Assignment 2
CSE 471 : Statistical Methods in Artificial Intelligence (SMAI)

by Nagaraj Poti – 20162010

Problem 1. Write a program to implement online perceptron algorithm

1. Linear classifier to separate classes C1 and C2

Code files : perceptron_q1_a.py   Data files : dataset_q1_1.csv

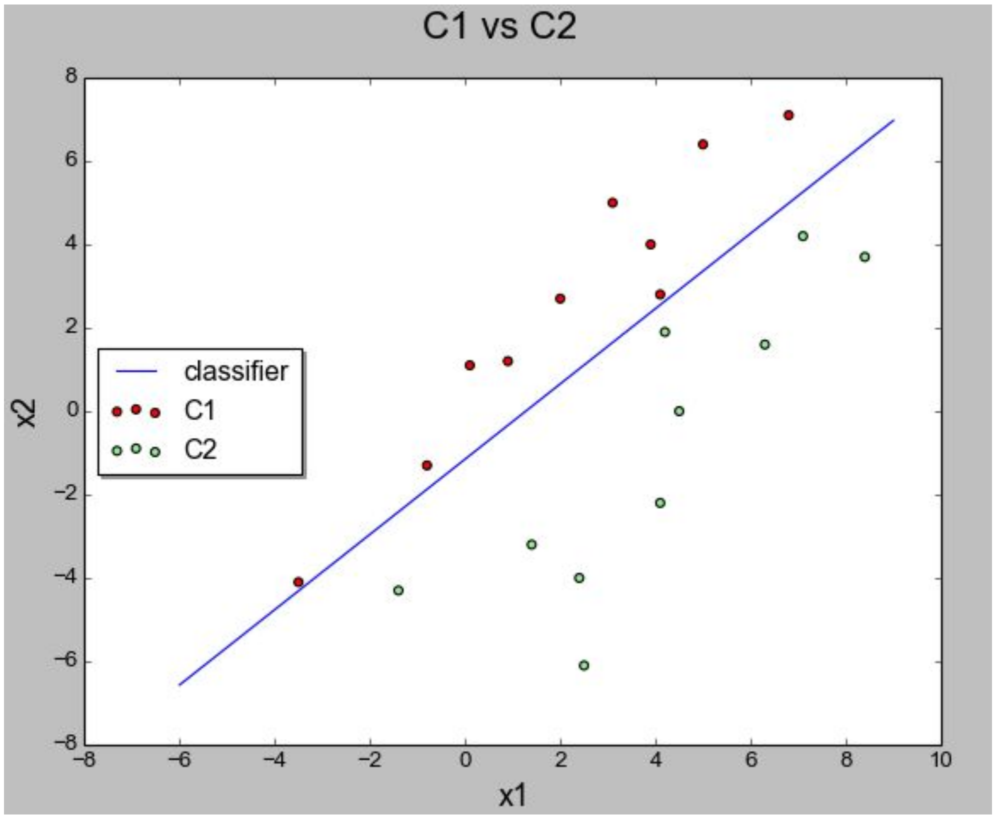
```python
# @Author - Nagaraj Poti
# @Roll - 20162010
#!/usr/bin/python

import csv
import numpy as np
import sys
import matplotlib.pyplot as plt

# Read csv files containing training data and populate array
def read_dataset(filename, y):
    with open(filename, 'rb') as csvfile:
        datasetreader = csv.reader(csvfile)
        for row in datasetreader:
            row = map(float, row)
            # Augment the feature vector
            row.append(1)
            # Append class label
            temp = [row, y]
            trainset.append(temp)

# Online perceptron
def online_perceptron(learning_rate, max_epoch_count, trainset):
    weights = [0.0 for i in range(len(trainset[0][0]))]
    iterations = 0
    epoch_count = 0
    change = True
    while change and epoch_count <= max_epoch_count:
        change = False
        for feature in trainset:
            x = feature[0]
            y = feature[1]
            if np.dot(y, np.dot(weights, x)) <= 0:
                weights = np.add(weights, np.dot(learning_rate, np.dot(y, x)))
                iterations += 1
                change = True
        epoch_count += 1
        print ("--> epoch = %d, iterations = %d" % (epoch_count, iterations))
    print
    return weights

# Read datasets from file and populate training set
trainset = []
read_dataset("dataset_q1_1.csv", 1)
read_dataset("dataset_q1_2.csv", -1)

print "Training set : "
print trainset ; print

learning_rate = 0.1
max_epoch_count = 100
print ("Learning rate = %.1f, Maximum epochs = %d" % (learning_rate, max_epoch_count))

weights = online_perceptron(learning_rate, max_epoch_count, trainset)
sys.stdout.write("Final augmented weight vector : ")
print weights

# Plotting code for given input dataset
d1x1 = []; d1x2 = []; d2x1 = []; d2x2 = []
for feature in trainset:
    if feature[1] == 1:
        d1x1.append(feature[0][0])
        d1x2.append(feature[0][1])
    else:
        d2x1.append(feature[0][0])
        d2x2.append(feature[0][1])
c1 = plt.scatter(d1x1, d1x2, c = "red", label = "C1")
c2 = plt.scatter(d2x1, d2x2, c = "lightgreen", label = "C2")
plt.suptitle("C1 vs C2", fontsize = 20)
plt.xlabel("x1", fontsize = 16)
plt.ylabel("x2", fontsize = 16)
```

```
# Plotting code for linear classifier
x = np.array(range(-6, 10))
y = eval("x * (-weights[0] / weights[1]) - (weights[2] / weights[1])")
classifier = plt.plot(x, y, label = 'classifier')

plt.legend(loc = 'center left', shadow = True)
plt.show()
```

Output

```
Learning rate = 0.1, Maximum epochs = 100
--> epoch = 1, iterations = 7
--> epoch = 2, iterations = 12
--> epoch = 3, iterations = 16
--> epoch = 4, iterations = 19
--> epoch = 5, iterations = 24
--> epoch = 6, iterations = 27
--> epoch = 7, iterations = 31
--> epoch = 8, iterations = 33
--> epoch = 9, iterations = 33

Final augmented weight vector : [-1.02  1.13  1.3]
```

Graph



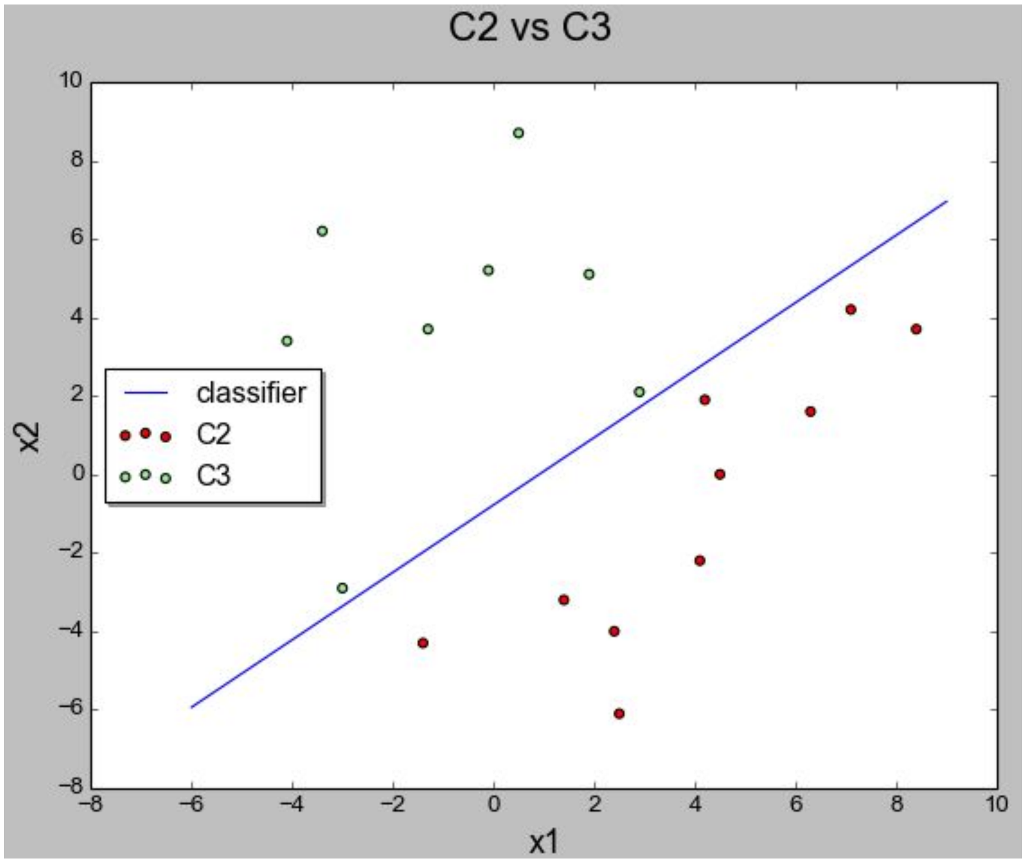
2. Linear classifier to separate classes C2 and C3

The program is the same as the previously give program except for the change in dataset.
Code files : perceptron_q1_b.py   Data files : dataset_q1_2.csv

Output

```
Learning rate = 0.1, Maximum epochs = 100
--> epoch = 1, iterations = 3
--> epoch = 2, iterations = 5
--> epoch = 3, iterations = 7
--> epoch = 4, iterations = 9
--> epoch = 5, iterations = 9

Final augmented weight vector : [ 0.55 -0.64 -0.5 ]
```

Graph



3. Comment on the difference on the number of iterations required for convergence

From the results it is observed that the online single sample perceptron algorithm converges in 33 iterations for dataset consisting of classes C1 and C2 and it converges in 9 iterations for dataset consisting of classes C2 and C3. This difference in the number of iterations required to converge is primarily due to the nature of the input datasets. In the case of C1 vs C2, the data is a lot more "denser", implying that the weight vector w is influenced by more number of points from both classes. Thus more iterations are required to stabilize the oscillations of the vector until classification is obtained. However in the case of C2 vs C3, we observe that the data is much more separated from each other. Thus the weight vector is less susceptible to changes and quickly stabilizes after separating the classes.

Problem 2. Write a program to implement voted perceptron algorithm

1. Dataset - Breast Cancer

The following program contains the code for both voted perceptron and vanilla perceptron for the above mentioned dataset. 10-fold cross-validation is applied by shuffling the training set and picking 10 mutually exclusive test sets accordingly. Tuples with missing data were ignored.

Code files : breast_cancer_aggregate_q2.py
perceptron_q2_2.py
voted_perceptron_q2_1.py

Data files : breast-cancer-wisconsin.data

```python
# @Author - Nagaraj Poti
# @Roll - 20162010
#!/usr/bin/python

import csv
import numpy as np
import random
import perceptron_q2_2 as vanilla
import matplotlib.pyplot as plt

# Pretty printing of float values
class prettyfloat(float):
    def __repr__(self):
        return "%0.2f" % self

# Read breast cancer csv file containing training data and populate array
def read_dataset(filename):
    with open(filename, 'rb') as csvfile:
        datasetreader = csv.reader(csvfile)
        for row in datasetreader:
            try :
                row = map(float, row)
                # Rearrange class label
                y = 1 if row[-1] == 2 else -1
                row.pop()
                # Augment the feature vector
                row.append(1)
                # Append class label
                trainset.append([row, y])
            except ValueError:
                pass;

# Voted perceptron
def voted_perceptron(learning_rate, max_epoch_count, trainset):
    weights = [0.0 for i in range(len(trainset[0][0]))]
    weight_history = []
    iterations = 0
    current_votes = 1
    epoch_count = 0
    change = True
    while change and epoch_count < max_epoch_count:
        change = False
        for feature in trainset:
            x = feature[0]
            y = feature[1]
            if np.dot(y, np.dot(weights, x)) <= 0:
                weight_history.append([weights, current_votes])
                iterations += 1
                current_votes = 1
                weights = np.add(weights, np.dot(learning_rate, np.dot(y, x)))
                change = True
            else:
                current_votes += 1
        epoch_count += 1
#   print ("--> perceptron = voted, epoch = %d, iterations = %d" % (epoch_count, iterations))
    weight_history.append([weights, current_votes])
    return weight_history

# Generate cross-validation test sets according to 10-fold cross validation
# Shuffle and split cross validation
```

```python
def cross_validation_generate(trainset, fold_count):
    test_dataset_folds = []
    trainset_c = list(trainset)
    fold_size = int(len(trainset) / fold_count)
    for i in range(fold_count):
        current_fold = []
        while (len(current_fold) < fold_size):
            i = random.randrange(0, len(trainset_c))
            current_fold.append(trainset_c.pop(i))
        test_dataset_folds.append(current_fold)
    return test_dataset_folds

# Test data classifier prediction score
def test_prediction(weights, fold):
    score = 0
    for feature in fold:
        x = feature[0]
        y = feature[1]
        s = 0
        for weight in weights:
            w = weight[0]
            c = weight[1]
            sign = 1 if np.dot(w, x) >= 0 else -1
            s += c * sign
        predicted_label = 1 if s >= 0 else -1
        if predicted_label == y:
            score += 1
    return float(score) / len(fold)

if __name__ == "__main__":

    # Read dataset from file and populate training set
    trainset = []
    read_dataset("breast-cancer-wisconsin.data")

    learning_rate = 1
    max_epoch = [10, 15, 20, 25, 30, 35, 40, 45, 50]

    # Average accuracy scores for both perceptrons
    voted_avg_acc = []
    vanilla_avg_acc = []

    for epoch_count in max_epoch:
        print ("Learning rate = %.1f, Maximum epochs = %d" % (learning_rate, epoch_count))
        testset = cross_validation_generate(trainset, 10)
        voted_scores = []
        vanilla_scores = []
        for fold in testset:
            current_trainset = list(testset)
            current_trainset.remove(fold)
            current_trainset = sum(current_trainset, [])
            weights = voted_perceptron(learning_rate, epoch_count, current_trainset)
            voted_scores.append(test_prediction(weights, fold))
            weights = vanilla.online_perceptron(learning_rate, epoch_count, current_trainset)
            vanilla_scores.append(vanilla.test_prediction(weights, fold))

        print "\nVoted perceptron validation scores : "
        voted_scores = map(prettyfloat, voted_scores)
        print voted_scores; print

        voted_avg_acc.append(float(sum(voted_scores)) / len(voted_scores) * 100)
        print ("Average accuracy : %.3f%%" % voted_avg_acc[-1])

        print "\nVanilla perceptron validation scores : "
        vanilla_scores = map(prettyfloat, vanilla_scores)
        print vanilla_scores; print

        vanilla_avg_acc.append(float(sum(vanilla_scores)) / len(vanilla_scores) * 100)
        print ("Average accuracy : %.3f%%" % vanilla_avg_acc[-1])
        print "-----------------------------------------------------------\n"

    # Plot accuracy values
    vot = plt.scatter([5 * i for i in range(2, 11)], voted_avg_acc, c = "red", label = "voted")
    van = plt.scatter([5 * i for i in range(2, 11)], vanilla_avg_acc, c = "green", label = "vanilla")
    vot = plt.plot([5 * i for i in range(2, 11)], voted_avg_acc, c = "red")
    van = plt.plot([5 * i for i in range(2, 11)], vanilla_avg_acc, c = "green")
    plt.suptitle("Vanilla perceptron vs Voted perceptron - Breast Cancer Dataset", fontsize = 18)
    plt.xlabel("Epochs", fontsize = 16)
    plt.ylabel("Accuracy %", fontsize = 16)
    plt.legend(loc = 'center left', shadow = True)
    plt.show()
```

The below code contains functions calls to the vanilla perceptron in the above code

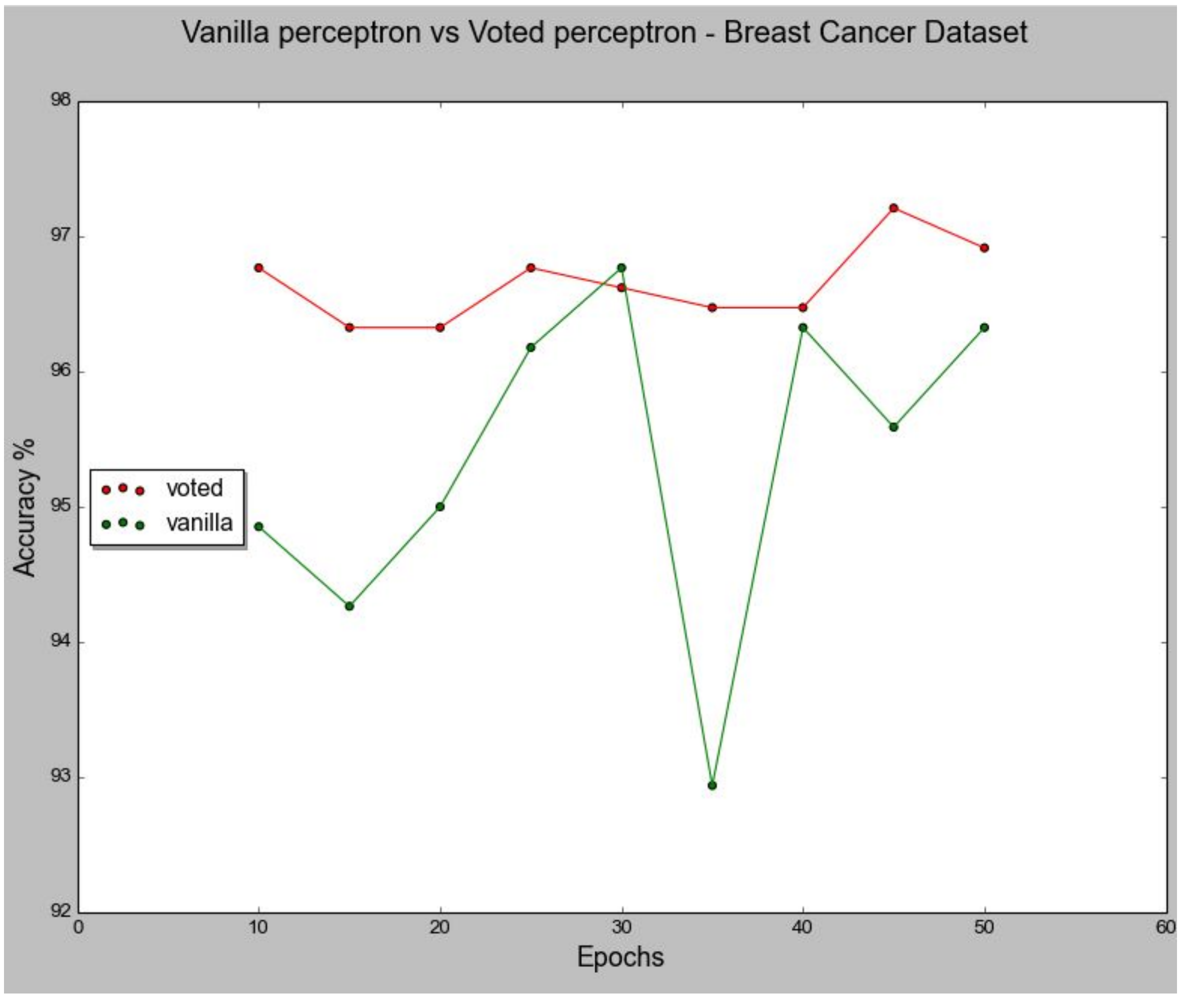
```python
# Online perceptron
def online_perceptron(learning_rate, max_epoch_count, trainset):
    weights = [0.0 for i in range(len(trainset[0][0]))]
    iterations = 0
    epoch_count = 0
    change = True
    while change and epoch_count < max_epoch_count:
        change = False
        for feature in trainset:
            x = feature[0]
            y = feature[1]
            if np.dot(y, np.dot(weights, x)) <= 0:
                weights = np.add(weights, np.dot(learning_rate, np.dot(y, x)))
                iterations += 1
                change = True
        epoch_count += 1
#   print ("--> perceptron = vanilla, epoch = %d, iterations = %d" % (epoch_count, ite
rations))
    return weights



# Test data classifier prediction score
def test_prediction(weights, fold):
    score = 0
    for feature in fold:
        x = feature[0]
        y = feature[1]
        sign = 1 if np.dot(weights, x) >= 0 else -1
        predicted_label = 1 if sign >= 0 else -1
        if predicted_label == y:
            score += 1
    return float(score) / len(fold)
```

Output

```
Learning rate = 1.0, Maximum epochs = 10

Voted perceptron validation scores :
[0.99, 0.94, 0.96, 0.94, 0.94, 0.97, 0.97, 0.99, 1.00, 0.99]

Average accuracy : 96.765%

Vanilla perceptron validation scores :
[0.91, 0.94, 0.94, 0.94, 0.90, 0.96, 0.91, 0.99, 1.00, 1.00]

Average accuracy : 94.853%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 15

Voted perceptron validation scores :
[0.97, 0.97, 0.99, 0.99, 0.94, 0.96, 0.97, 0.97, 0.93, 0.96]

Average accuracy : 96.324%

Vanilla perceptron validation scores :
[0.96, 0.97, 0.97, 0.94, 0.91, 0.91, 0.97, 0.94, 0.90, 0.96]

Average accuracy : 94.265%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 20

Voted perceptron validation scores :
[1.00, 0.96, 0.96, 0.99, 0.94, 0.93, 0.99, 0.96, 0.97, 0.96]

Average accuracy : 96.324%

Vanilla perceptron validation scores :
[0.99, 0.93, 0.94, 0.99, 0.97, 0.94, 0.85, 0.99, 0.96, 0.96]

Average accuracy : 95.000%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 25

Voted perceptron validation scores :
[0.97, 0.94, 0.94, 0.97, 0.93, 0.99, 1.00, 1.00, 0.96, 0.99]

Average accuracy : 96.765%

Vanilla perceptron validation scores :
[0.97, 0.93, 0.94, 0.99, 0.94, 0.99, 0.99, 0.94, 0.96, 0.99]

Average accuracy : 96.176%
-----------------------------------------------------------
Learning rate = 1.0, Maximum epochs = 30

Voted perceptron validation scores :
[0.97, 0.96, 0.99, 0.97, 0.97, 1.00, 0.94, 0.93, 0.96, 0.99]
```

```
Average accuracy : 96.618%

Vanilla perceptron validation scores :
[0.97, 0.96, 0.99, 0.97, 0.99, 0.99, 0.94, 0.96, 0.94, 0.99]

Average accuracy : 96.765%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 35

Voted perceptron validation scores :
[0.97, 0.99, 0.97, 0.96, 0.96, 0.96, 0.99, 1.00, 0.90, 0.97]

Average accuracy : 96.471%

Vanilla perceptron validation scores :
[0.99, 0.88, 0.97, 0.94, 0.84, 0.97, 0.96, 0.97, 0.85, 0.93]

Average accuracy : 92.941%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 40

Voted perceptron validation scores :
[0.97, 0.97, 0.99, 0.96, 0.94, 0.91, 0.97, 0.99, 0.96, 1.00]

Average accuracy : 96.471%

Vanilla perceptron validation scores :
[0.97, 0.97, 0.97, 0.96, 0.97, 0.90, 0.96, 0.99, 0.97, 0.99]

Average accuracy : 96.324%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 45

Voted perceptron validation scores :
[0.97, 0.97, 0.99, 0.91, 0.97, 0.99, 0.96, 1.00, 0.99, 0.99]

Average accuracy : 97.206%

Vanilla perceptron validation scores :
[0.97, 0.97, 0.90, 0.94, 0.96, 0.97, 0.96, 0.99, 0.96, 0.96]

Average accuracy : 95.588%
-----------------------------------------------------------
Learning rate = 1.0, Maximum epochs = 50

Voted perceptron validation scores :
[0.96, 0.96, 0.96, 0.94, 0.99, 0.99, 0.99, 0.99, 0.97, 0.97]

Average accuracy : 96.912%

Vanilla perceptron validation scores :
[0.96, 0.94, 0.96, 0.97, 0.99, 0.99, 0.99, 0.97, 0.97, 0.91]

Average accuracy : 96.324%
-----------------------------------------------------------
```

Graph



2. Dataset - Ionosphere

The program is the same as the previously give program except for the change in dataset.

Code files : ionosphere_aggregate_q2.py
perceptron_q2_4.py
voted_perceptron_q2_3.py

Data files : ionosphere.data

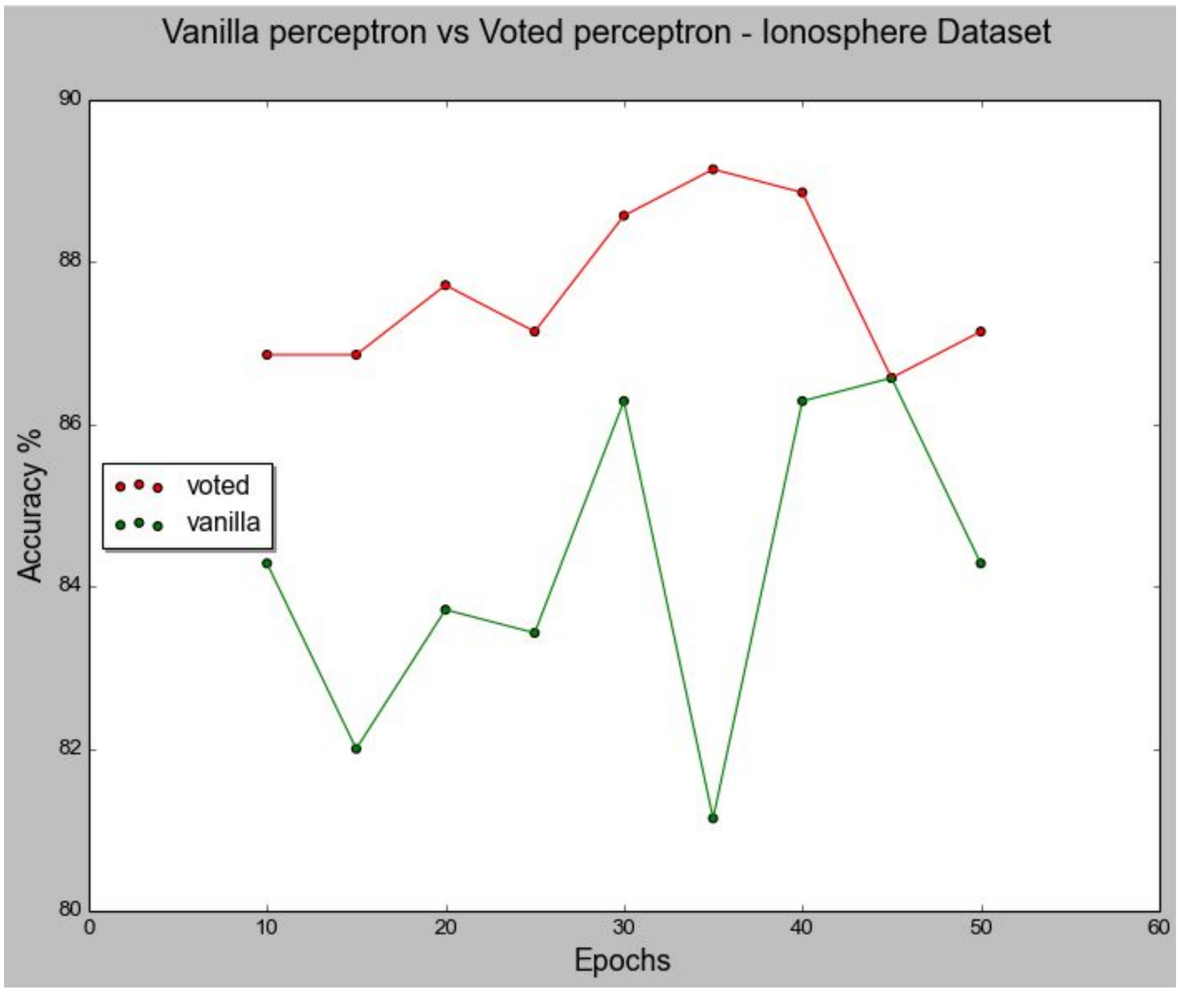
Output

```
Learning rate = 1.0, Maximum epochs = 10

Voted perceptron validation scores :
[0.86, 0.86, 0.91, 0.91, 0.74, 0.94, 0.89, 0.91, 0.86, 0.80]

Average accuracy : 86.857%
```

```
Vanilla perceptron validation scores :
[0.83, 0.86, 0.91, 0.86, 0.80, 0.91, 0.77, 0.86, 0.83, 0.80]

Average accuracy : 84.286%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 15

Voted perceptron validation scores :
[0.91, 0.86, 0.94, 0.86, 0.94, 0.80, 0.86, 0.86, 0.86, 0.80]

Average accuracy : 86.857%

Vanilla perceptron validation scores :
[0.91, 0.86, 0.80, 0.86, 0.91, 0.83, 0.71, 0.71, 0.83, 0.77]

Average accuracy : 82.000%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 20

Voted perceptron validation scores :
[0.91, 0.86, 0.86, 0.89, 0.89, 0.91, 0.77, 0.89, 0.86, 0.94]

Average accuracy : 87.714%

Vanilla perceptron validation scores :
[0.89, 0.83, 0.80, 0.89, 0.86, 0.91, 0.74, 0.86, 0.77, 0.83]

Average accuracy : 83.714%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 25

Voted perceptron validation scores :
[0.94, 0.94, 0.83, 0.83, 0.94, 0.86, 0.80, 0.89, 0.83, 0.86]

Average accuracy : 87.143%

Vanilla perceptron validation scores :
[0.89, 0.91, 0.69, 0.80, 0.94, 0.74, 0.80, 0.89, 0.83, 0.86]

Average accuracy : 83.429%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 30

Voted perceptron validation scores :
[0.91, 0.94, 0.86, 0.86, 0.91, 0.94, 0.83, 0.83, 0.91, 0.86]

Average accuracy : 88.571%

Vanilla perceptron validation scores :
[0.86, 0.89, 0.83, 0.83, 0.89, 0.91, 0.83, 0.86, 0.89, 0.86]
```

```
Average accuracy : 86.286%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 35

Voted perceptron validation scores :
[0.89, 0.89, 0.97, 0.83, 0.89, 0.86, 0.91, 0.91, 0.89, 0.89]

Average accuracy : 89.143%

Vanilla perceptron validation scores :
[0.86, 0.71, 0.77, 0.83, 0.80, 0.69, 0.89, 0.91, 0.80, 0.86]

Average accuracy : 81.143%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 40

Voted perceptron validation scores :
[0.83, 0.91, 0.94, 0.91, 0.89, 0.89, 0.83, 0.91, 0.86, 0.91]

Average accuracy : 88.857%

Vanilla perceptron validation scores :
[0.80, 0.94, 0.91, 0.86, 0.89, 0.89, 0.77, 0.91, 0.80, 0.86]

Average accuracy : 86.286%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 45

Voted perceptron validation scores :
[0.86, 0.89, 0.97, 0.83, 0.74, 0.80, 0.83, 0.91, 0.94, 0.89]

Average accuracy : 86.571%

Vanilla perceptron validation scores :
[0.89, 0.91, 0.91, 0.80, 0.80, 0.83, 0.83, 0.91, 0.94, 0.83]

Average accuracy : 86.571%
-----------------------------------------------------------

Learning rate = 1.0, Maximum epochs = 50

Voted perceptron validation scores :
[0.91, 0.77, 0.89, 0.91, 0.91, 0.86, 0.77, 0.91, 0.91, 0.86]

Average accuracy : 87.143%

Vanilla perceptron validation scores :
[0.89, 0.77, 0.86, 0.69, 0.94, 0.80, 0.74, 0.91, 0.91, 0.91]

Average accuracy : 84.286%
-----------------------------------------------------------
```

Graph



3. Comment on the performance of the two approaches

From the results obtained, we observe that the voted perceptron performs much better and more consistently than the vanilla perceptron. By using random 10-fold cross validation on both the datasets it was observed that the voted perceptron varied between 94% and 98% in terms of accuracy for breast-cancer-wisconsin dataset and between 86% and 94% for ionosphere dataset. On the other hand vanilla perceptron varied between 91% and 96% for breast-cancer-wisconsin dataset and between 80% and 87% for ionosphere dataset. These results can be attributed to the following :

- Voted perceptron gave more importance to better performing weight vectors, leading to better results with respect to classification.
- If the initial weight vector (in our case 0) was a bad guess, the vanilla perceptron is affected more and is not able to compensate fast enough as a result of which convergence is slower.
- As we have restricted the number of epochs, vanilla perceptron simply picks the last weight

vector. If that weight vector is poorer compared to the previous history of weight vectors, it does not know it. However voted keeps track of it and is thus able to make a better judgement of the weight vector that should be used.

Problem 3. Least squares approach

1a. Write a program to find the linear classifier using least square approach - Dataset 1

The following program implements Widrow-Hoff procedure to obtain the least square classifier

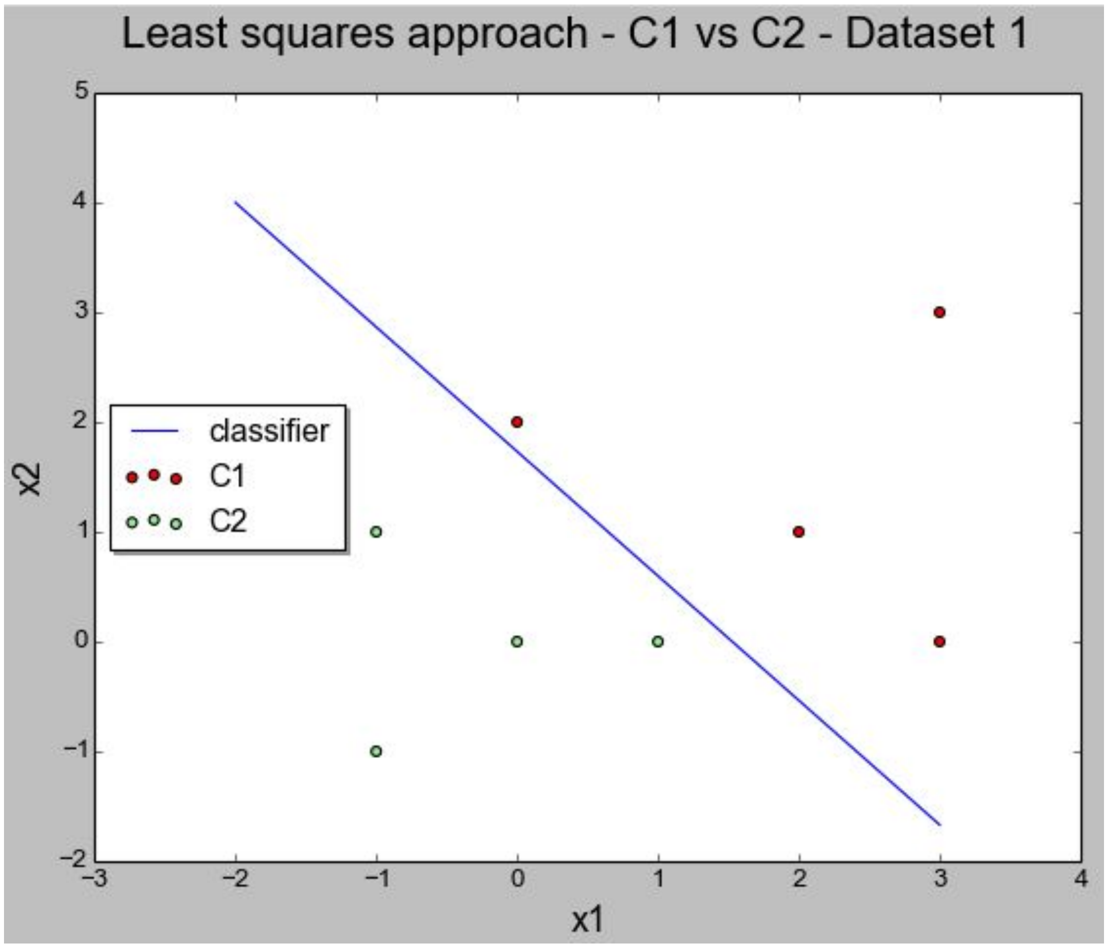
Code files : least_squares_q3_1.py

```python
# @Author - Nagaraj Poti
# @Roll - 20162010
#!/usr/bin/python

import numpy as np
import matplotlib.pyplot as plt
import sys

# Pretty printing of float values
class prettyfloat(float):
    def __repr__(self):
        return "%0.2f" % self

# Widrow Hoff procedure
def least_squares(learning_rate, trainset, max_iterations):
    weights = [0.0 for i in range(len(trainset[0]))]
    cur_iterations = 1
    # Learning rate eita updates after each iteration
    eita = learning_rate
    x = trainset[cur_iterations - 1]
  # Error
    delta = np.dot(np.dot(eita, np.subtract(1, np.dot(weights, x))), x)
    while cur_iterations <= max_iterations and np.linalg.norm(delta) > 10e-9:
        weights = np.add(weights, delta)
        x = trainset[cur_iterations % len(trainset)]
        cur_iterations += 1
        eita = float(learning_rate) / cur_iterations
        delta = np.dot(np.dot(eita, np.subtract(1, np.dot(weights, x))), x)
    return weights

# Dataset 1 - Augmented vectors
class2_orig = [[-1,1,1],[0,0,1],[-1,-1,1],[1,0,1]]
class1 = [[3,3,1],[3,0,1],[2,1,1],[0,2,1]]
class2 = [[1,-1,-1],[0,0,-1],[1,1,-1],[-1,0,-1]]

learning_rate = 1
max_iterations = 10000
print ("Learning rate = %.1f" % learning_rate)

weights = least_squares(learning_rate, class1 + class2, max_iterations)
sys.stdout.write("\nFinal augmented weight vector : ")
print map(prettyfloat, weights)

# Plotting code for given input dataset
d1x1 = []; d1x2 = []; d2x1 = []; d2x2 = []
for feature in class1:
    d1x1.append(feature[0])
    d1x2.append(feature[1])
for feature in class2_orig:
    d2x1.append(feature[0])
    d2x2.append(feature[1])
c1 = plt.scatter(d1x1, d1x2, c = "red", label = "C1")
c2 = plt.scatter(d2x1, d2x2, c = "lightgreen", label = "C2")
plt.suptitle("Least squares approach - C1 vs C2 - Dataset 1", fontsize = 20)
plt.xlabel("x1", fontsize = 16)
plt.ylabel("x2", fontsize = 16)

# Plotting code for linear classifier
x = np.array(range(-2, 4))
y = eval("x * (-weights[0] / weights[1]) - (weights[2] / weights[1])")
classifier = plt.plot(x, y, label = "classifier")

plt.legend(loc = 'center left', shadow = True)
plt.show()
```

Output

```
Learning rate = 1.0

Final augmented weight vector : [0.38, 0.33, -0.57]
```

Graph



1b. Write a program to find the linear classifier using least square approach - Dataset 2

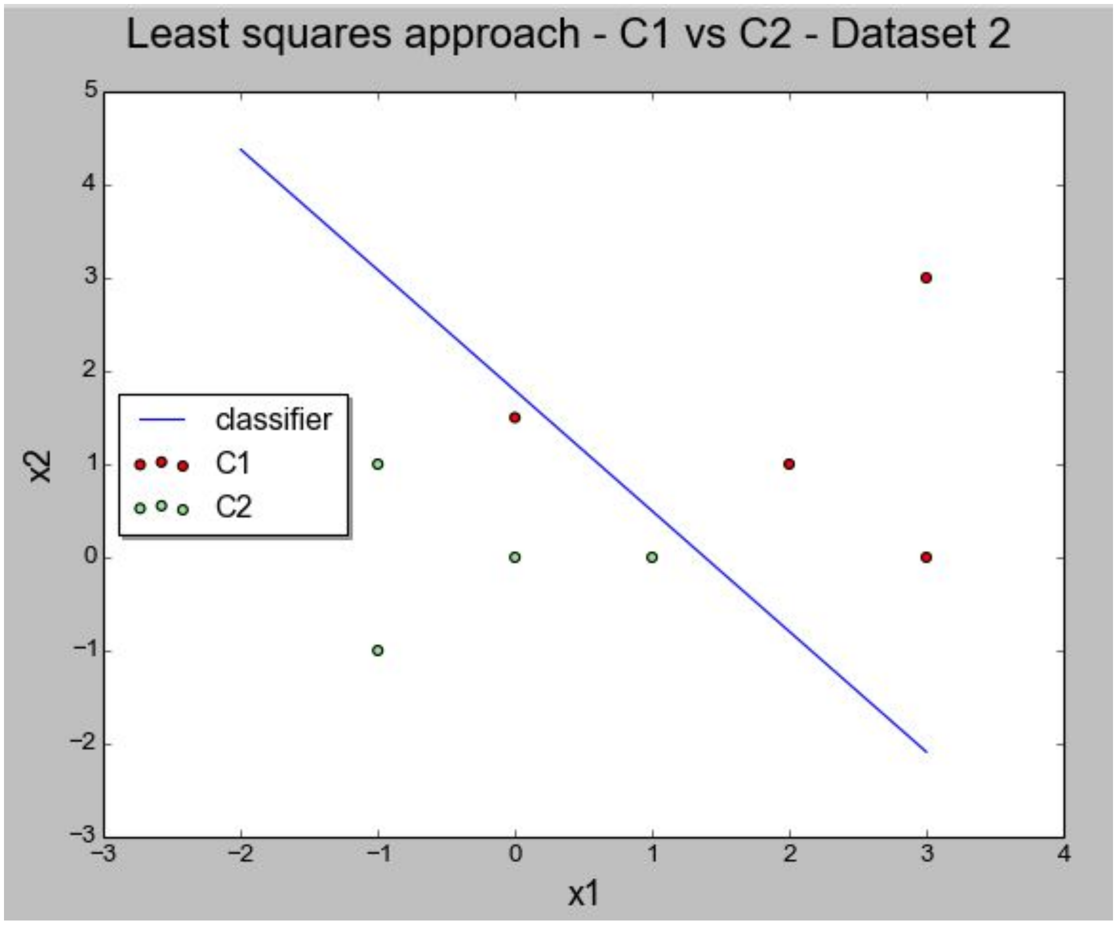
The program is the same as the previously give program except for the change in dataset.

Code files : least_squares_q3_2.py

Output

```
Learning rate = 1.0

Final augmented weight vector : [0.38, 0.29, -0.52]
```

Graph



2a. Write a program to find the linear classifier using Fisher's linear discriminant - Dataset 1

The Fisher's linear discriminant projection is first obtained. Afterwards, vanilla online perceptron is applied to obtain the classifier line.

Code files : fisher_q3_3.py
perceptron_q2_2.py

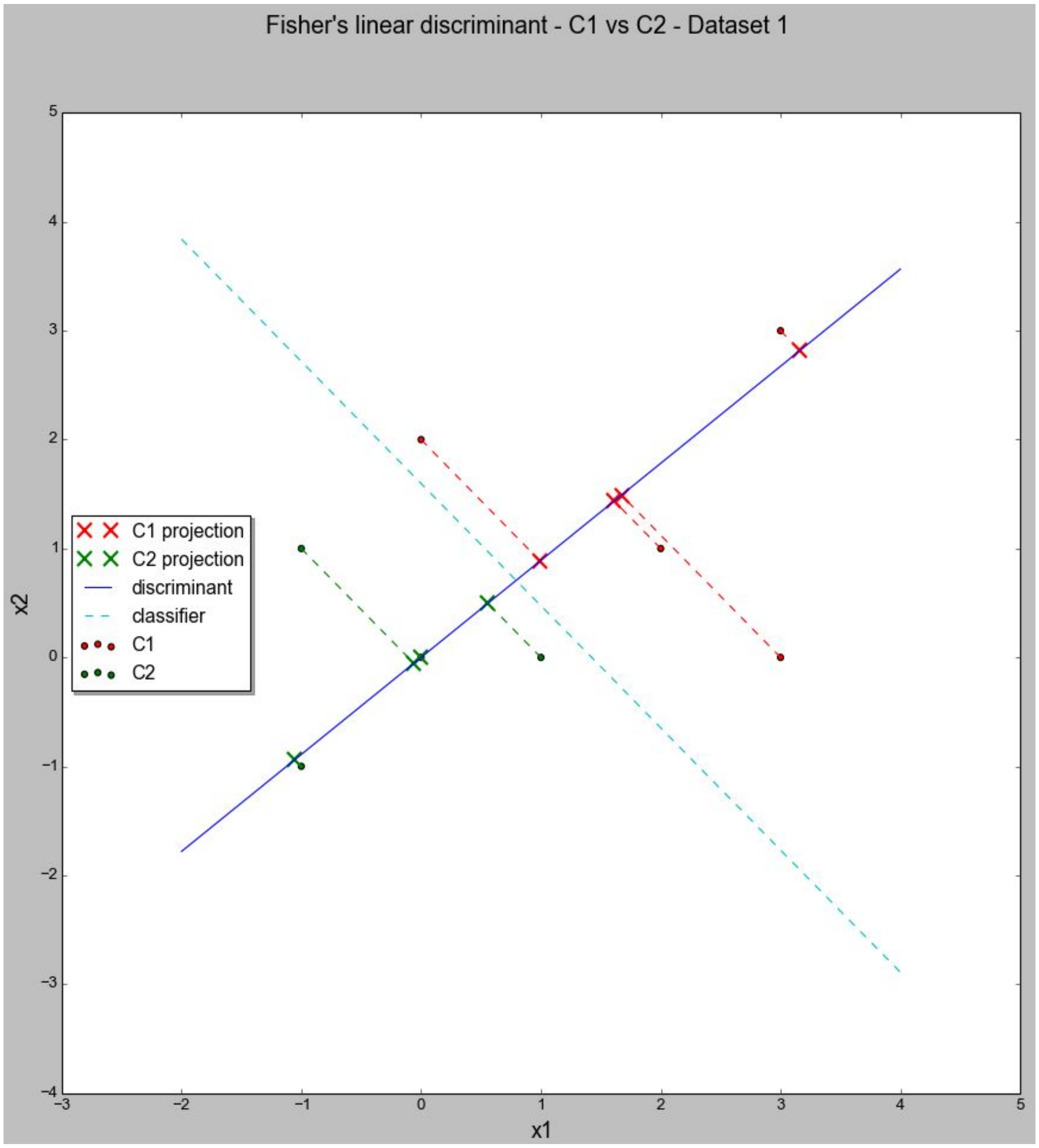
```python
# @Author - Nagaraj Poti
# @Roll - 20162010
#!/usr/bin/python

import numpy as np
import matplotlib.pyplot as plt
import perceptron_q2_2 as vanilla

```

```
# Pretty printing of float values
class prettyfloat(float):
    def __repr__(self):
        return "%0.2f" % self

# Dataset 1
class1 = [[3,3],[3,0],[2,1],[0,2]]
class2 = [[-1,1],[0,0],[-1,-1],[1,0]]

# Compute mean vectors of each class
mean_d1 = np.mean(class1, axis = 0)
mean_d2 = np.mean(class2, axis = 0)

# Compute within class scatter matrix
scatter_data1 = np.dot((class1 - mean_d1).T, (class1 - mean_d1))
scatter_data2 = np.dot((class2 - mean_d2).T, (class2 - mean_d2))
scatter_within = scatter_data1 + scatter_data2
print "Within class scatter matrix : "
for l in scatter_within:
    print map(prettyfloat, l)

# Calculate weight vector
weights = np.dot(np.linalg.inv(scatter_within), (mean_d1 - mean_d2))
weights = weights / np.linalg.norm(weights)
print "\nUnit weight vector : "
print map(prettyfloat, weights)

# Plot dataset
d1x1 = zip(*class1)[0]; d1x2 = zip(*class1)[1]
d2x1 = zip(*class2)[0]; d2x2 = zip(*class2)[1]
c1 = plt.scatter(d1x1, d1x2, c = "red", label = "C1")
c2 = plt.scatter(d2x1, d2x2, c = "green", label = "C2")
plt.suptitle("Fisher's linear discriminant - C1 vs C2 - Dataset 1", fontsize = 18)
plt.xlabel("x1", fontsize = 16)
plt.ylabel("x2", fontsize = 16)

# Plot fisher's linear discriminant - dimension reduction
# Class 1 plot
scalars = np.dot(class1, weights)
points = []
for value in scalars:
    points.append(np.dot(value, weights))
total_points = [[[i[0],i[1],1],1] for i in points]
plt.plot(zip(*points)[0], zip(*points)[1], "rx", markersize = 10, mew = 2, label = "C1 projection")

for i in range(len(points)):
    plt.plot([class1[i][0], points[i][0]], [class1[i][1], points[i][1]], 'r--')

# Class 2 plot
scalars = np.dot(class2, weights)
points = []
for value in scalars:
    points.append(np.dot(value, weights))
total_points += [[[i[0],i[1],1],-1] for i in points]
plt.plot(zip(*points)[0], zip(*points)[1], "gx", markersize = 10, mew = 2, label = "C2 projection")

for i in range(len(points)):
    plt.plot([class2[i][0], points[i][0]], [class2[i][1], points[i][1]], 'g--')

x = np.array(range(-2, 5))
y = eval("(x - points[0][0]) * (weights[1] / weights[0]) + points[0][1]")
discriminant = plt.plot(x, y, label = 'discriminant')

# Apply online perceptron to obtain classifier line
w = vanilla.online_perceptron(0.1, 10000, total_points)

# Plotting code for linear classifier
x = np.array(range(-2, 5))
y = eval("x * (-w[0] / w[1]) - (w[2] / w[1])")
classifier = plt.plot(x, y, 'c--', label = 'classifier')

plt.legend(loc = 'center left', shadow = True)
plt.show()
```

Output

```
Within class scatter matrix :
[8.75, -1.00]
[-1.00, 7.00]

Unit weight vector :
[0.75, 0.67]
```

Graph



2b. Write a program to find the linear classifier using Fisher's linear discriminant - Dataset 2

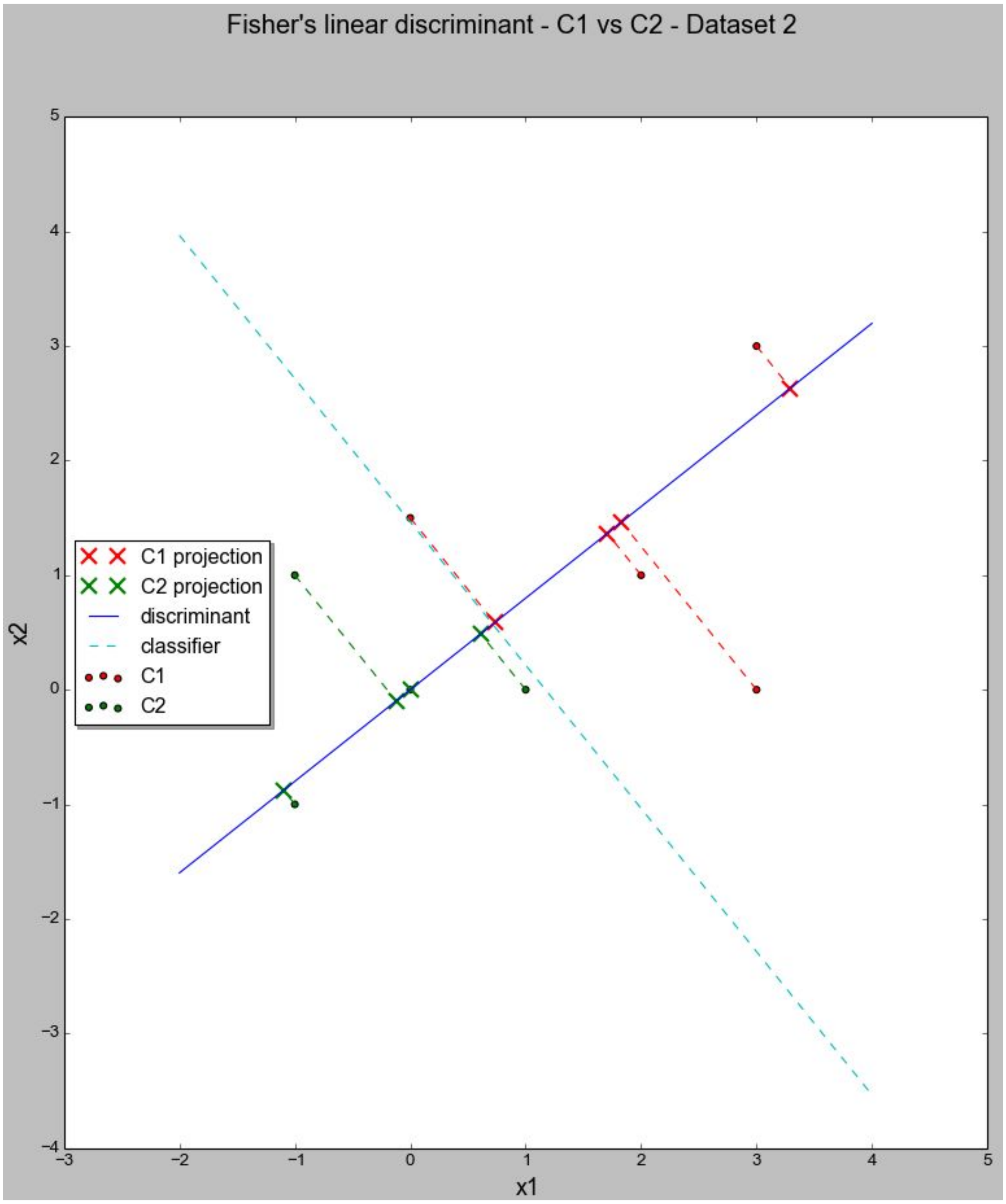
The program is the same as the previously give program except for the change in dataset.
Code files : fisher_q3_4.py , perceptron_q2_2.py

Output

```
Within class scatter matrix :
```

```
[8.75, 0.00]
[0.00, 6.69]

Unit weight vector :
[0.78, 0.62]
```

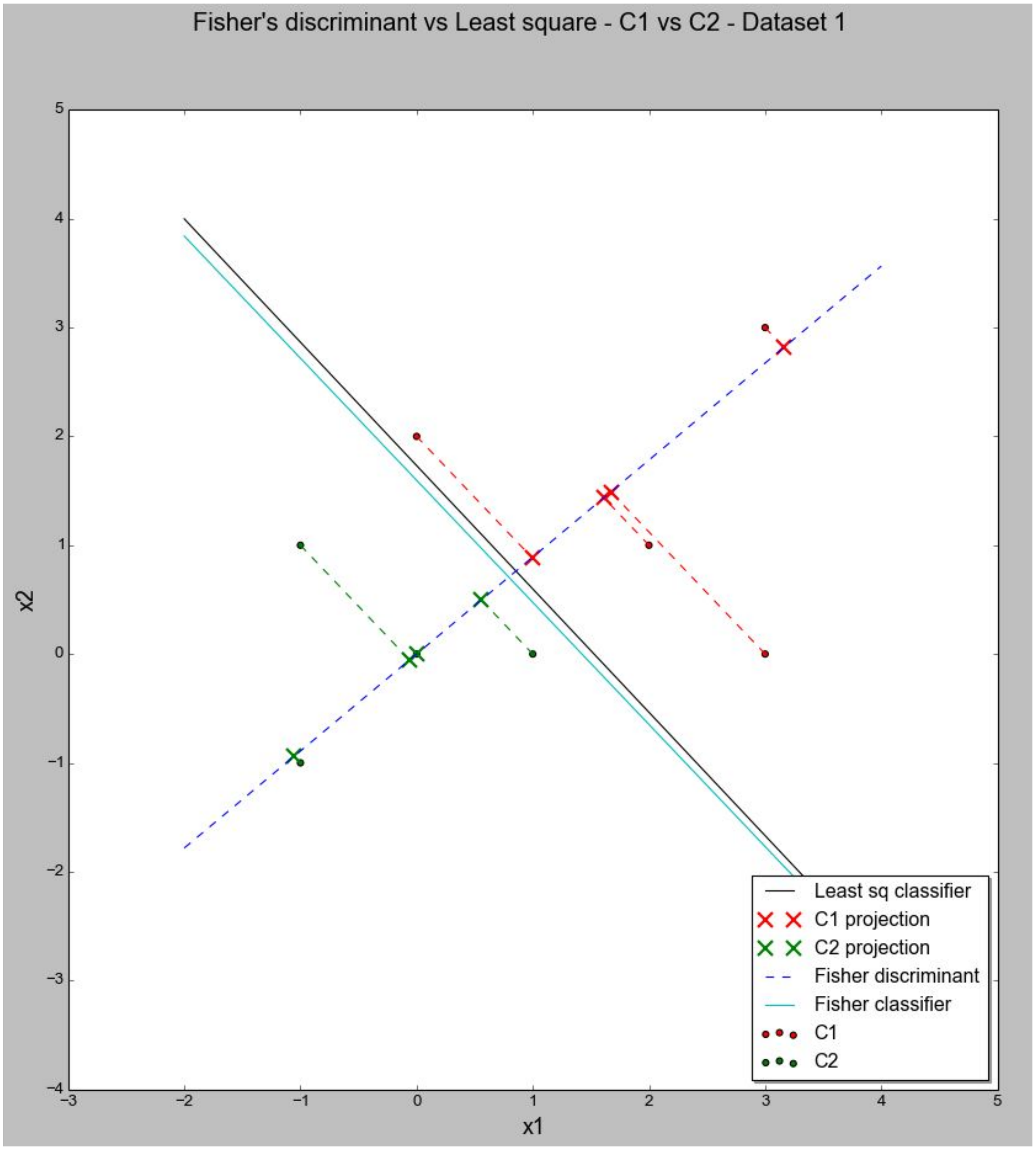
Graph



3. Plot both classifiers on the same graph - Dataset 1

The code of least squares and fisher discriminant was combined to plot results on the same graph.
Code files : d1_fisher_vs_lsquare.py

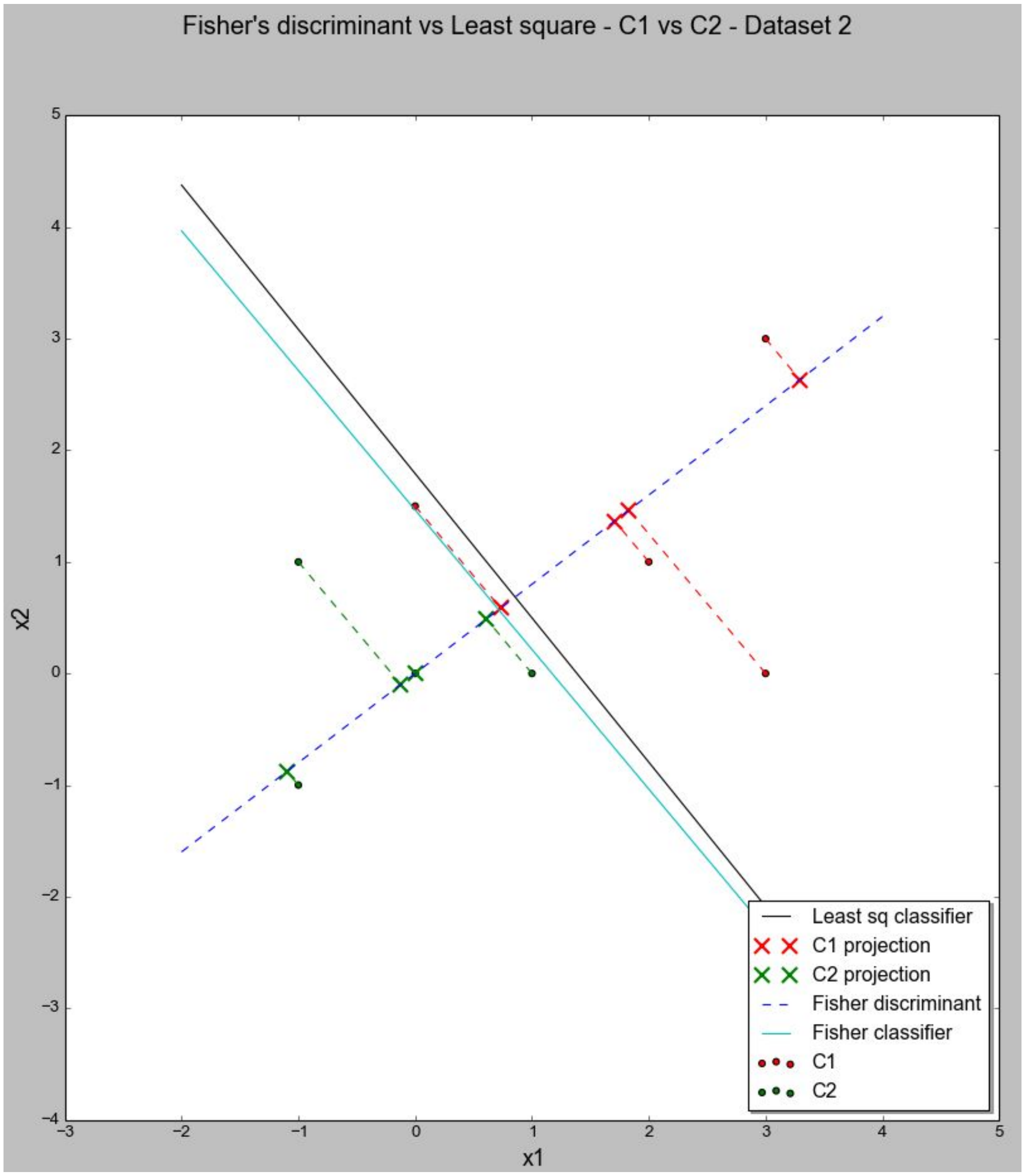
Graph



4. Plot both classifiers on the same graph - Dataset 2

The code of least squares and fisher discriminant was combined to plot results on the same graph.
Code files : d2_fisher_vs_lsquare.py

Graph



4. Comment on the difference in the classifier learnt in the above two cases. Give reasons.

From the above results, we can see that least squares classifier is able to linearly separate dataset 1 but is unable to do so with dataset 2. However, using fisher's discriminant we are able to linearly separate both the given datasets. From this we can draw the following conclusions:

- Least squares is not guaranteed to classify even if the dataset is linearly separable. This is evident in dataset 2. Least square tries to minimize the error at all instances, and thus does not give priority to misclassification.
- In the case of fisher's linear discriminant, it tried to maximize the separability between the classes. Thus the one dimensional projection was the best possible separation. Since the data was linearly classifiable as well, running the perceptron algorithm on the projected data gave us a classifier that was able to linearly separate between the classes.

Problem 4. Relation between least squares and Fisher's linear discriminant

In a two class problem, the minimum squared error solution seeks a linear discriminant function

$$g(x) = w^T x + w_0$$

Thus, we get

$$g(x) = \begin{cases} c_1 & \text{if } x \in \text{Class } 1 \\ c_2 & \text{if } x \in \text{Class } 2 \end{cases}$$

When c_i is prespecified for each class, the problem can be reformulated to minimize the squared error

$$\left\| \begin{bmatrix} 1 & a_1^T \\ \vdots & \vdots \\ 1 & a_n^T \end{bmatrix} [w_0 \ w]^T - \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 \quad — \text{①}$$

If we let $P = \begin{bmatrix} 1 & a_1^T \\ \vdots & \vdots \\ 1 & a_n^T \end{bmatrix}$,

then $\begin{bmatrix} w_0 \\ w \end{bmatrix} = P^+ \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$ — ② where P^+ is the pseudo inverse of P

If we choose different $c = [c_1 \ c_2]^T$, we wolld get different discriminant functions. Let us sppose that

$c_1 = M/m_1$ and $c_2 = -M/m_2$.

Thus the class labels can be specified as follows:

$$y_i = \begin{cases} \frac{m}{m_1} & \text{if } x_i \in \text{Class } 1 \\ -\frac{m}{m_2} & \text{if } x_i \in \text{Class } 2 \end{cases}$$

From ① and ②, we can write

$$\begin{bmatrix} I_1^t & -I_2^t \\ x_1^t & -x_2^t \end{bmatrix} \begin{bmatrix} I_1 & x_1 \\ -I_2 & -x_2 \end{bmatrix} \begin{bmatrix} w_0 \\ w \end{bmatrix} = \begin{bmatrix} I_1^t & -I_2^t \\ x_1^t & -x_2^t \end{bmatrix} \begin{bmatrix} m/m_1 I_1 \\ -m/m_2 I_2 \end{bmatrix} \quad - ③$$

We can define sample means μ_i and the combined scatter matrix S_w as

$$\mu_i = \frac{1}{n_i} \sum_{x \in D_i} x \qquad i = 1, 2 \quad - ④$$

and

$$S_w = \sum_{i=1}^{2} \sum_{x \in D_i} (x - \mu_i)(x - \mu_i)^t$$

Multiplying matrices in ③, we get

$$\begin{bmatrix} m & (m_1\mu_1 + m_2\mu_2)^t \\ m_1\mu_1 + m_2\mu_2 & S_w + m_1\mu_1\mu_1^t + m_2\mu_2\mu_2^t \end{bmatrix} \begin{bmatrix} w_0 \\ w \end{bmatrix} = \begin{bmatrix} 0 \\ m(\mu_1 - \mu_2) \end{bmatrix}$$

This can be viewed as a pair of equations.
The first can be solved for w_0 in terms of w:

$$w_0 = -\mu^t w,$$

where μ is the mean of all samples. Substituting in the second equation we get

$$\left[\frac{1}{m} S_w + \frac{m_1 \mu_2}{m^2} (\mu_1 - \mu_2)(\mu_1 - \mu_2)^t \right] w = \mu_1 - \mu_2$$

Since $(\mu_1 - \mu_2)(\mu_1 - \mu_2)^t w$ is in the direction of $\mu_1 - \mu_2$ for any value w, we can write

$$\frac{m_1 m_2}{m^2} (\mu_1 - \mu_2)(\mu_1 - \mu_2)^t w = (1-\alpha)(\mu_1 - \mu_2)$$

where α is some scalar.

We can re write the above equation as

$$w = \alpha m S_w^{-1} (\mu_1 - \mu_2)$$

We observe that this is almost same as fisher's linear discriminant rule with the omission of the sale factor α. The threshold weight w_0 can be obtained by the following decision rule.

Decide ω_1 if $w^T(x - \mu) \geq 0$; otherwise decide ω_2