In [ ]:
```
import numpy as np
import pandas as pd
import math
import collections

data = pd.read_csv("titanic_data.csv")
data = np.array(data)

X_Bernoulli = data[:,[0, 2]]
X_Multinomial = data[:, [0, 1, 4, 5]]
X_Gaussian = data[:, [0, 3, 6]]
X = data[:,1:]
Y = data[:,0].astype(int)

y_counter = collections.Counter(Y)
```

In [ ]:
```
# Normalize the data set
from sklearn import preprocessing
normalized_data = preprocessing.normalize(X, norm='l2')
normalized_data = np.concatenate(Y, normalized_data)

# Calculate the distance between two points
def distance(x1, x2):
    distance = 0
    for i in range(len(x1)-1):
        distance += (x1[i] - x2[i])**2
    return math.sqrt(distance)

# Select the k nearest neighbors
def neighbors(data, vector, k):
    distances = list()
    for row in data:
        d = distance(vector, row)
        distances.append((row, d))
    distances.sort(key = lambda x: x[1]) # Sort by the distance
    neighbors = list()
    for i in range(k):
        neighbors.append(distances[i][0])
    return neighbors

# Make a classification prediction with neighbors
def predict(data, vector, k):
    n = neighbors(data, vector, k)
    output_values = [row[0] for row in n]
    prediction = max(set(output_values), key=output_values.count)
    return prediction

vector = [3, 1, 30, 0, 0, 100]

for k in range(800):
    predict(normalized_data, vector, k+1)
```

Q1

(b) I used Euclidean distance, but on $normalized$ data. The Euclidean distance between two points is the most straightforward way to see if two points are close to or far from each other. However, in order to avoid the bias from weights placed on different features with vastly discrepant mean and variance, using normalized data is essential.

(c) The result shows that I would have survived the Titanic sinking given the vector $[3, 1, 30, 0, 0, 100]$.

(d) I would choose $k = 8$ because it is not too small to be unstable, and it gives the highest success rate of prediction.

(e) I would use cross validation to assess the confidence level. I can compare different KNNs for various K and find the one with the highest accuracy.

In [ ]:
```
# Calculate the prior
def prior(X, Y):
    m, n = X.shape
    label = np.unique(Y)
    prior_list = np.zeros(2)
    for i in range(m):
        label = Y[i]
        prior_list[int(label)] += 1
    prior_list = prior_list/m
    return prior_list

def Bernoulli_likelihood(vector, y):
    # Calculate the Bernoulli conditional probability
    nom = 0
    for i in range(len(data)):
        x = vector[1]
        if np.all(X_Bernoulli[i] == [y, x]):
            nom += 1

    denom = y_counter[y]
    return nom/denom

def Multinomial_likelihood(vector, y):

    # Select the columns with multinomial distribution
    X = list(i for i in vector if i in [0,3,4])

    # Calculate the Multinomial conditional probability
    nom = [0, 0, 0]
    for idx in range(len(X)):
        x = X[idx]
        for j in range(len(data)):
            if np.all(X_Multinomial[j, [0, idx+1]] == [y, x]):
                nom[idx] += 1

    denom = y_counter[y]
    likelihoods = [nom[idx]/denom for idx in range(len(X))]
    return likelihoods # Return a list of likelihoods

def normal_pdf(x, mean, var):
    denom = (2*math.pi*var)**.5
    num = math.exp(-(float(x)-float(mean))**2/(2*var))
```

```
    return num/denom

def Gaussian_likelihood(vector, y):

    # Select the columns with Gaussian distribution
    X = list(i for i in vector if i in [2, 5])

    # Calculate the Gaussian conditional probability
    likelihoods = []
    for idx in range(len(X)):
        x = X[idx]
        for j in range(len(y_counter)):
            mean = np.mean(X_Gaussian[X_Gaussian[:, 0] == j, :], axis = 0)[idx+
            var = np.var(X_Gaussian[X_Gaussian[:, 0] == j, :], axis = 0)[idx+1]
            likelihoods.append(normal_pdf(x, mean, var))
    return likelihoods # Return a list of likelihoods

def posterior(y):
    post = prior(X,Y)[y]*Bernoulli_likelihood(vector, 0)*np.prod(Multinomial_li
    return post

def predict(vector):
    post = []
    for i in range(len(y.counter)):
        y = y.counter[i]
        post[i] = posterior(y)
    result = np.argmax(post)
    return result
```

In [ ]:

```
vector = [3, 1, 30, 0, 0, 100]
predict(vector)
```

Q2

(b)

For Bernoulli and Multinomial, calculation of likelihoods is relatively simple. I counted the number of instances (e.g. $x_1 = 0, y = 0$) occured, and divide it by the total number of $y = 0$ occured.

For Gaussian, I first obtained the mean and variance of each instance, and then calculate the Gaussian density.

The first two are discrete distributions, and the last is continuous.

(c)

The result shows that I would have survived the Titanic sinking given the vector $[3, 1, 30, 0, 0, 100]$.

(d)

I could use k-fold cross validation to test the accuracy of my Naive Bayes classifier.

Q3

I would prefer Random Forests because it could handle large scale data set, and also categorical data really well, which is the case we have. Also, the features in Titanic data set is highly likely to be correlated, so other methods would suffer much more from the correlation.

Q4

$\hat{P}(y = ham|x) \propto \hat{P}(y = ham) \times \Pi_{j=1}^{D} \hat{P}(x_j|y = ham) = \frac{2}{5} \times \frac{1}{4} \times \frac{1}{2} \times \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4} \times \frac{1}{4}$

Therefore, the email is classified as Spam.

Q5

$\hat{P}(y = female|x) \propto \hat{P}(y = female) \times \Pi_{j=1}^{D} \hat{P}(x_j|y = female) = \frac{1}{3} \frac{1}{\sqrt{2\pi(2)}} e^{-\frac{(42-38)^2}{2(2)}} \frac{1}{\sqrt{2\pi(5}}$

$\approx 1.35 \times 10^{-8} < 6.4745 \times 10^{-4}$

Therefore, the suspect is classified as Male.
