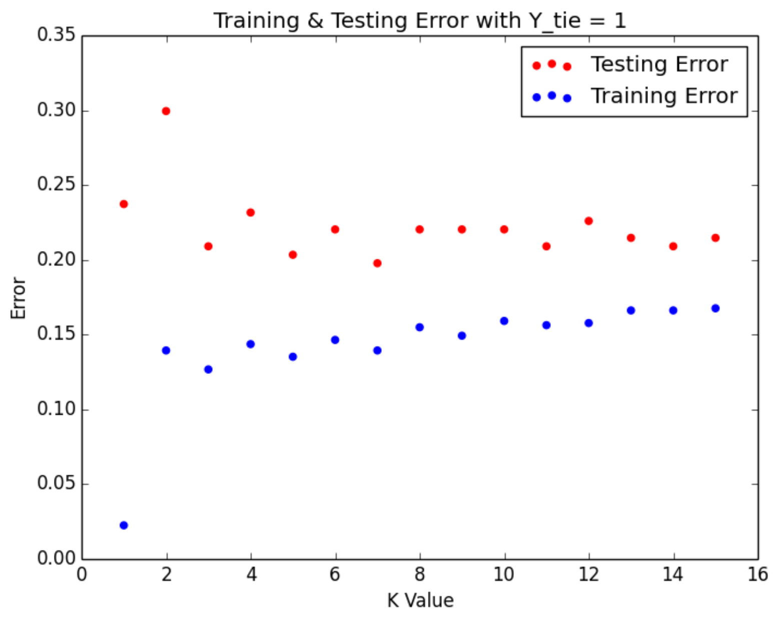
Zack Berger

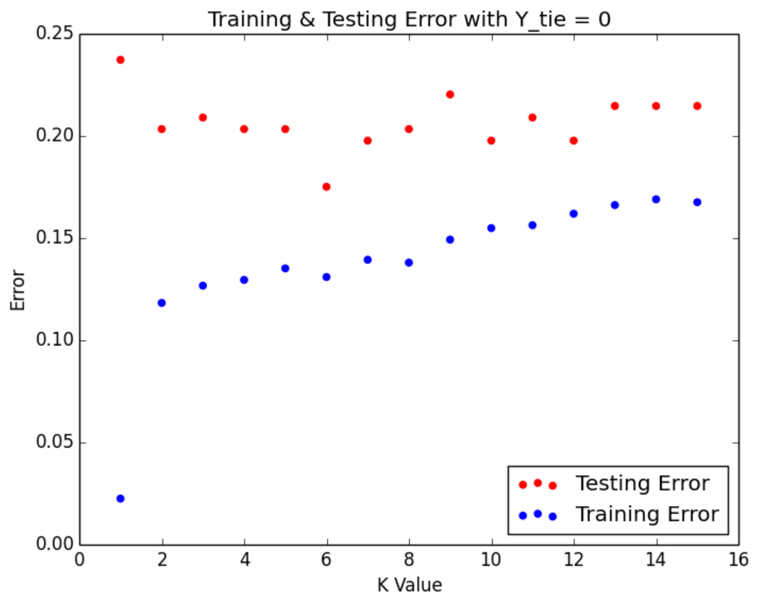
Question 1

Plots for Part A and Part B are here. Attached is the data used to generate these plots...

Part A



Part B



The Data

K	Training Error (y_{tie} = 1)	Testing Error (y_{tie} = 1)	Training Error (y_{tie} = 0)	Testing Error (y_{tie} = 0)
1	0.022535211267605604	0.23728813559322037	0.022535211267605604	0.23728813559322037
2	0.1394366197183099	0.2994350282485876	0.11830985915492953	0.2033898305084746
3	0.12676056338028174	0.20903954802259883	0.12676056338028174	0.20903954802259883
4	0.14366197183098595	0.23163841807909602	0.12957746478873244	0.2033898305084746
5	0.13521126760563384	0.2033898305084746	0.13521126760563384	0.2033898305084746
6	0.14647887323943665	0.22033898305084743	0.1309859154929578	0.17514124293785316
7	0.1394366197183099	0.19774011299435024	0.1394366197183099	0.19774011299435024
8	0.15492957746478875	0.22033898305084743	0.13802816901408455	0.2033898305084746
9	0.14929577464788735	0.22033898305084743	0.14929577464788735	0.22033898305084743
10	0.1591549295774648	0.22033898305084743	0.15492957746478875	0.19774011299435024
11	0.1563380281690141	0.20903954802259883	0.1563380281690141	0.20903954802259883
12	0.15774647887323945	0.22598870056497178	0.1619718309859155	0.19774011299435024
13	0.16619718309859155	0.21468926553672318	0.16619718309859155	0.21468926553672318
14	0.16619718309859155	0.20903954802259883	0.16901408450704225	0.21468926553672318
15	0.1676056338028169	0.21468926553672318	0.1676056338028169	0.21468926553672318

Part C
For the training error in both A and B, as k increases, the error increases on average. In both, the testing error is roughly consistent across each value of k with some notable exceptions, summarized here:

- Part A
    - Average error rate is roughly 0.225
    - k = 2 has an abnormally high error rate at 0.29
- Part B
    - Average error rate is roughly 0.20
    - k = 1 has an abnormally high error rate at 0.24
    - k = 6 has an abnormally low error rate at 0.17

Note that for k = 1 on the training set, the error was roughly zero. This is expected, as each point was compared to itself (unless two points were duplicate, which was rare in the dataset).

Even vs. Odd k values affect the error rates in the following way:

- Part A
    - Training: Typically, even k yielded a higher error rate than odd k (as can be seen in the range k = 4 to k = 9). That said, the difference is not pronounced
    - Testing: By and large, even k yields a higher error rate on average than odd k
- Part B
    - Training: No discernible trend between even vs. odd k
    - Testing: Odd k appears to yield a higher error rate than even k (as in the case of k = 1, k = 3, and k = 9, which have relatively high error rates)

The behavior for the testing data is contradictory. Namely, Part A (y_{tie} = 1) is more erratically plotted, while Part B (y_{tie} = 0) is more smooth. This is especially seen in the training error. Additionally, the overall error for Part B is lower than that in Part A.

Recalling from HW3 Q6, for the titanic training set, the number of 0 labels was 422 out of a total of 710 labels. Hence, it is more likely that a correct prediction would occur with y_{tie} set to 0, as in Part B. This explains the contradictory behavior between part A and B.

Code Attachment

```
#!/usr/bin/env python
"""Analysis of titanic dataset using K-NN algorithm"""

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

# Define class that implements K-NN algorithm
class Knear:
    def __init__(self, trainingData, trainingLabels):
        self.trainingData = trainingData
        self.trainingLabels = trainingLabels

        # If len(trainingData) != len(trainingLabels), throw error
        self.datamt = len(trainingData)

    # Return an array of predicted labels for testing_data
    def test(self, testing_data, k_value, y_tie):

        labels = []
        for i in range(0, len(testing_data)):
            val = self.testPoint(testing_data[i], k_value, y_tie)
            labels.append(val)

        return labels

    # Test an individual point and return its label
    def testPoint(self, point, k_value, y_tie):

        # Obtain array of Euclidian distances from point to training set
        distances = np.array([])
        for i in range(0, self.datamt):
            dist = euclid_six_distance(point, self.trainingData[i])
            distances = np.append(distances, dist)

        # Obtain the k lowest indices, which are the k nearest neighbors
        # Use mergesort, which is a stable sorting algorithm
        nearest_neighbors = distances.argsort(kind='stable')[:k_value]

        # Count the 1s and 0s in the k nearest neighbors
        ones = 0
        zeros = 0

        for i in range(0, len(nearest_neighbors)):
            if self.trainingLabels[nearest_neighbors[i]] == 1:
                ones += 1
            else:
                zeros += 1
        # Determine the label of the test point
        if ones > zeros:
            return 1
        elif zeros > ones:
            return 0
        else:
            return y_tie


def main():
    # Import all training and testing data
    training_data = np.loadtxt("/Users/zackberger/Desktop/Ml/HW/HW_3/dataTraining_X.csv", delimiter=',')
    actual_training_labels = np.loadtxt("/Users/zackberger/Desktop/Ml/HW/HW_3/dataTraining_Y.csv", delimiter=',')
    testing_data = np.loadtxt("/Users/zackberger/Desktop/Ml/HW/HW_3/dataTesting_X.csv", delimiter=',')
    actual_testing_labels = np.loadtxt("/Users/zackberger/Desktop/Ml/HW/HW_3/dataTesting_Y.csv", delimiter=',')

    # Instantiate K-NN framework with training data
    knn = Knear(training_data, actual_training_labels)

    # With tiebreaker set to 1, find testing error for k = 1, ..., 15
    tiebreaker = 1

    test1 = knn.test(testing_data, 1, tiebreaker)
    err1 = error(test1, actual_testing_labels)
    .
    . # Code omitted for brevity
    .
    test15 = knn.test(testing_data, 15, tiebreaker)
    err15 = error(test15, actual_testing_labels)
    errors = [err1, err2, err3, err4, err5, err6, err7, err8, err9, err10, err11, err12, err13, err14, err15]
    print('Testing Error:')
    print(errors)

    # With tiebreaker set to 1, find training error for k = 1, ..., 15
    . # Code omitted for brevity

    x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
    errorbs = [errb1, errb2, errb3, errb4, errb5, errb6, errb7, errb8, errb9, errb10, errb11, errb12, errb13, errb14,
               errb15]
    print('Training Error:')
    print(errorbs)

    plt.scatter(x, errors, color='r', label='Testing Error')
    plt.scatter(x, errorbs, color='b', label='Training Error')
    plt.title('Training & Testing Error with Y_tie = 1')
    plt.xlabel('K Value')
    plt.ylabel('Error')
    plt.legend(loc="upper right")
    plt.show()
```

```
. # tiebreaker = 0 code omitted for brevity

# Compute percentage of mismatches in two equivalently sized arrays
def error(arr1, arr2):
    matches = 0

    for i in range(0, len(arr1)):
        if arr1[i] == arr2[i]:
            matches += 1

    return 1 - ( float(matches) / len(arr1) )


# Find the Euclidian distance between two points with six attributes
def euclid_six_distance(arr1, arr2):

    return ( (arr1[0] - arr2[0])**2 + (arr1[1] - arr2[1])**2 + (arr1[2] - arr2[2])**2 + (arr1[3] - arr2[3])**2 + (arr1[4] -
arr2[4])**2 + (arr1[5] - arr2[5])**2 )


# Name guard
if __name__ == "__main__":
    main()
```

2] Consider K-NN classifier w/ Euclidean distance on binary classification task. class of test point is majority class of K-NN.

a) For each dataset, do the following:

- Select $K \in \{1, 3, 5, 7\}$
- For all points, leave it out then run K-NN on that point.
- Note how many erroneous classifications there were.
- Validation Error $= \frac{\text{total misclassified}}{\text{total points}}$ (note, total points = 14 for each sets).

Table 1

K	misclassifications	error
1	10	$10/14$
3	6	$6/14$
5	4	$4/14$
7	4	$4/14$

For table 1, K = 5, K = 7 both minimize leave-one-out cross-validation error at error $= \frac{2}{7}$.

For table 2, K = 1 minimizes the error at error $= \frac{1}{7}$.

Resulting validation error is documented in the tables to the left.

Table 2

K	misclassifications	error
1	2	$2/14$
3	4	$4/14$
5	6	$6/14$
7	6	$12/14$

The calculations were done by inspection and as needed the Pythagorean theorem was used to compute distances in table 2.

b) Using K=1 on table 1, error $= \frac{10}{14}$. Using K = 13 on table 1, error $= \frac{14}{14}$. This computation illustrates the following drawbacks.

→ Too big a K: decision boundary is smooth, so decision regions are uninterrupted. Generalizes the data too much, a form of underfitting. Seen here where the predicted label was just the majority class.

→ Too small a K: decision boundaries very fine-grain, could lead to overfitting. Small K doesn't accurately represent the neighbors surrounding a test point.

3| Sum of squares error is $J(\tilde{W}) = \frac{1}{2} \mathrm{Tr}\{(\tilde{X}\tilde{W} - T)^T (\tilde{X}\tilde{W} - T)\}$.
Find closed form solution of $\tilde{W}$ that minimizes $J(\tilde{W})$.

$\hookrightarrow \partial/\partial Z \, \mathrm{Tr}(Z^T A) = A \quad , \quad \partial/\partial Z \, \mathrm{Tr}(Z^T A Z) = (A^T + A) Z$.

Expanding, $J(\tilde{W}) = \frac{1}{2} \mathrm{Tr}\{(\tilde{W}^T \tilde{X}^T - T^T)(\tilde{X}\tilde{W} - T)\}$

$= \frac{1}{2} \mathrm{Tr}\{\tilde{W}^T \tilde{X}^T \tilde{X} \tilde{W} - \tilde{W}^T \tilde{X}^T T - T^T \tilde{X} \tilde{W} + T^T T\}$

$= \frac{1}{2} \mathrm{Tr}\{\tilde{W}^T \tilde{X}^T \tilde{X} \tilde{W} - 2\tilde{W}^T \tilde{X}^T T + T^T T\}$

b/c $\tilde{W}^T \tilde{X}^T T + T^T \tilde{X} \tilde{W} = 2 \tilde{W}^T \tilde{X}^T T$,

as $(\tilde{W}^T \tilde{X}^T T)^T = T^T \tilde{X} \tilde{W}$ and the matrix is square $(K \times K)$.

Then, $\partial/\partial \tilde{W} \, J(\tilde{W}) = \partial/\partial \tilde{W} \left(\frac{1}{2} \mathrm{Tr}\{\tilde{W}^T \tilde{X}^T \tilde{X} \tilde{W} - 2\tilde{W}^T \tilde{X}^T T + T^T T\} \right)$

$= \frac{1}{2} \frac{\partial}{\partial \tilde{W}} \mathrm{Tr}\{\tilde{W}^T \tilde{X}^T \tilde{X} \tilde{W}\} - \partial/\partial \tilde{W} \, \mathrm{Tr}\{\tilde{W}^T \tilde{X}^T T\} + \cancelto{0}{\frac{1}{2} \partial/\partial \tilde{W} \, \mathrm{Tr}\{T^T T\}}$

$= \frac{1}{2} \left((\tilde{X}^T \tilde{X})^T + (\tilde{X}^T \tilde{X}) \right) \tilde{W} - \tilde{X}^T T$

$= \frac{1}{2} \left(\tilde{X}^T \tilde{X} + \tilde{X}^T \tilde{X} \right) \tilde{W} - \tilde{X}^T T$

$= \tilde{X}^T \tilde{X} \tilde{W} - \tilde{X}^T T$

Setting $\frac{\partial J(\tilde{W})}{\partial \tilde{W}}$ equal to 0, obtain $\tilde{X}^T \tilde{X} \tilde{W} - \tilde{X}^T T = 0$

$\longrightarrow \tilde{X}^T \tilde{X} \tilde{W} = \tilde{X}^T T$

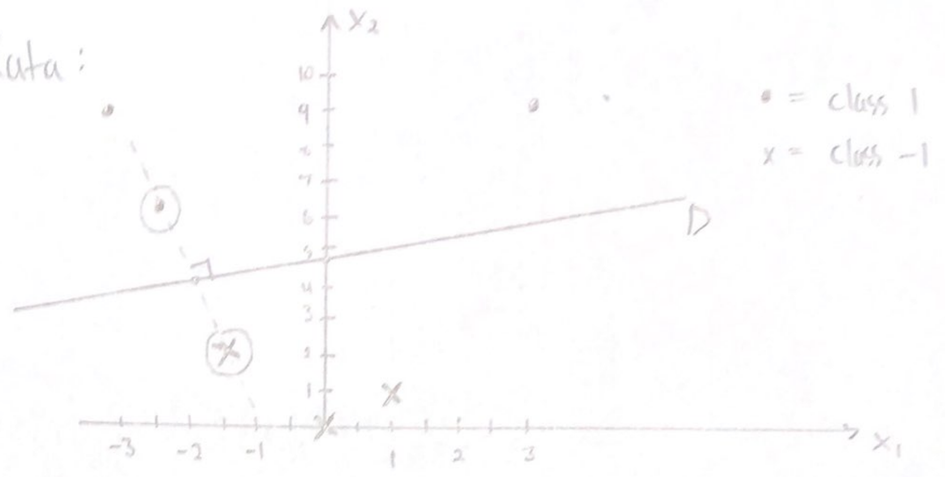
$\longrightarrow \tilde{W} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T T$ if $(\tilde{X}^T \tilde{X})$ is invertible.

The closed form solution of $\tilde{W}$ that minimizes $J(\tilde{W})$ is

$$\tilde{W} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T T$$

4) Consider the dataset on HW 4.

a) Plot of the data:



⟹ Evidently, the data is linearly separable.

b) By inspection, the support vectors are datapoints 2 and 4. They are circled in the above plot.

The maximum margin separating hyperplane satisfies
- intersects at the midpoint of the line that connects the two support vectors.
- is perpendicular to the connecting line.

It is graphed above and denoted D. $\boxed{D \text{ described by } x_2 = \frac{1}{4}x_1 + 4.75}$

computations...

→ 2 datapoints, $(-1.5, 2.25)$ and $(-2.5, 6.25)$. Slope of connecting line given by $m = \frac{6.25 - 2.25}{-2.5 + 1.5} = -4$.

→ Midpoint of two support vectors given by $\left(\frac{-1.5 + (-2.5)}{2}, \frac{(2.25) + (6.25)}{2}\right)$
$= (-2, 4.25)$

→ D is described by line with slope $\left(\frac{1}{4}\right)$ passing through $(-2, 4.25)$.
⟹ $4.25 = \frac{1}{4}(-2) + b \Rightarrow b = 4.75$

Note: the graph above appears as though D is not perpendicular to the dashed line, but that is just because the x_1, x_2 axes are scaled differently.

c) Dual form of quadratic program: $\mathcal{L} = \sum_{n=1}^{6} a_n - \frac{1}{2}\sum_{n=1}^{6}\sum_{k=1}^{6} a_n t_n a_k t_k x_n^T x_k$

subject to $a_n \geq 0$, $\sum_{n=1}^{6} a_n t_n = 0$

Note that there are only 2 support vectors, so only two terms with $a_n \neq 0$. Then, $\mathcal{L} = (a_2 + a_4) - \frac{1}{2}(a_2^2 \|x_2\|^2 + a_4^2 \|x_4\|^2 - 2a_2 a_4 x_2^T x_4)$.

Because $\sum_{n \in S} a_n t_n = a_2 t_2 + a_4 t_4 = 0$ w/ $a_2, a_4 > 0$, $a_2 = a_4$.

Let $a = a_2 = a_4$. Then, $\mathcal{L} = 2a - \frac{1}{2}(a^2\|x_2\|^2 + a^2\|x_4\|^2 - 2a^2 x_2^T x_4)$

$\hookrightarrow \|x_2\|^2 = (-2.5)^2 + (6.25)^2 = 45.3125$ $\quad x_2^T x_4 = (-2.5)(-1.5) + (6.25)(2.25) = 17.8125$

$\|x_4\|^2 = (-1.5)^2 + (2.25)^2 = 7.3125$

$\Rightarrow \mathcal{L} = 2a - \frac{1}{2}(a^2 52.625 - 2 \cdot 17.8125 a^2) = 2a - 8.5a^2$

Maximizing $\mathcal{L}$ wrt a, $\frac{\partial \mathcal{L}}{\partial a} = \frac{\partial}{\partial a}(2a - 8.5a^2) = 0 \Rightarrow 2 - 17a = 0 \Rightarrow a = \frac{2}{17}$

Noting that $a = \frac{2}{17}$, we can find w and b.

$\hookrightarrow w = \sum_{n \in S} a_n t_n x_n = \left(\frac{2}{17}\right)(1)\begin{bmatrix}-2.5 \\ 6.25\end{bmatrix} + \left(\frac{2}{17}\right)(-1)\begin{bmatrix}-1.5 \\ 2.25\end{bmatrix} = \begin{bmatrix}-2/17 \\ 8/17\end{bmatrix}$

using x_2, $b = \frac{1}{t_n} - w^T x_2 = \frac{1}{1} - \begin{bmatrix}-2/17 & 8/17\end{bmatrix}\begin{bmatrix}-2.5 \\ 6.25\end{bmatrix} = -\frac{38}{17}$

So the answers are:

$$\boxed{\begin{array}{l} a_1 = a_3 = a_5 = a_6 = 0 \\ a_2 = a_4 = 2/17 \\ w = \begin{bmatrix}-2/17 \\ 8/17\end{bmatrix}, \quad b = -\frac{38}{17} \end{array}}$$

The decision boundary is given by $w^T x + b = -\frac{2}{17}x_1 + \frac{8}{17}x_2 - \frac{38}{17} = 0$

$\Rightarrow D$ is $\frac{8}{17}x_2 = \frac{2}{17}x_1 + \frac{38}{17} \Rightarrow D$ is $x_2 = \frac{1}{4}x_1 + 4.75$

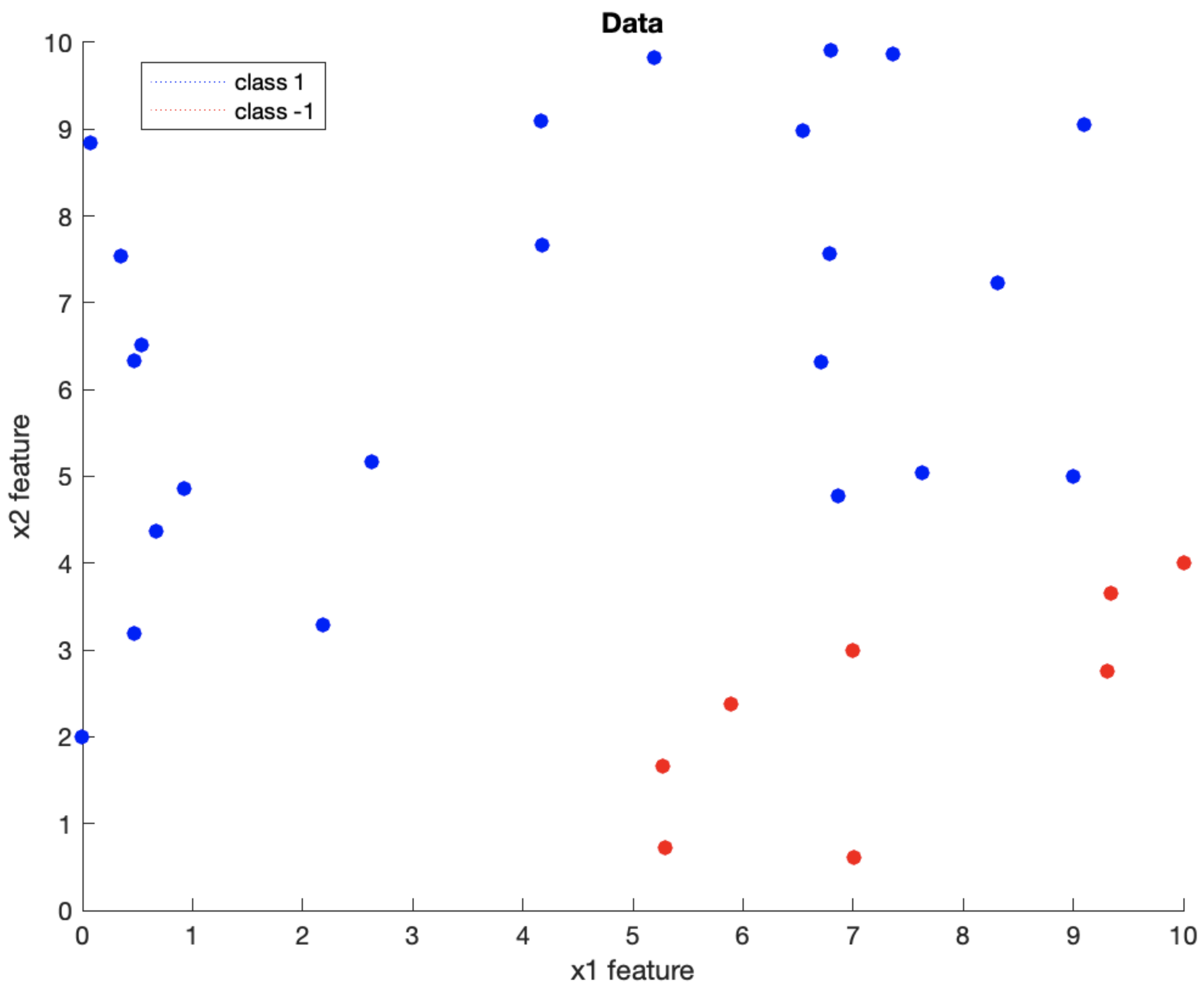
i.e. $\boxed{\text{w, b parameterize the same decision boundary as in Pt. b}}$

Zack Berger

Question 5

Part A - Visualization

By inspection of the following plot, the data is linearly separable…

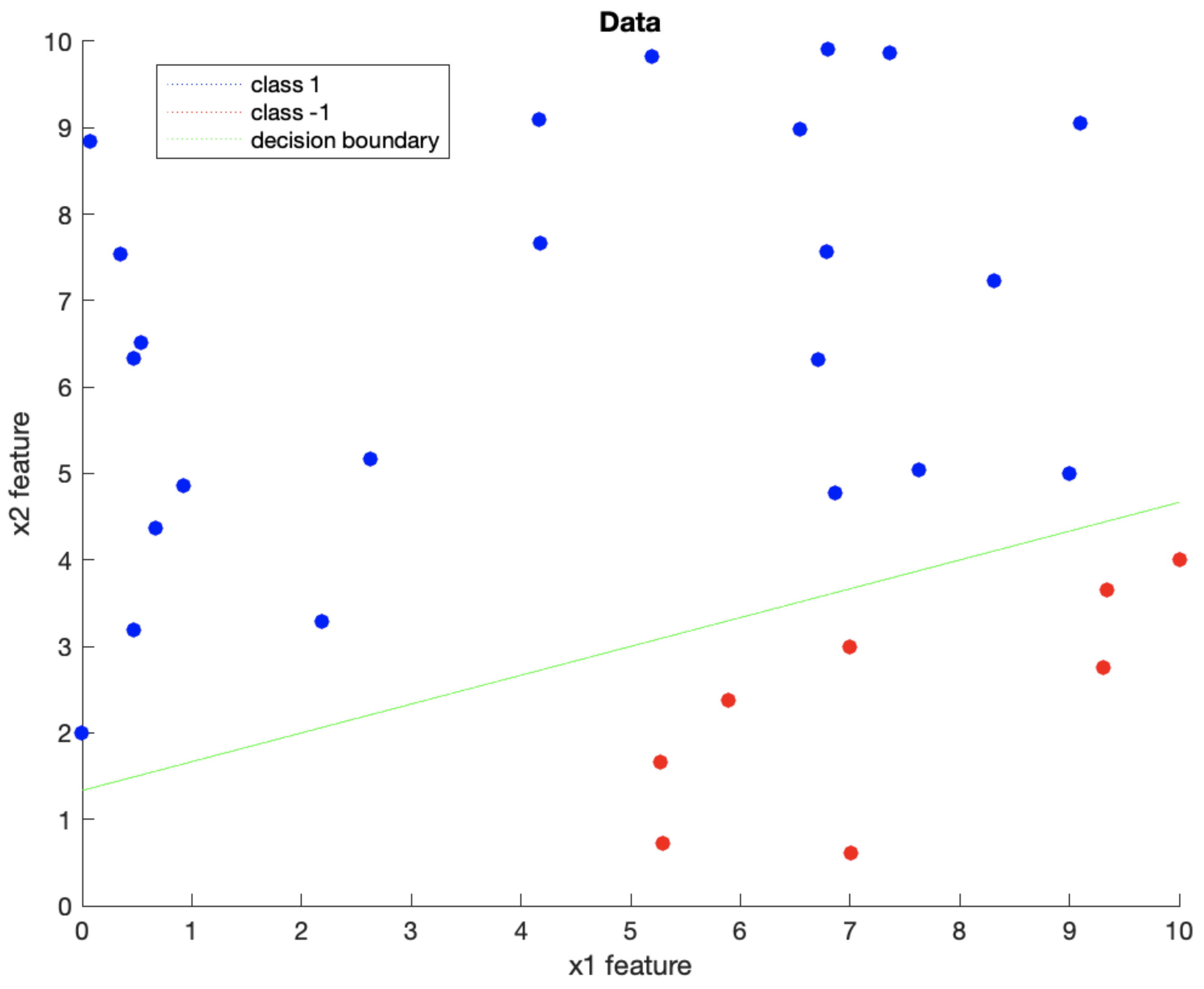


Part B

Using CVX to solve the primal problem for SVM, I found:

w = [-0.5, 1.5]
b = -2.0

The following shows a plot of the hyperplane defined by w and b overlaid with the data…



Part C

Suppose there are N data points that have feature vectors of dimension M.

As evidenced by the attached scratchwork that investigates the case of N = 3, the latter part of W(a) in the dual problem can be decomposed as $(\frac{1}{2})a^TPa$ where for the matrix P we have

$$P(i,j) = Y_iY_j\langle X_i,X_j\rangle$$

Making this substitution into MATLAB then applying CVX to solve the dual problem of SVM, I found the following non-zero a's and their corresponding support vectors:

a_{28} = 0.853301
Support vector: [7.000000, 3.000000]

a_{29} = 0.396669
Support vector: [10.000000, 4.000000]

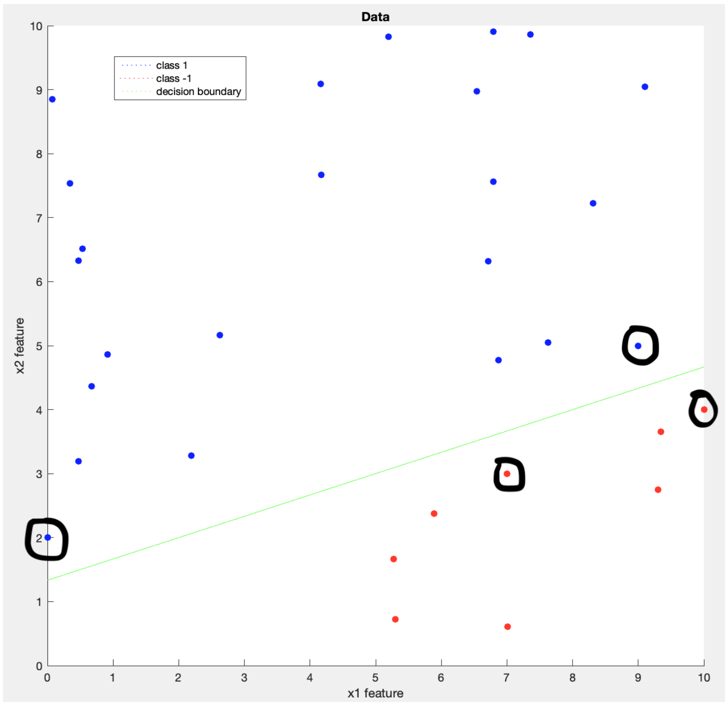
a_{30} = 0.201101
Support vector: [0.000000, 2.000000]

a_{31} = 1.048869
Support vector: [9.000000, 5.000000]

I used these values to calculate w and b, which are reported as follows:

w = [-0.5, 1.5]
b = -2.0

There are 4 support vectors, circled on the following graph…



5c) Finding P using $m=3$ case

Let $a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$, $Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$, $X = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix} = \begin{bmatrix} -x_1- \\ -x_2- \\ -x_3- \end{bmatrix}$

Then, $\sum_{i=1}^{m} \sum_{j=1}^{m} a_i a_j y_i y_j < x_i , x_j >$

$$= a_1 a_1 y_1 y_1 < x_1 , x_1 > + a_1 a_2 y_1 y_2 < x_1 , x_2 > + a_1 a_3 y_1 y_3 < x_1 , x_3 >$$
$$+ a_2 a_1 y_2 y_1 < x_2 , x_1 > + \dots \dots + a_2 a_3 y_2 y_3 < x_2 , x_3 >$$
$$+ \dots \dots + a_3 a_3 y_3 y_3 < x_3 , x_3 >$$

Also, $a^T \underset{P}{\begin{bmatrix} \text{I} & \text{II} & \text{III} \\ \text{IV} & \text{V} & \text{VI} \\ \text{VII} & \text{VIII} & \text{IX} \end{bmatrix}} a = a_1 a_1 \text{I} + a_1 a_2 \text{IV} + a_1 a_3 \text{VII}$
$$+ a_2 a_1 \text{II} + a_2 a_2 \text{V} + a_2 a_3 \text{VIII}$$
$$+ a_3 a_1 \text{III} + a_3 a_2 \text{VI} + a_3 a_3 \text{IX}$$

$$\Rightarrow P = \begin{bmatrix} y_1 y_1 < x_1 , x_1 > & y_1 y_2 < x_1 , x_2 > & y_1 y_3 < x_1 , x_3 > \\ y_2 y_1 < x_2 , x_1 > & \ddots & \\ y_3 y_1 < x_3 , x_1 > & & y_3 y_3 < x_3 , x_3 > \end{bmatrix}$$

$$\Rightarrow \underline{P_{ij} = y_i y_j < x_i , x_j >}$$

Code Attachment

```
% Import data
data = readtable('/Users/zackberger/Desktop/ML/HW/HW_4/Q5/Data.csv');
data = data{:,:};


% Separate data into column vectors
X = data(:,[1,2]);
Y = data(:,3);

x1 = data(:,1);
x2 = data(:,2);

[num_data, data_dim] = size(X);


% Optimize primal SVM problem with CVX
cvx_begin

  variable w_primal(2);
  variable b_primal;

  minimize( norm(w_primal) );

  subject to
    Y.*(X*w_primal + b_primal) >= 1;

cvx_end

w_primal;
b_primal;


% Calculate the matrix P, where P(i,j) = YiYj<Xi,Xj>
P = zeros(num_data,num_data);
for i = 1:num_data
  for j = 1:num_data
    P(i,j) = X(i,:)*X(j,:)'*Y(i)*Y(j);
  end
end

one_vector = ones(num_data, 1);

% Optimize dual SVM problem with CVX
cvx_begin

  variable alpha_dual(num_data);

  minimize( (1/2)*alpha_dual.'*P*alpha_dual - (one_vector.'*alpha_dual) );
```

```
    subject to
        alpha_dual >= 0;
        Y.'*alpha_dual == 0;
cvx_end

% Loop through alpha_dual and for non-zero alpha, print corresponding support vector
num_support_vectors = 0;
for i = 1:num_data
    if alpha_dual(i) > .0001
        fprintf('alpha %i = %f\n', i, alpha_dual(i));
        fprintf('Support vector: [%f, %f]\n\n', X(i,1), X(i,2));
        num_support_vectors = num_support_vectors + 1;
    end
end

% Use alpha_dual to compute w and b
w_dual = [0, 0];
for i = 1:num_data
    if alpha_dual(i) > .0001
        w_dual = w_dual + alpha_dual(i)*X(i,:)*Y(i);
    end
end

b_dual = 0
for i = 1:num_data
    if alpha_dual(i) > .0001
        b_dual = b_dual + (1/num_support_vectors) * (Y(i) - w_dual*X(i,:).');
    end
end

w_dual
b_dual


% Graph results!
hold on

% Scatter plot feature vectors
for i = 1: numel(Y)
    if Y(i) == 1
        scatter(x1(i), x2(i), 'filled', 'b')
    else
        scatter(x1(i), x2(i), 'filled', 'r')
    end
end

% Plot primal decision boundary (w1*x1 + w2*x2 + b = 0)
% x1_axis = 0:1/100:10;
% x2_axis = (-b_primal - w_primal(1)*x1_axis) / w_primal(2);
% plot(x1_axis, x2_axis, 'g');

% Plot dual decision boundary (w1*x1 + w2*x2 + b = 0)
x1_axis = 0:1/100:10;
x2_axis = (-b_dual - w_dual(1)*x1_axis) / w_dual(2);
plot(x1_axis, x2_axis, 'g');

% Create graph details
title("Data");
xlabel("x1 feature");
ylabel("x2 feature");

L(1) = plot(nan, nan, 'b:');
L(2) = plot(nan, nan, 'r:');
L(3) = plot(nan, nan, 'g:');
legend(L, {'class 1', 'class -1', 'decision boundary'})

hold off
```