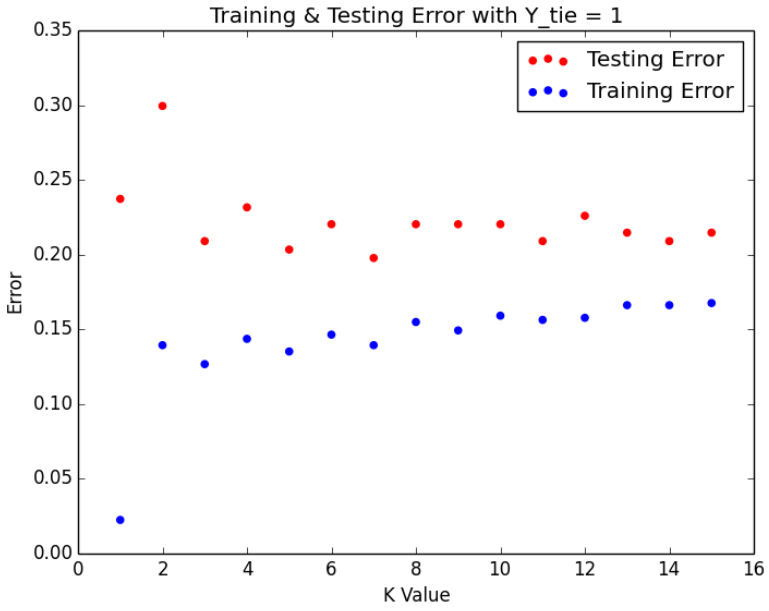
Zack Berger

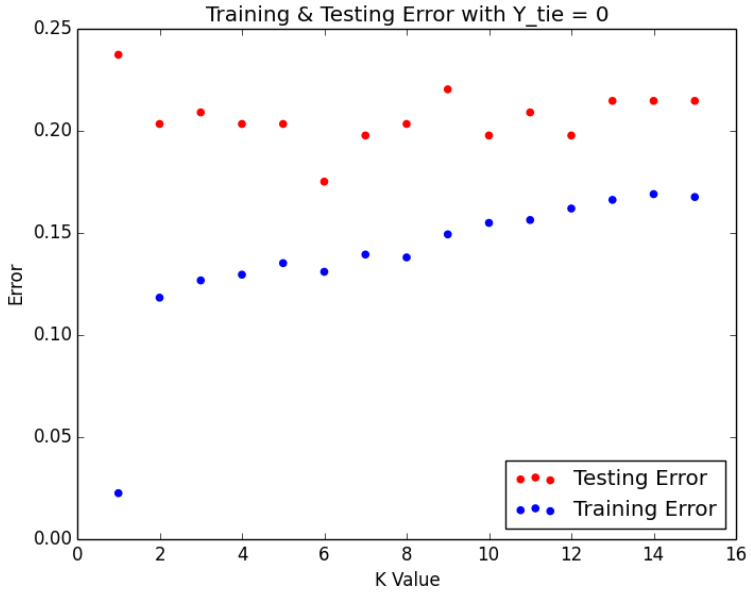
Question 1

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/MI/HW/HW_3/dataTraining_X.csv", delimiter=',')
    actual_training_labels = np.loadtxt("/Users/zackberger/Desktop/MI/HW/HW_3/dataTraining_Y.csv", delimiter=',')
    testing_data = np.loadtxt("/Users/zackberger/Desktop/MI/HW/HW_3/dataTesting_X.csv", delimiter=',')
    actual_testing_labels = np.loadtxt("/Users/zackberger/Desktop/MI/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()
```