accuracy using k =  110
82.12927756653993
Accuracy using k =  160
86.31178707224335
Accuracy using k =  210
88.212927756654
Accuracy using k =  260
91.25475285171103
Accuracy using k =  310
90.11406844106465
Accuracy using k =  360
89.35361216730038
Accuracy using k =  410
89.35361216730038
Accuracy using k =  460
91.25475285171103
Accuracy using k =  510
90.8745247148289
Accuracy using k =  560
90.8745247148289
Accuracy using k =  610
89.73384030418251
Accuracy using k =  660
90.11406844106465
Accuracy using k =  710
90.49429657794677
Accuracy using k =  760
92.01520912547528
Accuracy using k =  810
91.63498098859316
Accuracy using k =  860
90.11406844106465
Accuracy using k =  910
92.39543726235742
Accuracy using k =  960
93.5361216730038
Accuracy using k =  1010
92.39543726235742
Accuracy using k =  1060
92.01520912547528
Accuracy using k =  1110
93.91634980988593
Accuracy using k =  1160
93.5361216730038
Accuracy using k =  1210
93.15589353612167
Accuracy using k =  1260
93.15589353612167
Accuracy using k =  1310
93.15589353612167
Accuracy using k =  1360
93.5361216730038
Accuracy using k =  1410
92.77566539923954
Accuracy using k =  1460
92.39543726235742
raja@Ranajit-Inspiron-5559:~/Raja/Sem2/SMAI/Assignment1/FinalSubmission$
```

Code:

```python
def sign(val):
    return -1 if val < 0 else 1

def formTFMatrix(documents, terms):
    '''
    Construct the bag of words matrix for each document
    where each row represents each document and
    columns represent each terms
    '''
    tf_matrix = []
    for words in documents:
        doc_terms = []
        for t in terms:
            doc_terms.append(words.count(t))
        tf_matrix.append(doc_terms)
    return np.array(tf_matrix)

def multiclass_voted_perceptron(X, Y, epoch):
    '''
    Voted percetron training, trained with for class
    '''
    m = X.shape[0]
    number_of_features = X.shape[1]
    W = np.zeros((1, number_of_features))
    B = np.zeros((1, 1))
    C = np.zeros((1, 1))
    n = 0
    for round in range(epoch):
        for i in range(m):
            if Y[i] * (np.dot(W[n], X[i].reshape(number_of_features, 1)) + B[n]) <= 0:
                W = np.append(W, np.array(W[n] + Y[i] * X[i]), axis=0)
                B = np.append(B, np.array([B[n] + Y[i]]), axis=0)
                C = np.append(C, [[1]], axis=0)
                n += 1
            else:
                C[n] += 1
    return W[1:, :], B[1:, :], C[1:, :]
```

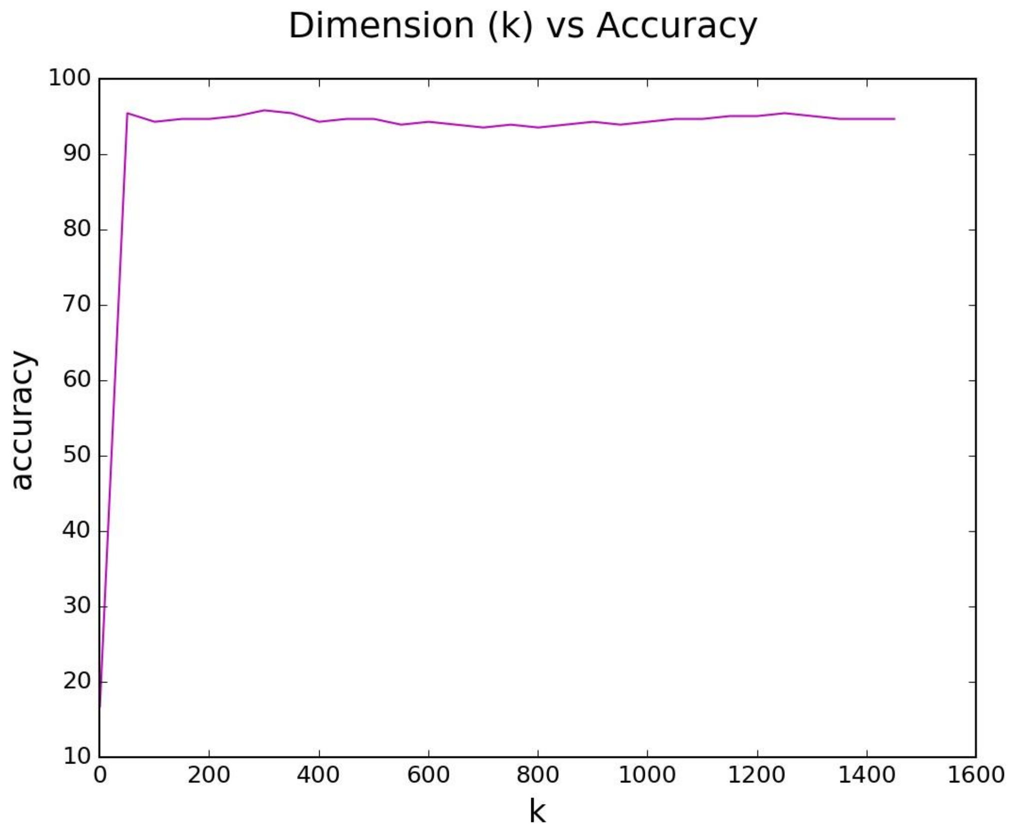
```python
def votedPerceptronPredict(X, Y, W_dict, B_dict, C_dict, class_labels):
    '''
    Predict the class label using the trained voted perceptron model
    Predict the class for which the absolute vaue prediction is maximum of other class
    '''
    Y_predicted = np.zeros((1, 1))
    examples = Y.shape[0]
    for e in range(examples):
        predict = np.asarray([])
        for x in class_labels:
            W = W_dict[x]
            B = B_dict[x]
            C = C_dict[x]
            shape_w = W.shape
            temp_val = np.dot(W, X[e].T).reshape(shape_w[0], 1) + B
            # temp_val[temp_val>0] = 1
            # temp_val[temp_val<0] = -1
            predict = np.append(predict, np.dot(C.T, temp_val))
        Y_predicted = np.append(Y_predicted, np.asarray([[(np.argmax(predict))]]))

    Y_predicted = Y_predicted[1:]
    return Y_predicted.reshape(examples, 1)

def one_vs_all_voted(X, Y, X_test, Y_test, epoch):
    '''
    Call voted perceptron for each of the classes
    '''
    class_labels = np.unique(Y)
    weight = {}
    bias = {}
    vote = {}
    Y_original = deepcopy(Y)
    for x in range(class_labels.shape[0]):
        Y = deepcopy(Y_original)
        for i in range(Y.shape[0]):
            if Y[i][0] == x:
                Y[i][0] = 1
            else:
                Y[i][0] = -1
        weight[x], bias[x], vote[x] = multiclass_voted_perceptron(X, Y, epoch)
    Y_predicted = votedPerceptronPredict(X_test, Y_test, weight, bias, vote, class_labels)
    correct_class = 0
    for i in range(Y_predicted.shape[0]):
        if Y_predicted[i][0] == Y_test[i][0]:
            correct_class += 1
```


```python
        for i in range(Y.shape[0]):
            if Y[i][0] == x:
                Y[i][0] = 1
            else:
                Y[i][0] = -1
        weight[x], bias[x], vote[x] = multiclass_voted_perceptron(X, Y, epoch)
    Y_predicted = votedPerceptronPredict(X_test, Y_test, weight, bias, vote, class_labels)
    correct_class = 0
    for i in range(Y_predicted.shape[0]):
        if Y_predicted[i][0] == Y_test[i][0]:
            correct_class += 1
```


```python
    print((correct_class / Y_predicted.shape[0]) * 100)

def extractDocs(dataPath):
    '''
    Extract words from the documents
    '''
    documents = []
    class_labels = []
    stop = set(stopwords.words('english')) # Reomoves the stop words eg the, an, a, and etc.
    stemmer = PorterStemmer() # Converts the words to their root words
    tokenizer = RegexpTokenizer(r'\w+') # Tokenize the document to et rid off non alphabets
    folders = [dataPath + name + '/' for name in os.listdir(dataPath) if os.path.isdir(dataPath + name)]
    class_name = [int(name) for name in os.listdir(dataPath) if os.path.isdir(dataPath + name)]
    it = 0
    for folder in folders:
        docFiles = [folder + docName for docName in os.listdir(folder)]
        doc_words = []
        for f in docFiles:
            file = codecs.open(f, 'r', 'ISO-8859-1')
            file_content = []
            for document in file:
                word_list = [stemmer.stem(line) for line in tokenizer.tokenize(document.lower()) if line not in '']
                file_content.extend([w for w in word_list if w not in stop])
            file.close()
            documents.append(file_content)
            class_labels.append(class_name[it])
        it += 1
    return documents, (np.array([class_labels])).T
```

```python

def getUniqueWords(documents):
    '''
    Find the dictionary of words
    '''
    words = [word for doc in documents for word in doc]
    unique_words = list(set(words))
    return unique_words

def classify_document_by_perceptron(dataPath, testPath):
    '''
    Classifies the documents after applying PCA
    and then using the voted perceptron model
    '''
    avg_accuracy = []
    documents, classes = extractDocs(dataPath)
    test_documents, test_classes = extractDocs(testPath)
    train_data_length = len(documents)
    for test_doc in test_documents:
        documents.append(test_doc)
    terms = getUniqueWords(documents)
    tf_matrix = formTFMatrix(documents, terms)
    tf = TfidfTransformer(norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False)
    tf_idf_matrix = tf.fit_transform(tf_matrix).todense() # Form the tf-idf matrix from bag of words
    test_data = tf_idf_matrix[train_data_length: , :]
    tf_idf_matrix = tf_idf_matrix[:train_data_length, :]
    U, Sigma_temp, V_T = np.linalg.svd(tf_idf_matrix) # Singular Value Decomposition
    Sigma = np.zeros((Sigma_temp.shape[0], Sigma_temp.shape[0]))
    for i in range(Sigma_temp.shape[0]):
        Sigma[i][i] = Sigma_temp[i]
    # k_list = list(range(10, min(tf_idf_matrix.shape[0], tf_idf_matrix.shape[1]), 50))
    k_list = [1110]
    for k in k_list:
        correct_similarity = 0
        # Reducce the dimension using the eigen value which contributes the most
        reduced_data = np.dot(np.dot(U[:, :k], Sigma[:k, :k]), V_T[:k, :])
        epoch = 40 # Epoch for the perceptron gtraing
        print("Accuracy using k = ", k)
        one_vs_all_voted(reduced_data, classes, test_data, test_classes, epoch)

dataPath = sys.argv[1] #'./LSAdata (copy)/train/'
testPath = sys.argv[2] #'./LSAdata (copy)/test/'

classify_document_by_perceptron(dataPath, testPath)

```

- **Predict the class label of the test document using cosine similarity:**
  <u>*Observation:*</u>
  As we increase the value of k, after some value the accuracy vs the values of k graph saturates, but at k = 1251 the accuracy is maximum.



SVD Accuracy:
Accuracy for k = 1
16.730038022813687
Accuracy for k = 51
95.43726235741445
Accuracy for k = 101
94.29657794676807
Accuracy for k = 151
94.67680608365019
Accuracy for k = 201
94.67680608365019
Accuracy for k = 251
95.05703422053232
Accuracy for k = 301
95.81749049429658
Accuracy for k = 351
95.43726235741445
Accuracy for k = 401
94.29657794676807
Accuracy for k = 451
94.67680608365019
Accuracy for k = 501
94.67680608365019
Accuracy for k = 551
93.91634980988593
Accuracy for k = 601
94.29657794676807

Accuracy for k = 651
93.91634980988593
Accuracy for k = 701
93.5361216730038
Accuracy for k = 751
93.91634980988593
Accuracy for k = 801
93.5361216730038
Accuracy for k = 851
93.91634980988593
Accuracy for k = 901
94.29657794676807
Accuracy for k = 951
93.91634980988593
Accuracy for k = 1001
94.29657794676807
Accuracy for k = 1051
94.67680608365019
Accuracy for k = 1101
94.67680608365019
Accuracy for k = 1151
95.05703422053232
Accuracy for k = 1201
95.05703422053232
Accuracy for k = 1251
95.43726235741445
Accuracy for k = 1301
95.05703422053232
Accuracy for k = 1351
94.67680608365019
Accuracy for k = 1401
94.67680608365019
Accuracy for k = 1451
94.67680608365019

Code:

```python
def extractDocs(dataPath):
    '''
    Extract words from the documents
    '''
    documents = []
    class_labels = []
    stop = set(stopwords.words('english')) # Reomoves the stop words eg the, an, a, and etc.
    stemmer = PorterStemmer() # Converts the words to their root words
    tokenizer = RegexpTokenizer(r'\w+') # Tokenize the document to et rid off non alphabets
    folders = [dataPath + name + '/' for name in os.listdir(dataPath) if os.path.isdir(dataPath + name)]
    class_name = [int(name) for name in os.listdir(dataPath) if os.path.isdir(dataPath + name)]
    it = 0
    for folder in folders:
        docFiles = [folder + docName for docName in os.listdir(folder)]
        doc_words = []
        for f in docFiles:
            file = codecs.open(f, 'r', 'ISO-8859-1')
            file_content = []
            for document in file:
                word_list = [stemmer.stem(line) for line in tokenizer.tokenize(document.lower()) if line not in '']
                file_content.extend([w for w in word_list if w not in stop])
            file.close()
            documents.append(file_content)
            class_labels.append(class_name[it])
        it += 1
    return documents, (np.array([class_labels])).T

def getUniqueWords(documents):
    '''
    Find the dictionary of words
    '''
    words = [word for doc in documents for word in doc]
    unique_words = list(set(words))
    return unique_words

def formTFMatrix(documents, terms):
    '''
    Construct the bag of words matrix for each document
    where each row represents each document and
    columns represent each terms
    '''
    td_matrix = []
    for words in documents:
        doc_terms = []
        for t in terms:
```

```python
def formTFMatrix(documents, terms):
    '''
    Construct the bag of words matrix for each document
    where each row represents each document and
    columns represent each terms
    '''
    td_matrix = []
    for words in documents:
        doc_terms = []
        for t in terms:
            doc_terms.append(words.count(t))
        td_matrix.append(doc_terms)
    return np.array(td_matrix)

def readtestDoc(testPath):
    '''
    Reading the test document
    '''
    documents = []
    stop = set(stopwords.words('english'))
    stemmer = PorterStemmer()
    tokenizer = RegexpTokenizer(r'\w+')
    file = codecs.open(testPath, 'r', 'ISO-8859-1')
    file_content = []
    for document in file:
        word_list = [stemmer.stem(line) for line in tokenizer.tokenize(document.lower()) if line not in '']
        file_content.extend([w for w in word_list if w not in stop])
    file.close()
    documents.append(file_content)
    return documents

def measure_similarity_by_cosine(dataPath, testPath, testDocLabel):
    avg_accuracy = []
    documents, classes = extractDocs(dataPath)
    # test_documents, test_classes = extractDocs(testPath)
    test_documents = readtestDoc(testPath)
    train_data_length = len(documents)
    for test_doc in test_documents:
        documents.append(test_doc)
    terms = getUniqueWords(documents)
    tf_matrix = formTFMatrix(documents, terms)
    tf = TfidfTransformer(norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False)
    tf_idf_matrix = tf.fit_transform(tf_matrix).todense() # Form the tf-idf matrix from bag of words
    test_data = tf_idf_matrix[train_data_length: , :]
```

```python
    tf_idf_matrix = tf_idf_matrix[:train_data_length, :]
    U, Sigma_temp, V_T = np.linalg.svd(tf_idf_matrix) # Singular Value Decomposition
    Sigma = np.zeros((Sigma_temp.shape[0], Sigma_temp.shape[0]))
    for i in range(Sigma_temp.shape[0]):
        Sigma[i][i] = Sigma_temp[i]

    # k_list = list(range(1, min(tf_idf_matrix.shape[0], tf_idf_matrix.shape[1]), 50))
    k_list = [1251] # Best value of k
    # print("SVD Accuracy:")
    for k in k_list:
        correct_similarity = 0
        reduced_data = np.dot(np.dot(U[:, :k], Sigma[:k, :k]), V_T[:k, :])
        cosine_similarity_matrix = sklearn.metrics.pairwise.cosine_similarity(reduced_data, test_data)
        for x in range(test_data.shape[0]):
            augmented_similarity_matrix = np.concatenate(
                (cosine_similarity_matrix[:, x].reshape(cosine_similarity_matrix[:, 0].shape[0], 1),
                    classes.reshape(classes.shape[0], 1)), axis = 1)
            sorted_simliarity_matrix = augmented_similarity_matrix[augmented_similarity_matrix[:, 0].argsort()]
            sorted_simliarity_matrix = sorted_simliarity_matrix[::-1]
            predicted_label = -1
            max_match = -1
            top_10_results = sorted_simliarity_matrix[:10, -1]
            labels = np.unique(top_10_results)
            for l in labels:
                if max_match < list(top_10_results).count(l):
                    max_match = list(top_10_results).count(l)
                    predicted_label = l
            print("The predicted class label of the document: ", predicted_label)
    #       if int(predicted_label) == int(test_classes[x]):
    #           correct_similarity += 1
    #   avg_accuracy.append(correct_similarity * 100 / test_data.shape[0])
    #   print("Accuracy for k = ", k),
    #   print(correct_similarity * 100 / test_data.shape[0])
    # return k_list, avg_accuracy

dataPath = sys.argv[1] #'./LSAdata (copy)/train/'
testPath = sys.argv[2] #'./LSAdata (copy)/test/'
testDocLabel = sys.argv[3]
measure_similarity_by_cosine(dataPath, testPath, testDocLabel)
# k, accuracies = measure_similarity_by_cosine(dataPath, testPath, testDocLabel)
# plt.suptitle("Dimension (k) vs Accuracy", fontsize = 18)
# plt.xlabel("k", fontsize = 16)
# plt.ylabel("accuracy", fontsize = 16)
# plt.plot(k, accuracies, color='m', label = "Accracy for different k")
# plt.savefig('KvsAccuracy_compressed.jpg', dpi = 150)
```