ÖZYEĞİN
UNIVERSITY

CS 454

Introduction to Machine Learning and Artificial Neural Networks

Prof. Dr. Ethem Alpaydın

Fall 2021

Homework #2

Nearest Mean and Nearest Neighbor Classification

by

Abdullah Saydemir

S014646

November 7, 2021

Part 1

a) Plots for the training and testing data are as follows.

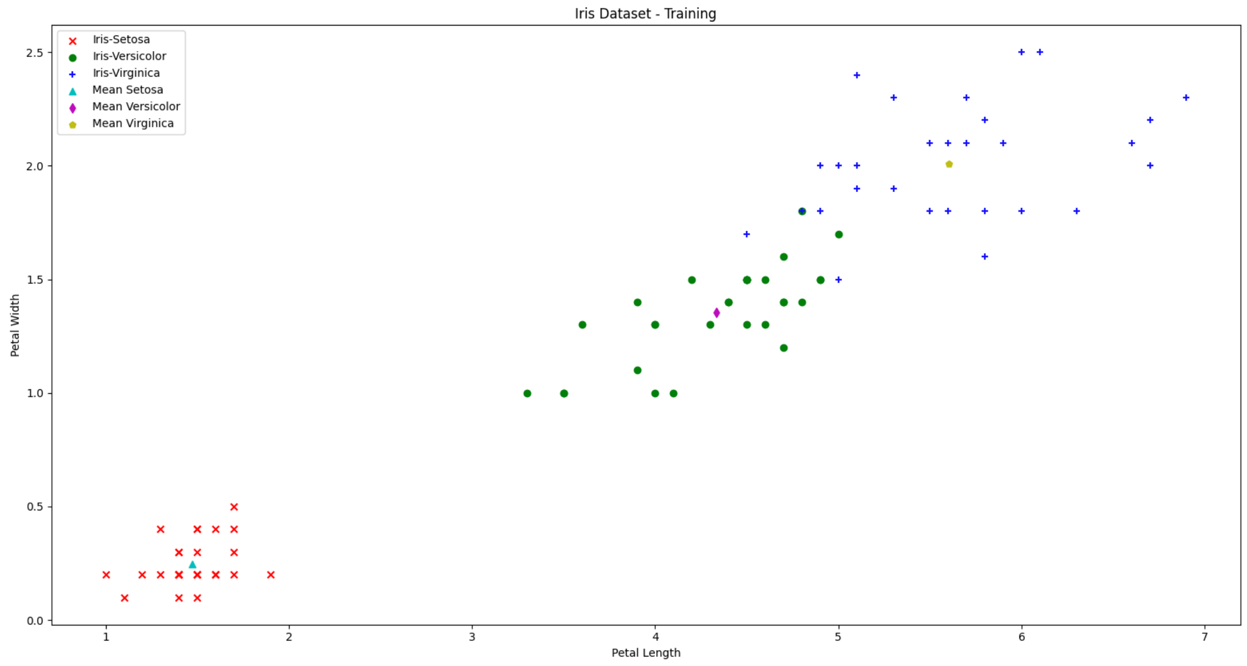


Figure 1

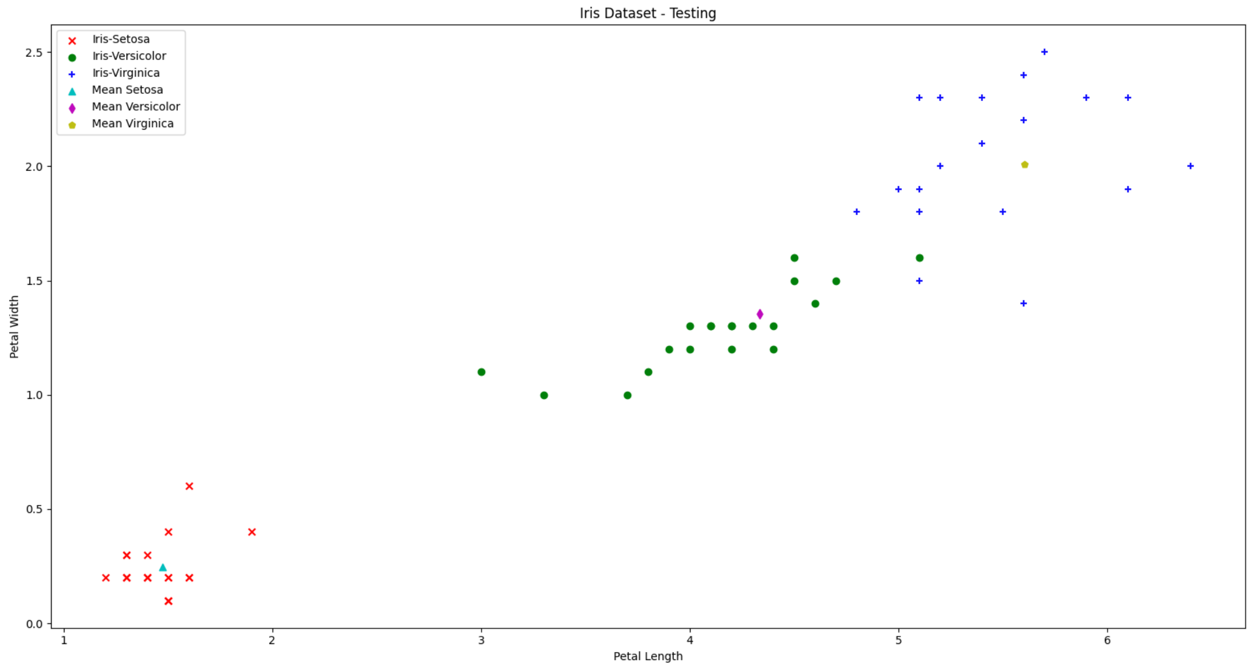


Figure 2

b) Confusion matrices of nearest mean classification for testing and training data are as follows.

Nearest Mean Training		Actual		
		Setosa	Versicolor	Virginica
Action	Setosa	30	0	0
	Versicolor	0	29	5
	Virginica	0	1	25

Figure 3

Nearest Mean Test		Actual		
		Setosa	Versicolor	Virginica
Action	Setosa	20	0	0
	Versicolor	0	19	1
	Virginica	0	1	19

Figure 4

Part 2

a) Confusion matrices of k-nearest neighbor classification for testing data are as follows.

Nearest Neighbor Test k = 1		Actual		
		Setosa	Versicolor	Virginica
Action	Setosa	20	0	0
	Versicolor	0	19	1
	Virginica	0	1	19

Figure 5

Nearest Neighbor Test k = 3		Actual		
		Setosa	Versicolor	Virginica
Action	Setosa	20	0	0
	Versicolor	0	20	1
	Virginica	0	0	19

Figure 6

Nearest Neighbor Test k = 5		Actual		
		Setosa	Versicolor	Virginica
Action	Setosa	20	0	0
	Versicolor	0	19	0
	Virginica	0	1	20

Figure 7

b) Test accuracy plots for training and test data are as follows.

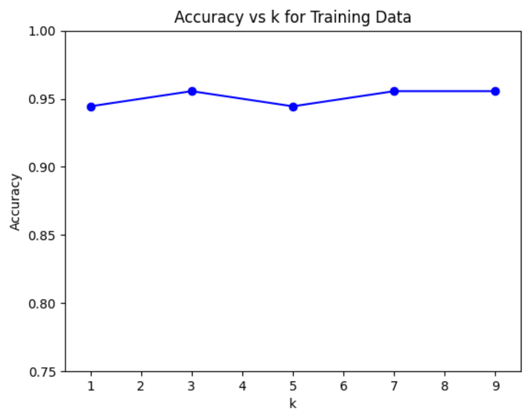


Figure 8

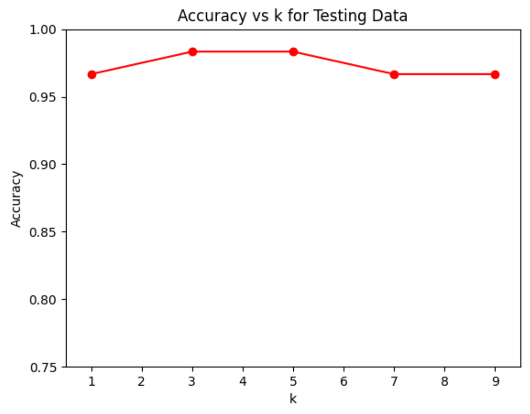


Figure 9

Observations

In both testing and training data, instances that are marked as setosa have very unique values of width and length. Furthermore, they are intact. Therefore, for both k-nearest neighbors and nearest mean classification, there were not any misclassifications.

On the other hand, versicolor and virginica instances are not very distinct from each other in the training data, where petal length is between 4.5 - 5.0 and petal width is 1.5 - 1.7. In that region, the nearest mean classifier falsely assigned 5 of the virginicas to versicolor type. This behaviour was also observed in test data, but the accuracy was higher since there were not much outlier instances in the test data.

In part two, when k=1, two instances in the training data misguided the nearest neighbor classifier to assign them to each other's class. Increasing the k increased the accuracy of the classification for k=3 and k=5, but did not have any effect for k=7 and k=9. Overall, the accuracy was around 95% which I think is reasonable considering that only 2 attributes are used.

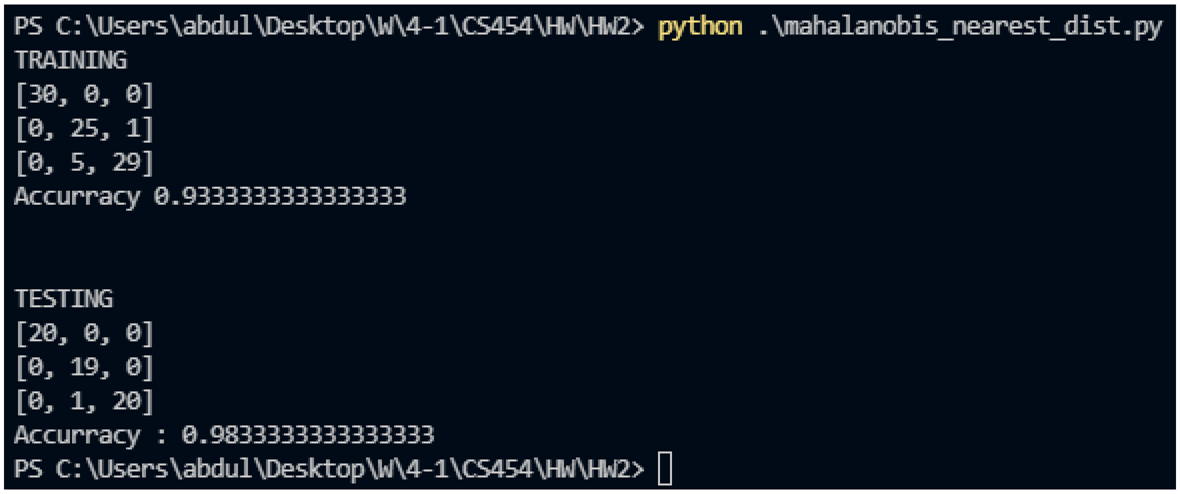
What I learned

- It is important to know what k value to choose in k nearest neighbor algorithm. If the k value is small, then neighbors force the algorithm to make false decisions. If the value is large, then the integrity of the groups does not matter. For example, if k = # instances in the dataset, then all instances would be assigned to the class with the largest number of members.
- I noticed that better algorithms on paper may not produce better results for small datasets.

Extra (Mahalanobis Distance)

I utilized Mahalanobis distance formula to classify both training and test data. Algorithm performed better than the euclidean distance in the test data by doing only one misclassification. However, using Mahalanobis distance did not improve the results for the training dataset. Both algorithms made the same number of misclassifications.

Confusion matrices for the training and test data are as follows.



```
PS C:\Users\abdul\Desktop\W\4-1\CS454\HW\HW2> python .\mahalanobis_nearest_dist.py
TRAINING
[30, 0, 0]
[0, 25, 1]
[0, 5, 29]
Accurracy 0.9333333333333333


TESTING
[20, 0, 0]
[0, 19, 0]
[0, 1, 20]
Accurracy : 0.9833333333333333
PS C:\Users\abdul\Desktop\W\4-1\CS454\HW\HW2>
```

Figure 10

What I learned

- By using Mahalanobis distance, we find the distance of one point to the distribution. In euclidean distance, we calculated the distance between two points. Mahalanobis distance takes both correlation between two variables and the scale difference into account. Therefore, it is better to use Mahalanobis when the weights of variables are different or there is a correlation between two variables.
- I learned that Mahalanobis distance can be used in k-Nearest neighbor algorithm. I couldn't find time to implement that code.

P.S. I will make the below code available after the submission deadline. It requires a lot of effort to extract the code from the pdf then run it. Sorry...
Link : https://github.com/Saydemr/CS454/tree/master/HW2

```
import csv
import matplotlib.pyplot as plt

lookup = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

def read_csv_file(file_name):
    """
    Reads a csv file after second line and returns list of dictionaries with the dat
a. Initial values of predictions are -1
    """
    numbers = []
    with open(file_name, 'r') as csv_file:
        csv_reader = csv.reader(csv_file)
        next(csv_reader)
        for item in list(csv_reader):
            my_dict = { 'petal_length': float(item[0]), 'petal_width': float(item[1]
) , 'species' : lookup.index(item[2]), 'prediction' : -1}
            numbers.append(my_dict)
    return numbers


def sum_numbers(numbers):
    """
    Sums all the numbers in a list
    """
    sum = 0.0
    for number in numbers:
        sum += float(number)
    return sum


def my_mean(numbers):
    """
    Calculates the mean of a list of numbers
    """
    return sum_numbers(numbers) / float(len(numbers))


def euclidian_distance(fx,fy,sx,sy):
    """
    Calculates the euclidian distance between two points
    """
    return (((fx-sx)**2 + (fy-sy)**2)**0.5)


training_data = read_csv_file('training.csv')

length_mean_setosa = my_mean([instance['petal_length'] for instance in training_data
 if instance['species'] == 0])
length_mean_versicolor = my_mean([instance['petal_length'] for instance in training_
data if instance['species'] == 1])
length_mean_virginica = my_mean([instance['petal_length'] for instance in training_d
ata if instance['species'] == 2])

width_mean_setosa = my_mean([instance['petal_width'] for instance in training_data i
f instance['species'] == 0])
width_mean_versicolor = my_mean([instance['petal_width'] for instance in training_da
ta if instance['species'] == 1])
width_mean_virginica = my_mean([instance['petal_width'] for instance in training_dat
a if instance['species'] == 2])

mean_setosa = [length_mean_setosa, width_mean_setosa]
mean_versicolor = [length_mean_versicolor, width_mean_versicolor]
mean_virginica = [length_mean_virginica, width_mean_virginica]

confusion_matrix = [[0,0,0],[0,0,0],[0,0,0]]

for flower in training_data:
    euclidian_setosa = euclidian_distance(flower['petal_length'], flower['petal_widt
h'], mean_setosa[0], mean_setosa[1])
    euclidian_versicolor = euclidian_distance(flower['petal_length'], flower['petal_
width'], mean_versicolor[0], mean_versicolor[1])
    euclidian_virginica = euclidian_distance(flower['petal_length'], flower['petal_w
idth'], mean_virginica[0], mean_virginica[1])

    if euclidian_setosa < euclidian_versicolor and euclidian_setosa < euclidian_virg
inica:
        flower['prediction'] = 0
    elif euclidian_versicolor < euclidian_setosa and euclidian_versicolor < euclidia
n_virginica:
        flower['prediction'] = 1
    elif euclidian_virginica < euclidian_setosa and euclidian_virginica < euclidian_
versicolor:
        flower['prediction'] = 2

    confusion_matrix[flower['prediction']][flower['species']] += 1


testing_data = read_csv_file('testing.csv')
confusion_matrix_test = [[0,0,0],[0,0,0],[0,0,0]]

for flower in testing_data:
    euclidian_setosa = euclidian_distance(flower['petal_length'], flower['petal_widt
h'], mean_setosa[0], mean_setosa[1])
    euclidian_versicolor = euclidian_distance(flower['petal_length'], flower['petal_
width'], mean_versicolor[0], mean_versicolor[1])
    euclidian_virginica = euclidian_distance(flower['petal_length'], flower['petal_w
idth'], mean_virginica[0], mean_virginica[1])

    if euclidian_setosa < euclidian_versicolor and euclidian_setosa < euclidian_virg
inica:
        flower['prediction'] = 0
    elif euclidian_versicolor < euclidian_setosa and euclidian_versicolor < euclidia
n_virginica:
        flower['prediction'] = 1
    elif euclidian_virginica < euclidian_setosa and euclidian_virginica < euclidian_
versicolor:
        flower['prediction'] = 2

    confusion_matrix_test[flower['prediction']][flower['species']] += 1


print('Confusion matrix for training data:', *confusion_matrix,sep='\n', end='\n\n')
print('Confusion matrix for testing data:', *confusion_matrix_test,sep='\n', end='\n
\n')


plt.title("Iris Dataset - Training")
plt.scatter([x['petal_length'] for x in training_data if x['species']==0] , [x['peta
l_width'] for x in training_data if x['species']==0], c='r', label='Iris-Setosa', ma
rker="x")
plt.scatter([y['petal_length'] for y in training_data if y['species']==1] , [y['peta
l_width'] for y in training_data if y['species']==1], c='g', label='Iris-Versicolor'
, marker="o")
plt.scatter([z['petal_length'] for z in training_data if z['species']==2] , [z['peta
l_width'] for z in training_data if z['species']==2], c='b', label='Iris-Virginica',
 marker="+")

plt.scatter(mean_setosa[0], mean_setosa[1], c='c', marker='^', label='Mean Setosa')
plt.scatter(mean_versicolor[0], mean_versicolor[1], c='m', marker='d', label='Mean V
ersicolor')
plt.scatter(mean_virginica[0], mean_virginica[1], c='y', marker='p', label='Mean Vir
ginica')

plt.legend(loc='upper left')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()


plt.title("Iris Dataset - Testing")
plt.scatter([x['petal_length'] for x in testing_data if x['species']==0] , [x['petal
_width'] for x in testing_data if x['species']==0], c='r', label='Iris-Setosa', mark
er="x")
plt.scatter([y['petal_length'] for y in testing_data if y['species']==1] , [y['petal
_width'] for y in testing_data if y['species']==1], c='g', label='Iris-Versicolor',
marker="o")
plt.scatter([z['petal_length'] for z in testing_data if z['species']==2] , [z['petal
_width'] for z in testing_data if z['species']==2], c='b', label='Iris-Virginica', m
arker="+")

plt.scatter(mean_setosa[0]    , mean_setosa[1]    , c='c', marker='^', label='Mean S
etosa')
plt.scatter(mean_versicolor[0], mean_versicolor[1], c='m', marker='d', label='Mean V
ersicolor')
plt.scatter(mean_virginica[0] , mean_virginica[1] , c='y', marker='p', label='Mean V
irginica')

plt.legend(loc='upper left')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```

```
import csv
import matplotlib.pyplot as plt
import math

lookup = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

def read_csv_file(file_name):
    """
    Reads a csv file after second line and returns list of dictionaries with the dat
a. Initial values of predictions are -1
    """
    numbers = []
    with open(file_name, 'r') as csv_file:
        csv_reader = csv.reader(csv_file)
        next(csv_reader)
        for item in list(csv_reader):
            my_dict = { 'petal_length': float(item[0]), 'petal_width': float(item[1]
) , 'species' : lookup.index(item[2]), 'prediction' : -1, 'nearest_nine_index' :[]}
            numbers.append(my_dict)
    return numbers


def sum_numbers(numbers):
    """
    Sums all the numbers in a list
    """
    sum = 0.0
    for number in numbers:
        sum += float(number)
    return sum


def my_mean(numbers):
    """
    Calculates the mean of a list of numbers
    """
    return sum_numbers(numbers) / float(len(numbers))


def euclidian_distance(fx,fy,sx,sy):
    """
    Calculates the euclidian distance between two points
    """
    return (((float(fx)-float(sx))**2 + (float(fy)-float(sy))**2.0)**0.5)


def k_nearest_neighbor_types(k, flower, data):
    """
    Returns the species of k nearest neighbors of a flower
    """
    species = []
    for i in flower['nearest_nine_index']:
        species.append(data[i]['species'])
    return species[:k]

def my_mode(numbers):
    """
    Returns the most common element in a list
    """
    return max(set(numbers), key=numbers.count)


print('---Training---')
training_data = read_csv_file('training.csv')


adjacency_matrix_training = [[0 for j in range(len(training_data))] for i in range(l
en(training_data))]
accuracy_training = []


for i in range(len(training_data)):
    for j in range(len(training_data)):
        adjacency_matrix_training[i][j] = (euclidian_distance(training_data[i]['peta
l_length'],training_data[i]['petal_width'],training_data[j]['petal_length'],training
_data[j]['petal_width']) if i != j else math.inf)

    training_data[i]['nearest_nine_index'] = sorted(range(len(adjacency_matrix_train
ing[i])), key=lambda k: adjacency_matrix_training[i][k])[:9]

for k in range(1,10,2):
    confusion_matrix_training = [[0,0,0],[0,0,0],[0,0,0]]
    for flower in training_data:
        flower['prediction'] = my_mode(k_nearest_neighbor_types(k, flower, training_
data))
        confusion_matrix_training[flower['prediction']][flower['species']] += 1

    print('k = ', k, end='\n')
    #print('Confusion Matrix', *confusion_matrix_training, sep='\n',end='\n\n')
    print('Accuracy: ', (confusion_matrix_training[0][0] + confusion_matrix_training
[1][1] + confusion_matrix_training[2][2]) / float(sum_numbers([sum_numbers(x) for x
in confusion_matrix_training])),end='\n')
    accuracy_training.append((confusion_matrix_training[0][0] + confusion_matrix_tra
ining[1][1] + confusion_matrix_training[2][2]) / float(sum_numbers([sum_numbers(x) f
or x in confusion_matrix_training])))


plt.plot(range(1,10,2), accuracy_training, 'b-', marker='o')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title('Accuracy vs k for Training Data')
plt.axis([0.5,9.5,0.75,1])
plt.show()









print('\n\n---Testing---')
test_data = read_csv_file('testing.csv')

adjacency_matrix_test = [[0 for j in range(len(training_data))] for i in range(len(t
est_data))]
accuracy_test = []

for i in range(len(test_data)):
    for j in range(len(training_data)):
        adjacency_matrix_test[i][j] = euclidian_distance(test_data[i]['petal_length'
],test_data[i]['petal_width'],training_data[j]['petal_length'],training_data[j]['pet
al_width'])

    test_data[i]['nearest_nine_index'] = sorted(range(len(adjacency_matrix_test[i]))
, key=lambda k: adjacency_matrix_test[i][k])[:9]

for k in range(1,10,2):
    confusion_matrix_test = [[0,0,0],[0,0,0],[0,0,0]]
    for flower in test_data:
        flower['prediction'] = my_mode(k_nearest_neighbor_types(k, flower, training_
data))
        confusion_matrix_test[flower['prediction']][flower['species']] += 1

    accuracy_test.append((confusion_matrix_test[0][0] + confusion_matrix_test[1][1]
+ confusion_matrix_test[2][2]) / float(sum_numbers([sum_numbers(x) for x in confusio
n_matrix_test])))

    print('k = ', k, end='\n')
    print('Accuracy: ', (confusion_matrix_test[0][0] + confusion_matrix_test[1][1] +
 confusion_matrix_test[2][2]) / float(sum_numbers([sum_numbers(x) for x in confusion
_matrix_test])),end='\n')
    print('Confusion Matrix', *confusion_matrix_test, sep='\n',end='\n\n')


plt.plot(range(1,10,2), accuracy_test, 'r-', marker='o')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.title('Accuracy vs k for Testing Data')
plt.axis([0.5,9.5,0.75,1])
plt.show()
```

```
import csv
import matplotlib.pyplot as plt

lookup = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']

def read_csv_file(file_name):
    """
    Reads a csv file after second line and returns list of dictionaries with the dat
a. Initial values of predictions are -1
    """
    numbers = []
    with open(file_name, 'r') as csv_file:
        csv_reader = csv.reader(csv_file)
        next(csv_reader)
        for item in list(csv_reader):
            my_dict = { 'petal_length': float(item[0]), 'petal_width': float(item[1]
) , 'species' : lookup.index(item[2]), 'prediction' : -1}
            numbers.append(my_dict)
    return numbers


def sum_numbers(numbers):
    """
    Sums all the numbers in a list
    """
    sum = 0.0
    for number in numbers:
        sum += float(number)
    return sum


def my_mean(numbers):
    """
    Calculates the mean of a list of numbers
    """
    return sum_numbers(numbers) / float(len(numbers))


def my_transpose(matrix):
    """
    Transposes a matrix
    """
    return [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0])
)]


def matrix_multiplication(m1, m2):
    """
    Multiplies two matrices
    """
    result = [[sum_numbers(a*b for a,b in zip(row, col)) for col in zip(*m2)] for ro
w in m1]
    return result

def inverse_two_by_two(matrix):
    """
    Calculates the inverse of a 2x2 matrix
    """
    det = matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]
    return [[matrix[1][1] / det, -matrix[0][1] / det], [-matrix[1][0] / det, matrix[
0][0] / det]]


training_data = read_csv_file('training.csv')

length_mean_setosa = my_mean([instance['petal_length'] for instance in training_data
 if instance['species'] == 0])
length_mean_versicolor = my_mean([instance['petal_length'] for instance in training_
data if instance['species'] == 1])
length_mean_virginica = my_mean([instance['petal_length'] for instance in training_d
ata if instance['species'] == 2])

width_mean_setosa = my_mean([instance['petal_width'] for instance in training_data i
f instance['species'] == 0])
width_mean_versicolor = my_mean([instance['petal_width'] for instance in training_da
ta if instance['species'] == 1])
width_mean_virginica = my_mean([instance['petal_width'] for instance in training_dat
a if instance['species'] == 2])

mean_setosa = [length_mean_setosa, width_mean_setosa]
mean_versicolor = [length_mean_versicolor, width_mean_versicolor]
mean_virginica = [length_mean_virginica, width_mean_virginica]


c_s  = [ [x['petal_length'] - mean_setosa[0] for x in training_data if x['species']
== 0] ,
         [x['petal_width'] - mean_setosa[1]  for x in training_data if x['species']
== 0] ]
c_ve = [ [x['petal_length'] - mean_versicolor[0] for x in training_data if x['specie
s'] == 1] ,
         [x['petal_width'] - mean_versicolor[1]  for x in training_data if x['specie
s'] == 1] ]

c_vi = [ [x['petal_length'] - mean_virginica[0] for x in training_data if x['species
'] == 2] ,
         [x['petal_width'] - mean_virginica[1]  for x in training_data if x['species
'] == 2] ]

covariance_setosa     = matrix_multiplication(c_s, my_transpose(c_s))
covariance_setosa_1   = [[ x / 30.0 for x in covariance_setosa[i]] for i in range(le
n(covariance_setosa))]
covariance_versicolor = matrix_multiplication(c_ve, my_transpose(c_ve))
covariance_versicolor_1 = [[ x / 30.0 for x in covariance_versicolor[i]] for i in ra
nge(len(covariance_versicolor))]
covariance_virginica  = matrix_multiplication(c_vi, my_transpose(c_vi))
covariance_virginica_1  = [[ x / 30.0 for x in covariance_virginica[i]] for i in ran
ge(len(covariance_virginica))]

covariance_setosa_i     = inverse_two_by_two(covariance_setosa_1)
covariance_versicolor_i = inverse_two_by_two(covariance_versicolor_1)
covariance_virginica_i  = inverse_two_by_two(covariance_virginica_1)


confusion_matrix = [[0,0,0],[0,0,0],[0,0,0]]
for flower in training_data:
    x_minus_mean_setosa = [[flower['petal_length'] - length_mean_setosa, flower['pet
al_width'] - width_mean_setosa]]
    x_minus_mean_versicolor = [[flower['petal_length'] - length_mean_versicolor, flo
wer['petal_width'] - width_mean_versicolor]]
    x_minus_mean_virginica = [[flower['petal_length'] - length_mean_virginica, flowe
r['petal_width'] - width_mean_virginica]]

    x_minus_mean_setosa_transpose = my_transpose(x_minus_mean_setosa)
    x_minus_mean_versicolor_transpose = my_transpose(x_minus_mean_versicolor)
    x_minus_mean_virginica_transpose = my_transpose(x_minus_mean_virginica)

    mahalanobis_setosa     = matrix_multiplication(matrix_multiplication(x_minus_mea
n_setosa_transpose    , covariance_setosa_i)    , x_minus_mean_setosa)
    mahalanobis_versicolor = matrix_multiplication(matrix_multiplication(x_minus_mea
n_versicolor_transpose, covariance_versicolor_i), x_minus_mean_versicolor)
    mahalanobis_virginica  = matrix_multiplication(matrix_multiplication(x_minus_mea
n_virginica_transpose , covariance_virginica_i) , x_minus_mean_virginica)

    if mahalanobis_setosa < mahalanobis_versicolor and mahalanobis_setosa < mahalano
bis_virginica:
        flower['prediction'] = 0
    elif mahalanobis_versicolor < mahalanobis_setosa and mahalanobis_versicolor < ma
halanobis_virginica:
        flower['prediction'] = 1
    elif mahalanobis_virginica < mahalanobis_setosa and mahalanobis_virginica < maha
lanobis_versicolor:
        flower['prediction'] = 2

    confusion_matrix[flower['prediction']][flower['species']] += 1

print('TRAINING')
print(*confusion_matrix, sep='\n')
print('Accurracy', (confusion_matrix[0][0] + confusion_matrix[1][1] + confusion_matr
ix[2][2]) / len(training_data))
print('\n')

testing_data = read_csv_file('testing.csv')
confusion_matrix_test = [[0,0,0],[0,0,0],[0,0,0]]

for flower in testing_data:
    x_minus_mean_setosa     = [[flower['petal_length'] - length_mean_setosa, flower[
'petal_width'] - width_mean_setosa]]
    x_minus_mean_versicolor = [[flower['petal_length'] - length_mean_versicolor, flo
wer['petal_width'] - width_mean_versicolor]]
    x_minus_mean_virginica  = [[flower['petal_length'] - length_mean_virginica, flow
er['petal_width'] - width_mean_virginica]]

    x_minus_mean_setosa_transpose     = my_transpose(x_minus_mean_setosa)
    x_minus_mean_versicolor_transpose = my_transpose(x_minus_mean_versicolor)
    x_minus_mean_virginica_transpose  = my_transpose(x_minus_mean_virginica)

    mahalanobis_setosa     = matrix_multiplication(matrix_multiplication(x_minus_mea
n_setosa_transpose    , covariance_setosa_i)    , x_minus_mean_setosa)
    mahalanobis_versicolor = matrix_multiplication(matrix_multiplication(x_minus_mea
n_versicolor_transpose, covariance_versicolor_i), x_minus_mean_versicolor)
    mahalanobis_virginica  = matrix_multiplication(matrix_multiplication(x_minus_mea
n_virginica_transpose , covariance_virginica_i) , x_minus_mean_virginica)

    if mahalanobis_setosa < mahalanobis_versicolor and mahalanobis_setosa < mahalano
bis_virginica:
        flower['prediction'] = 0
    elif mahalanobis_versicolor < mahalanobis_setosa and mahalanobis_versicolor < ma
halanobis_virginica:
        flower['prediction'] = 1
    elif mahalanobis_virginica < mahalanobis_setosa and mahalanobis_virginica < maha
lanobis_versicolor:
        flower['prediction'] = 2

    confusion_matrix_test[flower['prediction']][flower['species']] += 1

print('TESTING')
print(*confusion_matrix_test, sep='\n')
print('Accurracy :', (confusion_matrix_test[0][0] + confusion_matrix_test[1][1] + co
nfusion_matrix_test[2][2]) / len(testing_data))
```