HW2_Sol

February 23, 2020

ComS 574

HW 2

Kanak Choudhury

1 Problem 1

We know that

$$\frac{P(y=1 \mid x)}{1-P(y=1 \mid x)} = \beta_0 + \beta_1 x$$

$$\frac{P(y=1 \mid x)}{1-P(y=1 \mid x)} = 1.24 - 3.74x$$

Now, if we consider $x = az$, where x is measured in kilometers, z is measured in miles and $a = 0.6214$, then we write-

$$\begin{aligned} \frac{P(y=1 \mid x)}{1-P(y=1 \mid x)} &= 1.24 - 3.74x && (1) \\ &= 1.24 - 3.74(az) && (2) \\ &= 1.24 - a \times 3.74z && (3) \end{aligned}$$

So, Sammie's estimated β_0 and β_1 will be

$$\hat{\beta}_0 = 1.24$$

and

$$\hat{\beta}_1 = a \times 3.74 = 0.6214 \times 3.74 = 2.324$$

2 Problem 2 (Book problem Chapter 4, #4)

2.1 (a)

We have, X is uniformly distributed on $[0, 1]$. Now,

$$x \in [0.05, 0.95] \text{ represents fraction of } 0.10$$
$$x \in [0, 0.05] \text{ and } x \in [0.95, 1] \text{ represents fraction of } (x + 0.05)$$

So,

$$\int_{0.05}^{0.95} 0.1dx + \int_{0}^{0.05} (x + 0.05)\, dx + \int_{0.95}^{1} (x + 0.05)\, dx = 0.09 + 0.00375 + 0.00375 = 0.0975$$

Thus, on average, about 0.0975 fraction of the available observations will be used to make the prediction.

2.2 (b)

Let, X_1 and X_2 be independent. Then the fraction of the available observations will be used to make the prediction is-

$$0.0975 \times 0.0975 = 0.009506$$

2.3 (c)

If we consider all X_i $i = 1, 2, ..., 100$ are independent, then the fraction of the available observations will be used to make the prediction is

$$0.0975^p = 0.0975^{100} \approx 0 \qquad \text{where } p \text{ is the number of features}$$

2.4 (d)

From the above problem, we see that as the number of features $p \to \infty$ he fraction of the available observations $\to 0$, implies that there are very few training observations "near" any given test observation.

2.5 (e)

For $p = 1, 2$, and 100, the length of each side of the hypercube are-

$$p = 1 : length = 0.1 p = 2 : length = 0.1^{\frac{1}{2}} = 0.31623 p = 100 : length = 0.1^{\frac{1}{100}} = 0.97724$$

3 Problem 3 (Book problem Chapter 4, #6)

3.1 (a)

Estimated probability that a student who studies for 40h and has an undergrad GPA of 3.5 gets an A in the class is -

$$P(y = 1|x) = \frac{exp\left(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2\right)}{1 + exp\left(\hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2\right)}$$

$$P(y = 1|x) = \frac{exp(-6 + 0.05 \times 40 + 3.5)}{1 + exp(-6 + 0.05 \times 40 + 3.5)} = 0.3775$$

3.2 (b)

$$P(y=1|x) = \frac{exp(-6+0.05\times X_1+3.5)}{1+exp(-6+0.05\times X_1+3.5)} = 0.5$$

Hours need to study to have a 50% chance of getting an A in the class is-

$$X_1 = 50$$

4 Problem 4 (Book problem Chapter 4, #8)

For the logistic regression we have an error rate of 20% on the training data and 30% on the test data. And for 1−-nearest neighbors, the average error rate (averaged over both test and training data sets) is 18%. For 1−-nearest neighbors, the trainng error will 0%. Thus, the testing error rate is 36% for the 1−-nearest neighbors. Thus, we should prefer to use **logistic regression classifier** for new observations because it has lower test error, 30%.

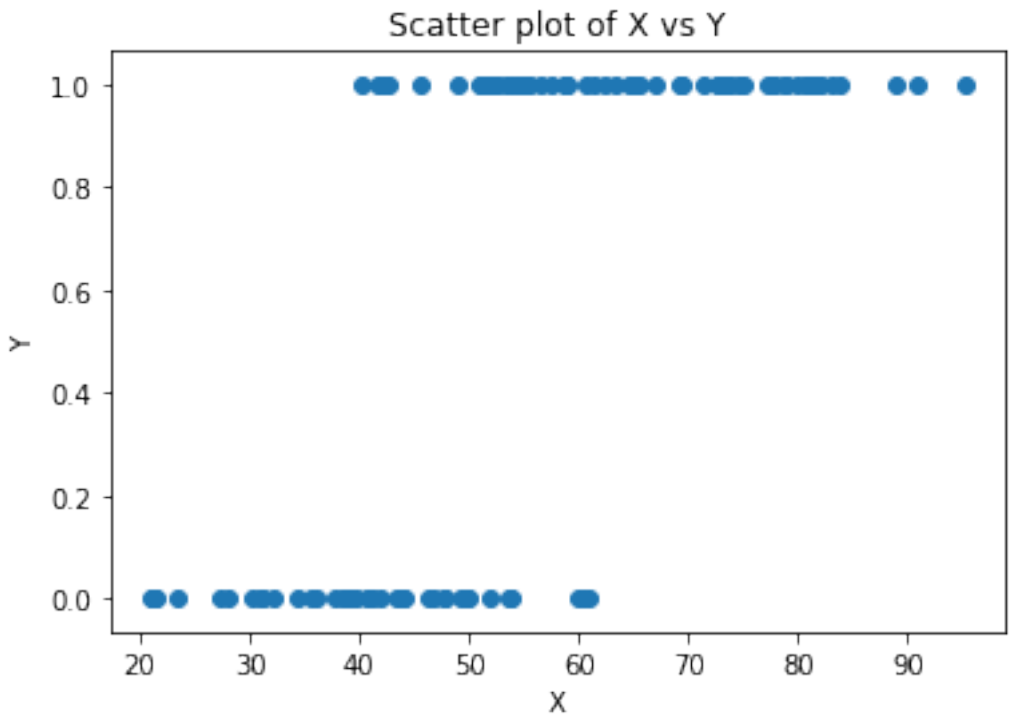
5 Problem 5

[20]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression as lr
from sklearn.neighbors import KNeighborsClassifier as kn
```

5.1 (A)

[21]:
```
df_train = pd.read_csv('D:/ISU/COMS 574 - Introduction to Machine Learning/HW/
 ↪HW2/HW2train.csv', sep=',',
                       header=None, names=["Y", "X"])
df_test = pd.read_csv('D:/ISU/COMS 574 - Introduction to Machine Learning/HW/
 ↪HW2/HW2test.csv', sep=',',
                      header=None, names=["Y", "X"])
tr_size = df_train.shape
ts_size = df_test.shape
```

[22]:
```
plt.scatter(df_train['X'], df_train['Y'])
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Scatter plot of X vs Y")
plt.show()
```

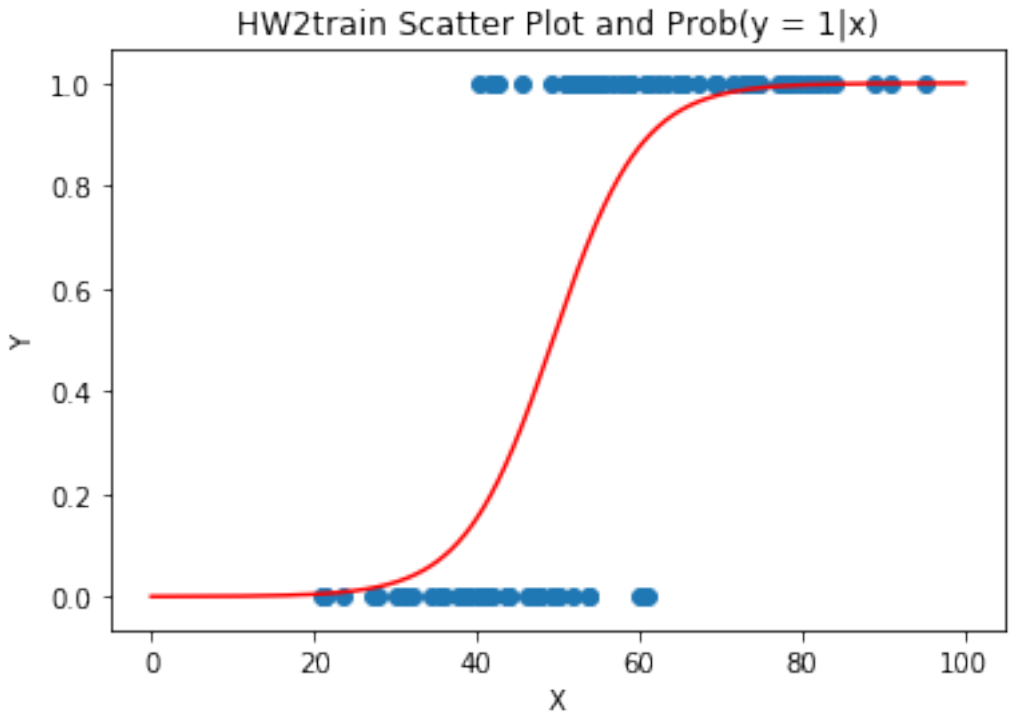


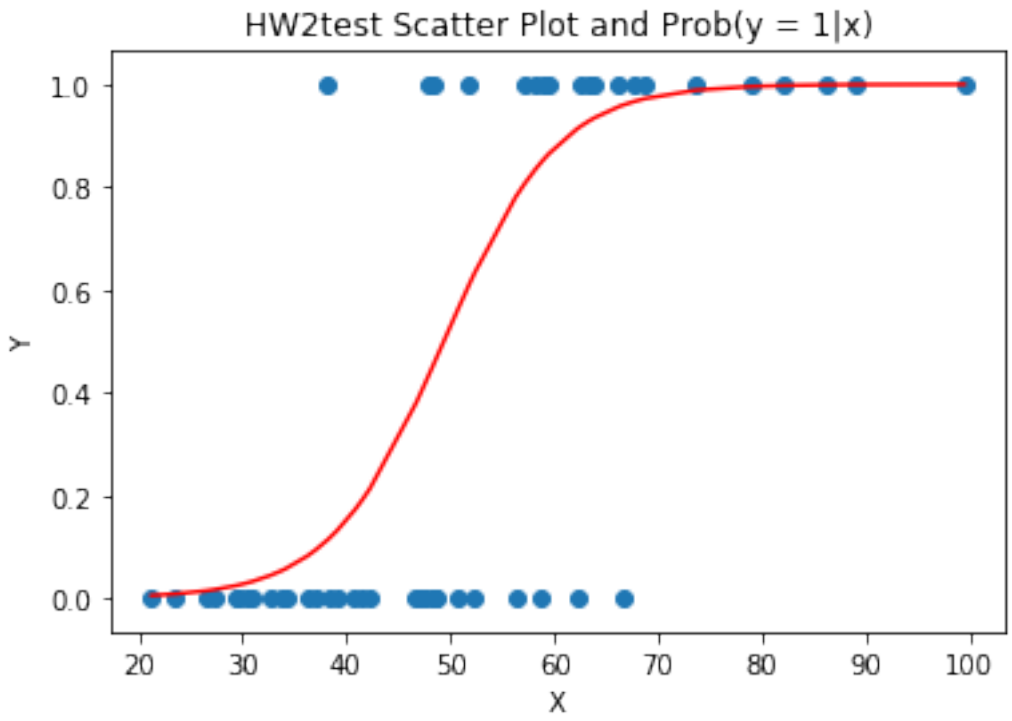
5.2 (B)

[23]:
```
x_train = np.array(df_train[['X']])
y_train = np.array(df_train[['Y']])
model = lr(penalty='none', fit_intercept=True, solver='lbfgs').fit(x_train,␣
 ↪y_train.ravel())

print("beta_0 = %.4f and beta_1 = %.4f" %(model.intercept_, model.coef_))
print("LR Accuracy for HW2train %.2f%%" %(model.score(x_train, y_train)*100))

gen_x = np.linspace(0,100, num = 1000)
prob_gen = model.predict_proba(gen_x.reshape(1000,-1))

ts_x = np.array(df_test['X'])
ts_y = np.array(df_test['Y'])
prob_ts = model.predict_proba(ts_x.reshape(ts_size[0],-1))

print("LR Accuracy for HW2test %.2f%%" %(model.score(ts_x.
 ↪reshape(ts_size[0],-1), ts_y)*100))


plt.scatter(df_train['X'], df_train['Y'])
plt.plot(gen_x, prob_gen[:,1], color = 'red')
```

```
plt.xlabel("X")
plt.ylabel("Y")
plt.title("HW2train Scatter Plot and Prob(y = 1|x)")
plt.show()

plt.scatter(ts_x, ts_y)
plt.plot(ts_x[np.argsort(ts_x)], prob_ts[:,1][np.argsort(ts_x)], color = 'red')
plt.xlabel("X")
plt.ylabel("Y")
plt.title("HW2test Scatter Plot and Prob(y = 1|x)")
plt.show()
```

```
beta_0 = -9.0587 and beta_1 = 0.1835
LR Accuracy for HW2train 86.00%
LR Accuracy for HW2test 82.00%
```





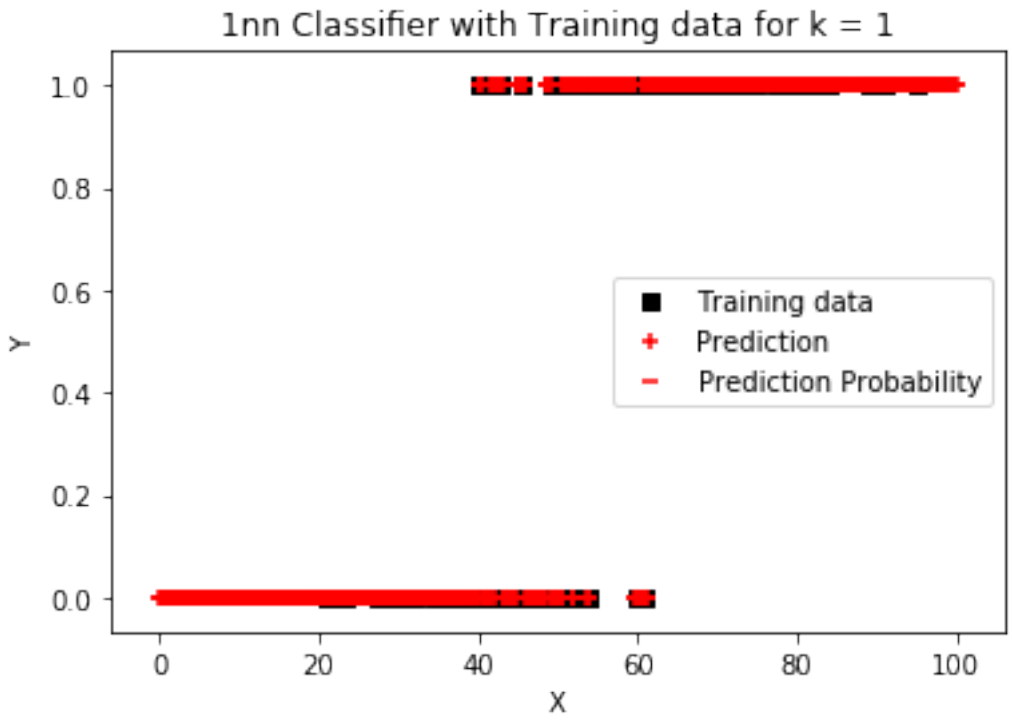
5.3 (C)

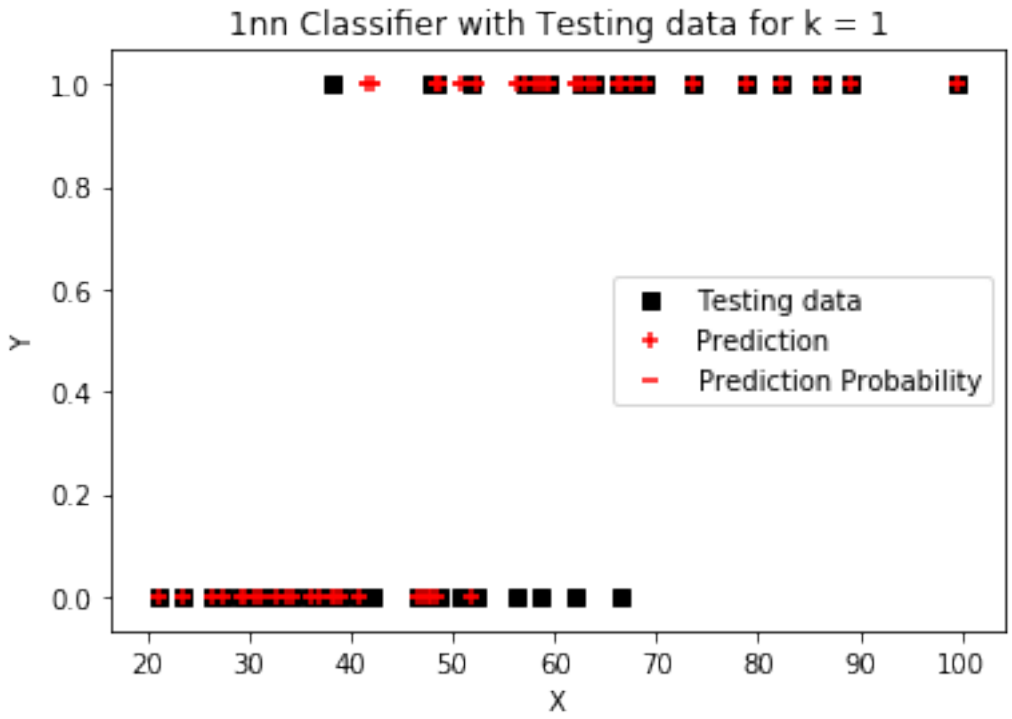
5.3.1 (1)

[24]:
```
for i in [1,3,9]:
    model_kn = kn(n_neighbors=i, weights='uniform', algorithm='auto').
↪fit(x_train, y_train.ravel())



    gen_x = np.linspace(0,100, num = 1000)
    pred_knn_gen = model_kn.predict(gen_x.reshape(1000,-1))
    pred_knn_gen_pr = model_kn.predict_proba(gen_x.reshape(1000,-1))


    ts_x = np.array(df_test['X'])
    ts_y = np.array(df_test['Y'])
    prob_ts_pr = model_kn.predict_proba(ts_x.reshape(ts_size[0],-1))
    prob_ts = model_kn.predict(ts_x.reshape(ts_size[0],-1))

    print("KNN Accuracy for HW2train using score() function- %.2f%%" %(model_kn.
↪score(x_train, y_train)*100))
    print("KNN(%d) Accuracy for HW2test %.2f%%" %(i, model_kn.score(ts_x.
↪reshape(ts_size[0],-1), ts_y)*100))
```

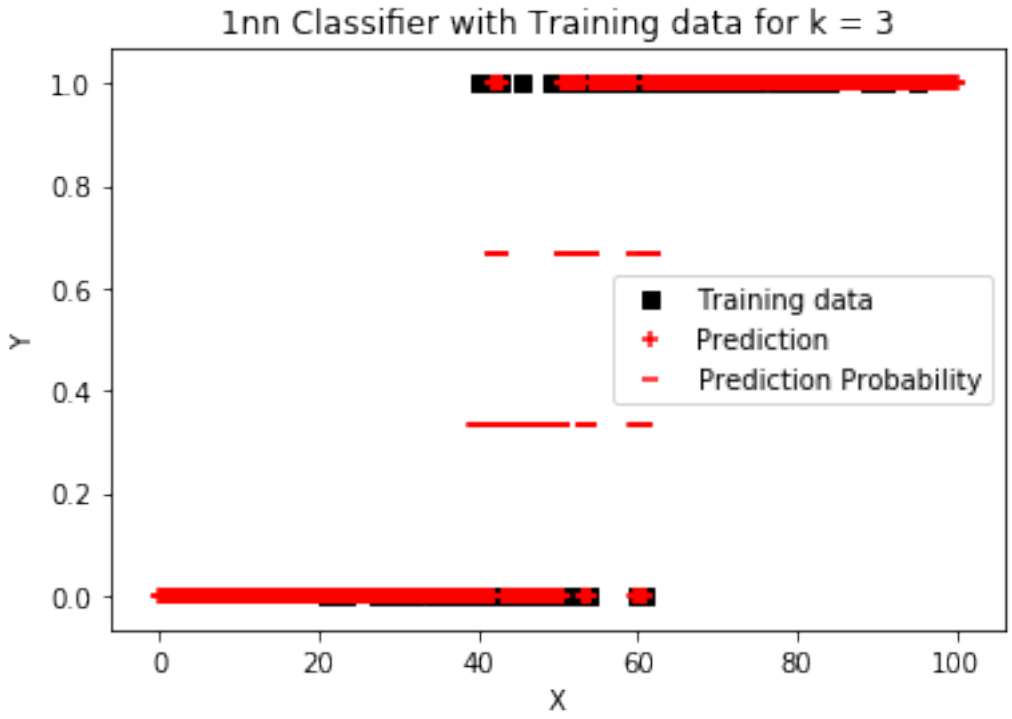
```
   x_tr = x_train.flatten()
   a1 = plt.scatter(df_train['X'], df_train['Y'], color = 'black', label =␣
↪'Training data', marker = 's')
   a2 = plt.scatter(gen_x, pred_knn_gen, color = 'r', marker='+',  label =␣
↪'Prediction class')
   a3 = plt.scatter(gen_x, pred_knn_gen_pr[:,1], color = 'red', marker='_',␣
↪label = 'Prediction Probability')
   plt.xlabel("X")
   plt.ylabel("Y")
   plt.title("1nn Classi er with Training data for k = %i" %i)
   plt.legend([a1, a2, a3], ['Training data', 'Prediction', 'Prediction␣
↪Probability'])
   plt.show()

   a1 = plt.scatter(ts_x, ts_y, color = 'black', label = 'Testing data',␣
↪marker = 's')
   a2 = plt.scatter(ts_x[np.argsort(ts_x)], prob_ts[np.argsort(ts_x)],
                    color = 'r', marker='+', label = 'Prediction class')
   a3 = plt.scatter(ts_x[np.argsort(ts_x)], prob_ts_pr[:,1][np.argsort(ts_x)],
                    color = 'red', marker='_', label = 'Prediction␣
↪Probability')
   plt.xlabel("X")
   plt.ylabel("Y")
   plt.title("1nn Classi er with Testing data for k = %i" %i)
   plt.legend([a1, a2, a3], ['Testing data', 'Prediction', 'Prediction␣
↪Probability'])
   plt.show()
```

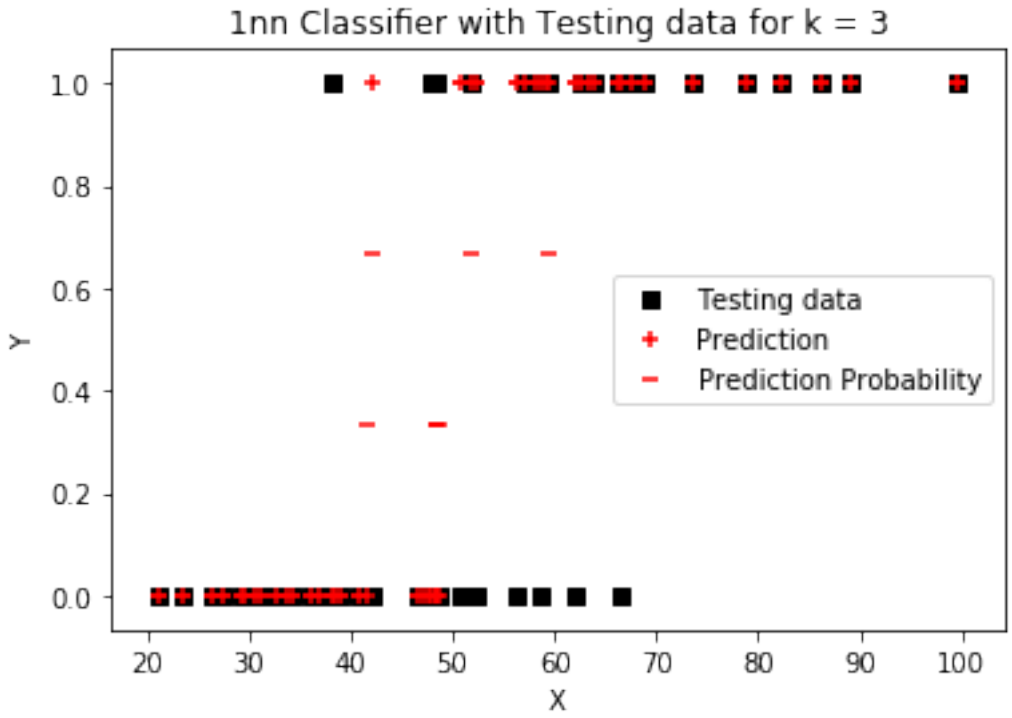
```
KNN Accuracy for HW2train using score() function- 100.00%
KNN(1) Accuracy for HW2test 70.00%
```


1nn Classifier with Training data for k = 1
1.0
0.8
0.6
0.4
0.2
0.0
Y
0
20
40
60
80
100
X
Training data
Prediction
Prediction Probability

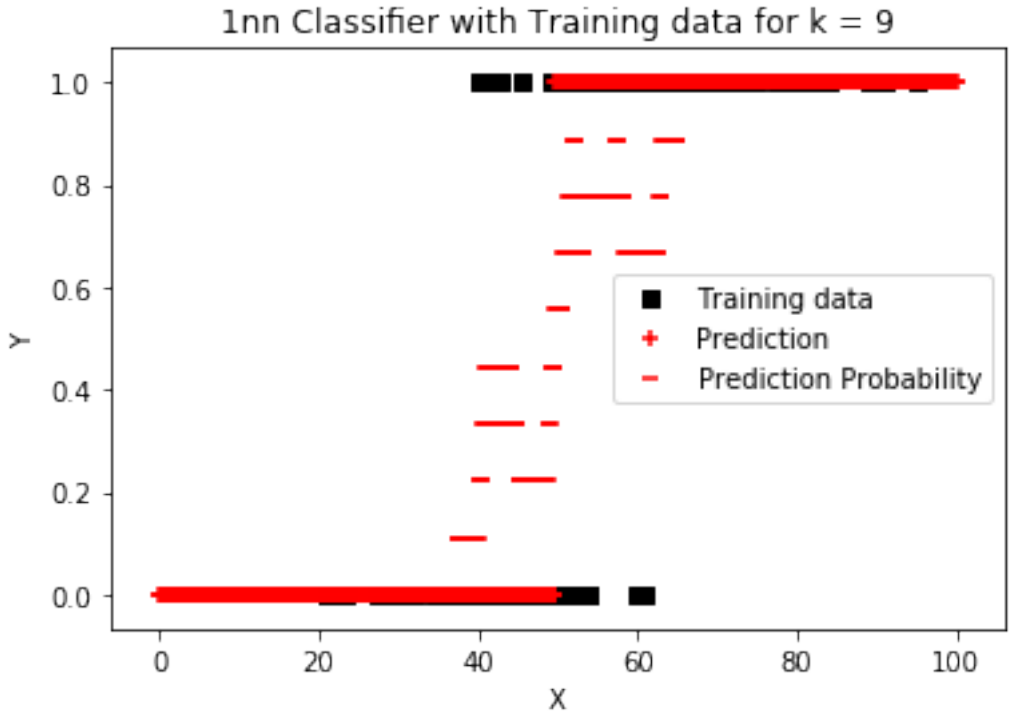

1nn Classifier with Testing data for k = 1
1.0
0.8
0.6
0.4
0.2
0.0
Y
20
30
40
50
60
70
80
90
100
X
Testing data
Prediction
Prediction Probability

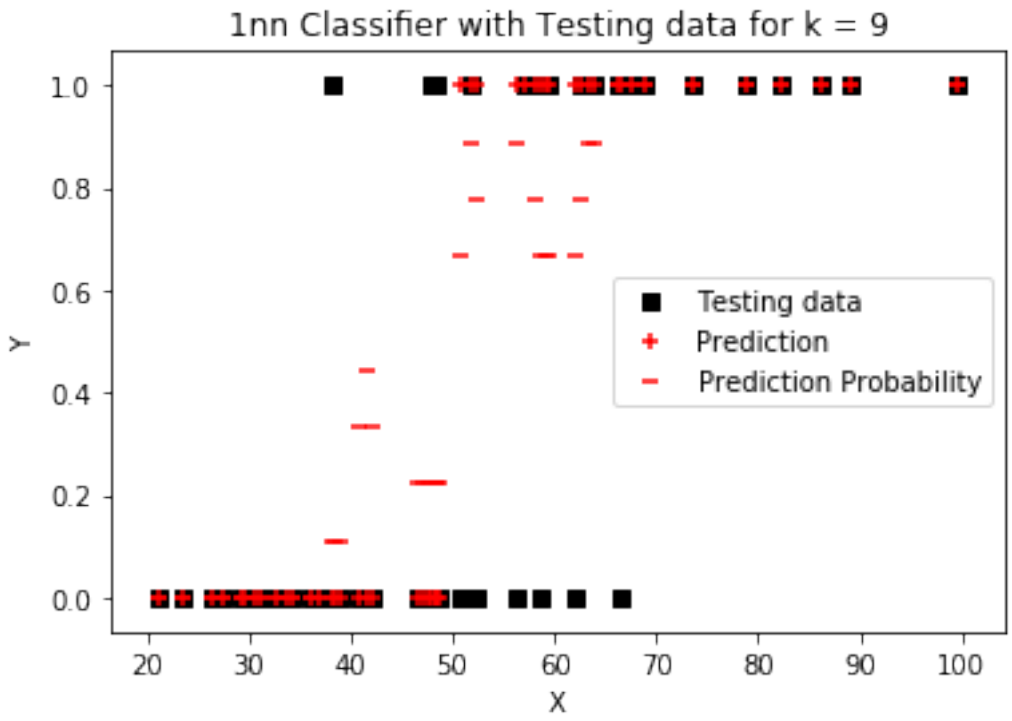
```
KNN Accuracy for HW2train using score() function- 90.00%
KNN(3) Accuracy for HW2test 80.00%
```





```
KNN Accuracy for HW2train using score() function- 86.00%
KNN(9) Accuracy for HW2test 82.00%
```



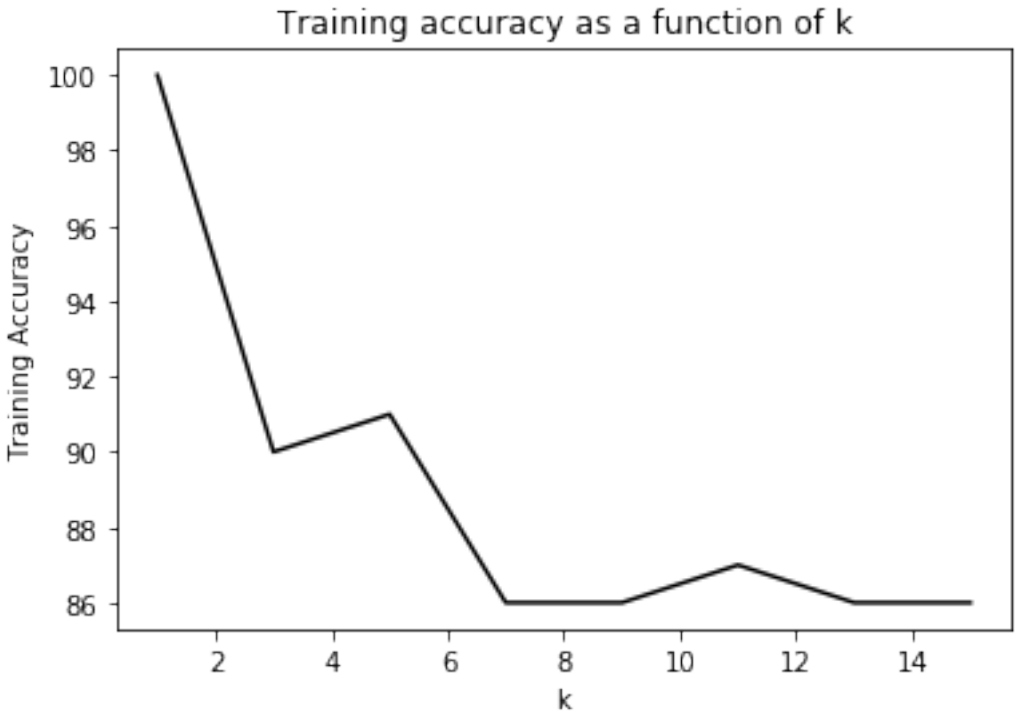


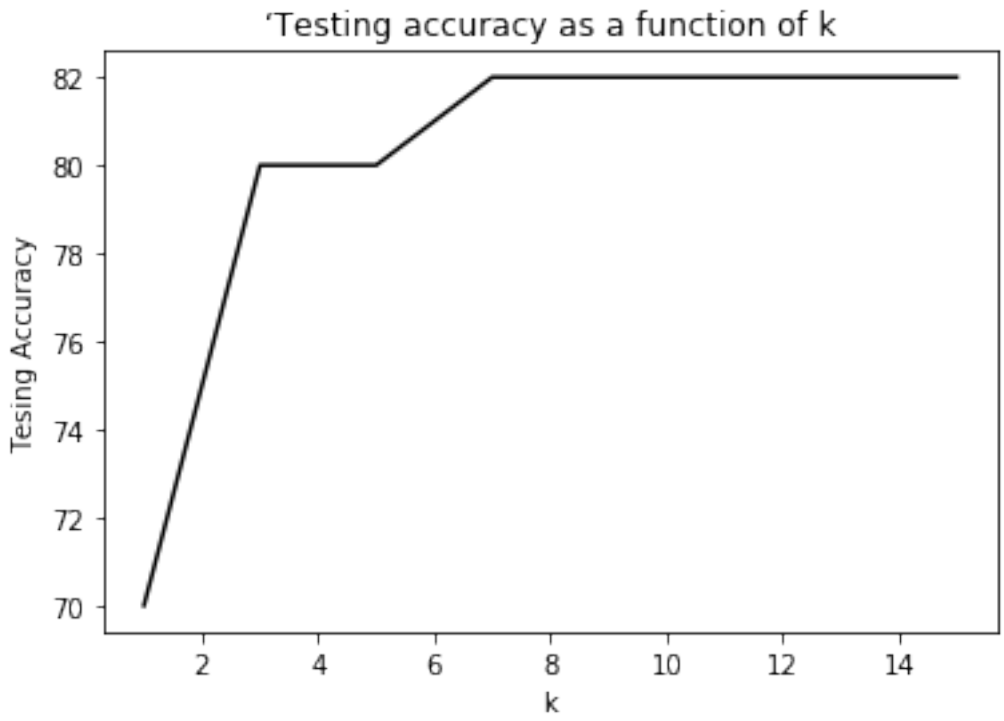
5.4 (C)

5.4.1 (2) - (3)

[26]:
```
tr_acc = []
ts_acc = []
k = []
for i in range(1, 16):
    if i % 2 != 0:
        model_kn = kn(n_neighbors=i, weights='uniform', algorithm='auto').
↪fit(x_train, y_train.ravel())
        k.append(i)
        tr_acc.append(model_kn.score(x_train, y_train)*100)
        ts_acc.append(model_kn.score(ts_x.reshape(ts_size[0],-1), ts_y)*100)


plt.plot(k, tr_acc, color = 'black')
plt.xlabel("k")
plt.ylabel("Training Accuracy")
plt.title("Training accuracy as a function of k")
plt.show()

plt.plot(k, ts_acc, color = 'black')
```

```
plt.xlabel("k")
plt.ylabel("Tesing Accuracy")
plt.title("'Testing accuracy as a function of k")
plt.show()
```





5.5 (D)

Based on test accuracy, nearest neighbor for $k = 9$ is the best (82% accuracy). Definitely, nearest neighbor for $k = 1$ over-fits the data (100% accuracy for the training data and 70% accuracy for the test data). From the accuracy plots for different k values, testing accuracy for $k = 7$ and higher is almost same that indicates nearest neighbor for $k = 7$ is the best model for this data. Both logistic regression and KNN model for $k = 9$ has the same training and testing accuracy. However, in the training data proportion of response for $Y = 1$ is about 60% which indicates that the optimal cutoff probability may not be exactly 0.5 that is what we have used in this problem to classify new data. If we can find the optimal cutoff probability, it may be possible to find the better training and testing accuracy for this data.

$--END--$