5.1 (a)

```
[ ]  def euclidean_distance(point1, point2):
       return np.linalg.norm(point1[2:] - point2[2:])

     def categorical_dist(point1, point2):
       return (point1[0] == point2[0]) + (point1[1] == point2[1])

     def distance(point1, point2):
       return categorical_dist(point1, point2) + euclidean_distance(point1, point2)

     def k_nearest_neighbors(dist, K, X, y, x):
       distances = []
       for i in range(len(X)):
         distances.append(dist(np.array(X[i,:]) , x))
       distances = np.array(distances)
       kClosest = np.argsort(distances)[:K]
       pred_y = y[kClosest]
       label = mode(pred_y)[0]
       return label[0]

     x = [3,1,26,0,0,7.925]
     K = 12
     print(k_nearest_neighbors(distance, K, X, y, x))
```

As explained in 5.1(b) and 5.1(d), I will be using K = 12 and a custom distance metric in the kNN model

5.1 (b)

I used a combination of two distance measures, one for categorical data and one for non-categorical.

For categorical columns like Pclass and Sex, if the values are different, distance will be counted as 1 and 0 if they are same.

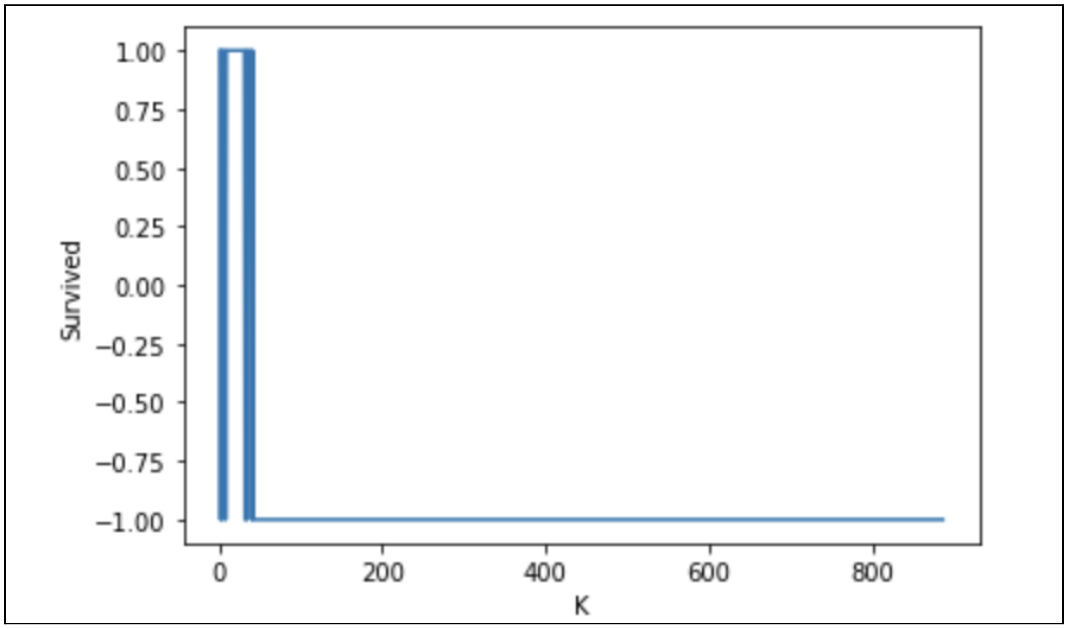
For non-categorical, I am using a regular euclidean distance.

The reason for handling them differently is that it doesn't make sense using any of the standard co-ordinate based distance-metrics such as euclidean or manhattan distance on categorical data.

5.1 ©

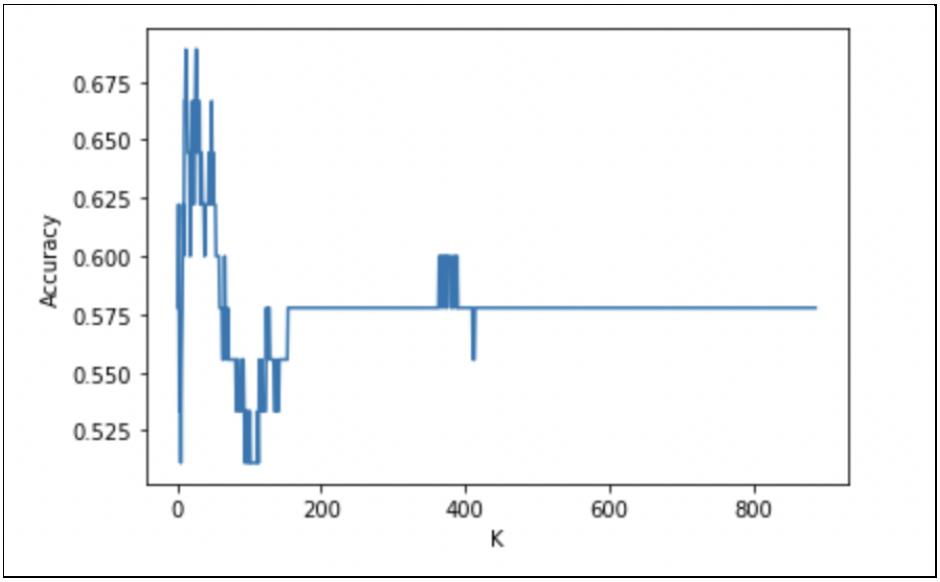
```
x = [1, 0, 25, 3, 2, 20]
```

-1 here refers to **Not survived** and 1 refers to **Survived**




From K = 4 to 40, kNN predicts ‘Survived’ and for the rest of values of K, the model predicts ‘Not survived’.

5.1(d)



I created a test-set and calculated the accuracies of kNN model for different values of K. Highest accuracy of 69% was attained for K = 12. Running cross-validation would have given an even more accurate description of the accuracy but due to the large amount of compute-time that it takes to run the cross-validation set on all values of K = 1 to N, I was unable to do it.

5.1 (e)

Although we have computed the accuracy, it is insufficient to get a good understanding of an ML model. We have to compute various metrics such as F1 score, confusion matrix, precision, recall, etc  to get proper confidence scores of th model. Cross-validation will paint an even better picture of the model since it will generalize well to an independent data set.

5.2 (a)

```
[ ]  def prior():
       n = len(y)
       d = np.bincount(y.astype(int))
       return (d[0]/n, d[1]/n)

     # prior()
```

```
     def multinomial_dist(index, val, label):
       X_dash = X[np.where(y == label)]
       n = len(X_dash[:, index])
       return (np.bincount(X_dash[:,index].astype(int)+1)[int(val)])/(n+len(X))

     # multinomial_dist(0, 1, 0)
     # multinomial_dist(2, 1)
```

```
     0.14678899082568808
```

```
[ ]  def gaussian_dist(index, val, label):
       X_dash = X[np.where(y == label)]
       mean = np.mean(X_dash[:, index])
       variance = np.var(X_dash[:, index])
       denom = (2*math.pi*variance)**.5
       num = math.exp(-(float(val)-float(mean))**2/(2*variance))
       return num/denom

     gaussian_dist(3, 2, 0)
```

```
[ ]  def likelihood(index, val, label):
       if index in [0,1]:
         return multinomial_dist(index, val, label)
       else:
         return gaussian_dist(index, val, label)
```

```
[ ]  def posterior(x, label):
       p = prior()[label]
       for i in range(len(x)):
         p *= likelihood(i, x[i], label)
       return p

     x = [3,0,22,1,0,7.25]
     posterior(x, 0)
     posterior(x, 1)
```

```
     9.803578347977938e-07
```

```
     def naive_bayes(x):
       y_0 = posterior(x, 0)
       y_1 = posterior(x, 1)
       return 1 if y_1 >= y_0 else 0
```

5.2 (b)

Modeled first two columns (pclass and sex) as multinomial and binomial distribution respectively and the rest of the columns as gaussian distribution. This is because first two columns are categorical and the rest are non-categorical. Here we are assuming that non-categorical data follows gaussian distribution. It's possible that they don't, but in the absence of prior knowledge about the data, I will be making this assumption.

5.2 ©

For `x = [1, 0, 45, 2, 2, 20]`, Naive Bayes predicts 'Not survived'

5.2 (d)

Although we have computed the accuracy, it is insufficient to get a good understanding of an ML model. We have to compute various metrics such as F1 score, confusion matrix, precision, recall, etc to get proper confidence scores of th model. Cross-validation will paint an even better picture of the model since it will generalize well to an independent data set.

5.3

I would prefer to use the ensemble of all the models - Random Forests, logistic regression, kNN and Naive Bayes. As explained earlier, while comparing ML models, we have to compare not just the accuracy but other metrics too such as F1 score, confusion matrix, AUC curves, etc. Ideally, we should have computed these for all the models, but since we haven't, it's better to rely on all the model predictions. The final output can be a weighted average of the models with weights being assigned as per their accuracies.

5.4) Not applying any smoothing

$$\hat{P}(y=ham|x) \propto \hat{P}(y=ham) \cdot \prod_{j=1}^{p} \hat{P}(x_j|y=ham)$$

$$= \frac{2}{5} * \frac{1}{4} * \frac{1}{2} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{4} * \frac{1}{2} * \frac{1}{2}$$

$$= 0.48 \times 10^{-4}$$

$P(y=spam|x) > P(y=ham|x)$

Predict x as spam

5.

5.5)

$x = [42 \quad 180 \quad 5.5]^T$

$$\hat{p}(x_1 | y = \text{female}) = \frac{1}{\sqrt{2\pi(2)}} e^{\frac{-(42-38)^2}{2(2)}}$$

$$= 0.0051$$

$$\hat{p}(x_2 | y = \text{female}) = \frac{1}{\sqrt{2\pi(50)}} e^{\frac{-(180-165)^2}{2(50)}}$$

$$= 0.0059$$

$$\hat{p}(x_3 | y = \text{female}) = \frac{1}{\sqrt{2\pi(0.125)}} e^{\frac{-(5.5-6.75)^2}{2(0.125)}}$$

$$= 0.0021$$

$$\hat{p}(y = \text{female} | x) = p(y = \text{female}) \prod_{j=1}^{D} \hat{p}(x_i | y = \text{female})$$

$$= \frac{1}{3} * 0.0051 * 0.0059 * 0.0021$$

$$= 2.1 * 10^{-8}$$

$$p(y = \text{male} | x) > p(y = \text{female} | x)$$

Predict x as male