Machine Learning(SWE4012)

T.Akash Reddy

21MIS1128

KNN Algorithm

Dataset used:

- I took a dataset related to breast cancer which is having 699 rows and 11 columns with different class variables.
- Here the dataset suits best for knn because we are going to say whether it is bengin or malignant cancer which is something like Yes or No.

Jupyter notebook screenshots with explanation:

- Imported all the necessary libraries.

```
In [105]: import numpy as np # linear algebra
          import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
          import matplotlib.pyplot as plt # for data visualization purposes
          import seaborn as sns # for data visualization
          %matplotlib inline
```

- Imported the dataset.

```
In [107]: #importing dataset
          filepath = r'D:\Akash Personal Files\Elango\breastcancer\breast-cancer-wisconsin.data.txt'
          df = pd.read_csv(filepath, header=None)
```

```
In [108]: # view dimensions of dataset

          df.shape

Out[108]: (699, 11)
```

```
In [109]: # preview the dataset

          df.head()

Out[109]:
```

	0	1	2	3	4	5	6	7	8	9	10
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

- For better understanding renamed all the columns with meaningful names.

```
In [110]: #Renaming the column names for better visualization and columns can have meaningfull names
col_names = ['Id', 'Clump_thickness', 'Uniformity_Cell_Size', 'Uniformity_Cell_Shape', 'Marginal_Adhesion',
             'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin', 'Normal_Nucleoli', 'Mitoses', 'Class']

df.columns = col_names

df.columns
```

```
Out[110]: Index(['Id', 'Clump_thickness', 'Uniformity_Cell_Size',
       'Uniformity_Cell_Shape', 'Marginal_Adhesion',
       'Single_Epithelial_Cell_Size', 'Bare_Nuclei', 'Bland_Chromatin',
       'Normal_Nucleoli', 'Mitoses', 'Class'],
      dtype='object')
```

```
In [111]: #preview the data
df.head()
```

Out[111]:

	Id	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Norma
0	1000025	5	1	1	1	2	1	3	
1	1002945	5	4	4	5	7	10	3	
2	1015425	3	1	1	1	2	2	3	
3	1016277	6	8	8	1	3	4	3	
4	1017023	4	1	1	3	2	1	3	

- Pre-Processing the data.(80% for training and 20% for testing)

```
In [58]: #preprocessing the data
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
In [59]: X_train.shape, X_test.shape

Out[59]: ((559, 9), (140, 9))
```

- Checking the data types in train dataset.

```
In [60]: # check data types in X_train

X_train.dtypes
```

```
Out[60]: Clump_thickness                int64
Uniformity_Cell_Size           int64
Uniformity_Cell_Shape          int64
Marginal_Adhesion              int64
Single_Epithelial_Cell_Size    int64
Bare_Nuclei                  float64
Bland_Chromatin                int64
Normal_Nucleoli                int64
Mitoses                        int64
dtype: object
```

- Frequency distribution of values in class

```
In [51]: # view percentage of frequency distribution of values in `Class` variable

df['Class'].value_counts()/np.float(len(df))
```

```
Out[51]: 2    0.655222
4    0.344778
Name: Class, dtype: float64
```

We can see that the Class variable contains 2 class labels 2 and 4. 2 stands for benign and 4 stands for malignant cancer.

- Train and Test dataset:

```
In [67]: X_train.head()
```

Out[67]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucle
293	10	4	4	6	2	10.0	2	
62	9	10	10	1	10	8.0	3	
485	1	1	1	3	1	3.0	1	
422	4	3	3	1	2	1.0	3	
332	5	2	2	2	2	1.0	2	

```
In [68]: X_test.head()
```

Out[68]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucle
476	4	1	2	1	2	1.0	1	
531	4	2	2	1	2	1.0	2	
40	6	6	6	9	6	1.0	7	
432	5	1	1	1	2	1.0	2	
14	8	7	5	10	7	9.0	5	

We now have training and testing set ready for model building. Before that, we should map all the feature variables onto the same scale. It is called feature scaling.

- Feature Scaling:

```
In [69]: #Feature Scaling
         cols = X_train.columns

In [70]: from sklearn.preprocessing import StandardScaler

         scaler = StandardScaler()

         X_train = scaler.fit_transform(X_train)

         X_test = scaler.transform(X_test)

In [72]: X_train = pd.DataFrame(X_train, columns=[cols])
         X_test = pd.DataFrame(X_test, columns=[cols])
         X_train.head() #train dataset
```

Out[72]:

	Clump_thickness	Uniformity_Cell_Size	Uniformity_Cell_Shape	Marginal_Adhesion	Single_Epithelial_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli
0	2.028383	0.299506	0.289573	1.119077	-0.546543	1.858357	-0.577774	0.041241
1	1.669451	2.257680	2.304569	-0.622471	3.106879	1.297589	-0.159953	0.041241
2	-1.202005	-0.679581	-0.717925	0.074148	-1.003220	-0.104329	-0.995595	-0.608165
3	-0.125209	-0.026856	-0.046260	-0.622471	-0.546543	-0.665096	-0.159953	0.041241
4	0.233723	-0.353219	-0.382092	-0.274161	-0.546543	-0.665096	-0.577774	-0.283462

We now have X_train dataset ready to be fed into the Logistic Regression classifier.

- Fit K Neighbours Classifier to the training set:
  First we will take k as 3 and later will check with different k-values.

```
In [73]: #K-neighbours fit into knn
         # import KNeighbors ClaSSifier from sklearn
         from sklearn.neighbors import KNeighborsClassifier


         # instantiate the model
         knn = KNeighborsClassifier(n_neighbors=3)


         # fit the model to the training set
         knn.fit(X_train, y_train)

Out[73]: KNeighborsClassifier(n_neighbors=3)
```

- Prediction:

```
In [74]: #Predicting the type of cancer
         y_pred = knn.predict(X_test)

         y_pred

Out[74]: array([2, 2, 4, 2, 4, 2, 4, 2, 4, 2, 2, 2, 4, 4, 4, 2, 2, 4, 4, 2, 4, 4,
                2, 2, 2, 4, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2,
                4, 4, 2, 4, 2, 4, 4, 2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 4, 4, 4,
                4, 2, 2, 4, 2, 2, 4, 4, 2, 2, 2, 2, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2,
                4, 4, 2, 2, 2, 4, 2, 2, 2, 4, 2, 4, 4, 2, 2, 2, 4, 2, 2, 2, 2, 2,
                4, 4, 4, 2, 2, 2, 2, 2, 4, 4, 4, 4, 2, 4, 2, 2, 4, 4, 4, 4, 4, 2,
                2, 4, 4, 2, 2, 4, 2, 2], dtype=int64)
```

```
In [75]: # probability of getting output as 2 - benign cancer

         knn.predict_proba(X_test)[:,0]

Out[75]: array([1.        , 1.        , 0.33333333, 1.        , 0.        ,
                1.        , 0.        , 1.        , 0.        , 0.66666667,
                1.        , 1.        , 0.        , 0.33333333, 0.        ,
                1.        , 1.        , 0.        , 0.        , 1.        ,
                0.        , 0.        , 1.        , 1.        , 1.        ,
                0.        , 1.        , 1.        , 0.        , 0.        ,
                1.        , 1.        , 1.        , 1.        , 1.        ,
                0.66666667, 1.        , 0.        , 1.        , 1.        ,
                1.        , 1.        , 1.        , 1.        , 0.        ,
                0.        , 1.        , 0.        , 1.        , 0.        ,
                0.        , 1.        , 1.        , 0.        , 1.        ,
                1.        , 1.        , 1.        , 0.66666667, 1.        ,
                0.        , 1.        , 1.        , 0.        , 0.        ,
                0.33333333, 0.        , 1.        , 1.        , 0.        ,
                1.        , 1.        , 0.        , 0.        , 1.        ,
                1.        , 1.        , 1.        , 0.        , 1.        ,
                1.        , 1.        , 0.        , 1.        , 1.        ,
                1.        , 0.        , 1.        , 0.        , 0.        ,
                1.        , 1.        , 0.66666667, 0.        , 1.        ,
                1.        , 1.        , 0.        , 1.        , 0.        ,
                0.        , 1.        , 1.        , 1.        , 0.        ,
                1.        , 1.        , 1.        , 1.        , 1.        ,
                0.        , 0.33333333, 0.        , 1.        , 1.        ,
                1.        , 1.        , 1.        , 0.        , 0.        ,
                0.        , 0.33333333, 1.        , 0.        , 1.        ,
                1.        , 0.33333333, 0.33333333, 0.        , 0.        ,
                0.        , 1.        , 1.        , 0.33333333, 0.        ,
                1.        , 1.        , 0.        , 1.        , 1.        ])
```

```
In [76]: # probability of getting output as 4 - malignant cancer

         knn.predict_proba(X_test)[:,1]

Out[76]: array([0.        , 0.        , 0.66666667, 0.        , 1.        ,
                0.        , 1.        , 0.        , 1.        , 0.33333333,
                0.        , 0.        , 1.        , 0.66666667, 1.        ,
                0.        , 0.        , 1.        , 1.        , 0.        ,
                1.        , 1.        , 0.        , 0.        , 0.        ,
                1.        , 0.        , 0.        , 1.        , 1.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                0.33333333, 0.        , 1.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 0.        , 1.        ,
                1.        , 0.        , 1.        , 0.        , 1.        ,
                1.        , 0.        , 0.        , 1.        , 0.        ,
                0.        , 0.        , 0.        , 0.33333333, 0.        ,
                1.        , 0.        , 0.        , 1.        , 1.        ,
                0.66666667, 1.        , 0.        , 0.        , 1.        ,
                0.        , 0.        , 1.        , 1.        , 0.        ,
                0.        , 0.        , 0.        , 1.        , 0.        ,
                0.        , 0.        , 1.        , 0.        , 0.        ,
                0.        , 1.        , 0.        , 1.        , 1.        ,
                0.        , 0.        , 0.33333333, 1.        , 0.        ,
                0.        , 0.        , 1.        , 0.        , 1.        ,
                1.        , 0.        , 0.        , 0.        , 1.        ,
                0.        , 0.        , 0.        , 0.        , 0.        ,
                1.        , 0.66666667, 1.        , 0.        , 0.        ,
                0.        , 0.        , 0.        , 1.        , 1.        ,
                1.        , 0.66666667, 0.        , 1.        , 0.        ,
                0.        , 0.66666667, 0.66666667, 1.        , 1.        ,
```

- Model Accuracy:

```
In [77]: #Model Accuracy
         from sklearn.metrics import accuracy_score

         print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))

         Model accuracy score: 0.9714
```

- Comparing the test and train dataset accuracy:

```
In [78]: #comparing the train dataset and test dataset accuracy
         y_pred_train = knn.predict(X_train)
         print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))

         Training-set accuracy score: 0.9821
```

```
In [79]: # checking for overfitting

         print('Training set score: {:.4f}'.format(knn.score(X_train, y_train)))

         print('Test set score: {:.4f}'.format(knn.score(X_test, y_test)))

         Training set score: 0.9821
         Test set score: 0.9714
```

The training dataset accuracy score is 0.9821 while the test dataset accuracy to be 0.9714. These two values are quite comparable. So, there is no question of overfitting.

- Checking our model with different k-values.

In [80]:
```python
#Checking our model with different k-values
# instantiate the model with k=5
knn_5 = KNeighborsClassifier(n_neighbors=5)


# fit the model to the training set
knn_5.fit(X_train, y_train)


# predict on the test-set
y_pred_5 = knn_5.predict(X_test)


print('Model accuracy score with k=5 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_5)))
```

```
Model accuracy score with k=5 : 0.9714
```

In [81]:
```python
# instantiate the model with k=6
knn_6 = KNeighborsClassifier(n_neighbors=6)


# fit the model to the training set
knn_6.fit(X_train, y_train)


# predict on the test-set
y_pred_6 = knn_6.predict(X_test)


print('Model accuracy score with k=6 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_6)))
```

```
Model accuracy score with k=6 : 0.9786
```

In [82]:
```python
# instantiate the model with k=7
knn_7 = KNeighborsClassifier(n_neighbors=7)


# fit the model to the training set
knn_7.fit(X_train, y_train)


# predict on the test-set
y_pred_7 = knn_7.predict(X_test)


print('Model accuracy score with k=7 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_7)))
```

```
Model accuracy score with k=7 : 0.9786
```

In [83]:
```python
# instantiate the model with k=8
knn_8 = KNeighborsClassifier(n_neighbors=8)


# fit the model to the training set
knn_8.fit(X_train, y_train)


# predict on the test-set
y_pred_8 = knn_8.predict(X_test)


print('Model accuracy score with k=8 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_8)))
```

```
Model accuracy score with k=8 : 0.9786
```

```
In [84]: # instantiate the model with k=9
knn_9 = KNeighborsClassifier(n_neighbors=9)


# fit the model to the training set
knn_9.fit(X_train, y_train)


# predict on the test-set
y_pred_9 = knn_9.predict(X_test)


print('Model accuracy score with k=9 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_9)))

Model accuracy score with k=9 : 0.9714
```

- Interpretation:
    1. For k=3 we got 0.9714 accuracy and got same with k=5, when we increase the k value with 6,7,8 we got 0.9786 as accuracy but with k=9 again we got 0.9714. Based on this our model is working very good in terms of predicting the class labels.
    2. But our model does not tell about the errors that classifier is doing, for that we are going to use confusion matrix, it summarizes the performance of our model and types of errors.

- Confusion Matrix:

```
In [85]: # Print the Confusion Matrix with k =3 and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])

Confusion matrix

 [[83  2]
 [ 2 53]]

True Positives(TP) =  83

True Negatives(TN) =  53

False Positives(FP) =  2

False Negatives(FN) =  2
```

We have 83+53=136 correct predictions and 2+2=4 incorrect predictions.

In [86]:
```
# Print the Confusion Matrix with k =7 and slice it into four pieces

cm_7 = confusion_matrix(y_test, y_pred_7)

print('Confusion matrix\n\n', cm_7)

print('\nTrue Positives(TP) = ', cm_7[0,0])

print('\nTrue Negatives(TN) = ', cm_7[1,1])

print('\nFalse Positives(FP) = ', cm_7[0,1])

print('\nFalse Negatives(FN) = ', cm_7[1,0])
```

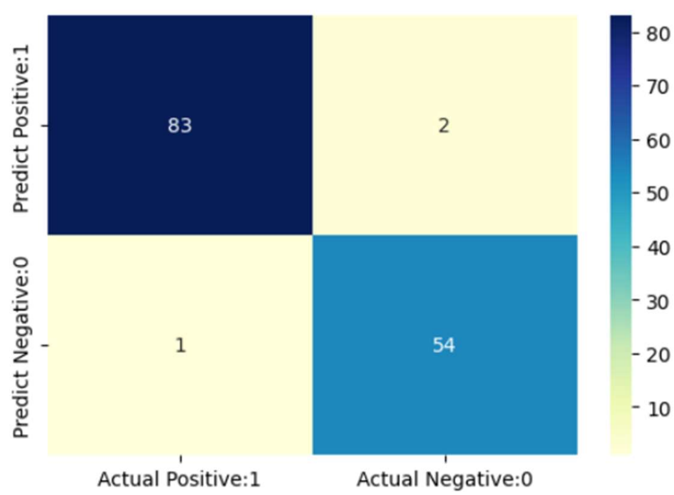
```
Confusion matrix

 [[83  2]
 [ 1 54]]

True Positives(TP) =  83

True Negatives(TN) =  54

False Positives(FP) =  2

False Negatives(FN) =  1
```

For k=7,we only got 3 incorrect predictions so the performance of the model is increased with k=7

- Visualization of confusion matrix:

In [87]:
```
#we got performance improvement with k=7,because it make only 3 incorrect predictions
# visualize confusion matrix with seaborn heatmap

plt.figure(figsize=(6,4))

cm_matrix = pd.DataFrame(data=cm_7, columns=['Actual Positive:1', 'Actual Negative:0'],
                                    index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[87]: <AxesSubplot:>



- Classification Report:

```
In [88]: #classification Report
         from sklearn.metrics import classification_report

         print(classification_report(y_test, y_pred_7))

                       precision    recall  f1-score   support

                    2       0.99      0.98      0.98        85
                    4       0.96      0.98      0.97        55

             accuracy                           0.98       140
            macro avg       0.98      0.98      0.98       140
         weighted avg       0.98      0.98      0.98       140
```

```
In [89]: #Classification Accuracy
         TP = cm_7[0,0]
         TN = cm_7[1,1]
         FP = cm_7[0,1]
         FN = cm_7[1,0]
```

```
In [90]: # print classification accuracy

         classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

         print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))

         Classification accuracy : 0.9786
```

```
In [91]: # print classification error

         classification_error = (FP + FN) / float(TP + TN + FP + FN)

         print('Classification error : {0:0.4f}'.format(classification_error))

         Classification error : 0.0214
```

```
In [92]: # print precision score

         precision = TP / float(TP + FP)


         print('Precision : {0:0.4f}'.format(precision))

         Precision : 0.9765
```

```
In [93]: #Print Recall score
         recall = TP / float(TP + FN)

         print('Recall or Sensitivity : {0:0.4f}'.format(recall))

         Recall or Sensitivity : 0.9881
```

```
In [94]: #print true positive rate
         true_positive_rate = TP / float(TP + FN)


         print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))

         True Positive Rate : 0.9881
```

```
In [95]: #print false positive rate
         false_positive_rate = FP / float(FP + TN)


         print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))

         False Positive Rate : 0.0357
```

```
In [96]: #print specifity
         specificity = TN / (TN + FP)

         print('Specificity : {0:0.4f}'.format(specificity))

         Specificity : 0.9643
```

- Printing the first 10 predicted probabilities of class 2 and 4

```
In [98]: # print the first 10 predicted probabilities of two classes- 2 and 4

y_pred_prob = knn.predict_proba(X_test)[0:10]

y_pred_prob
```

```
Out[98]: array([[1.        , 0.        ],
       [1.        , 0.        ],
       [0.33333333, 0.66666667],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.66666667, 0.33333333]])
```

```
In [99]: # store the probabilities of cancer in dataframe

y_pred_prob_df = pd.DataFrame(data=y_pred_prob, columns=['Prob of - benign cancer (2)', 'Prob of - malignant cancer (4)'])

y_pred_prob_df
```

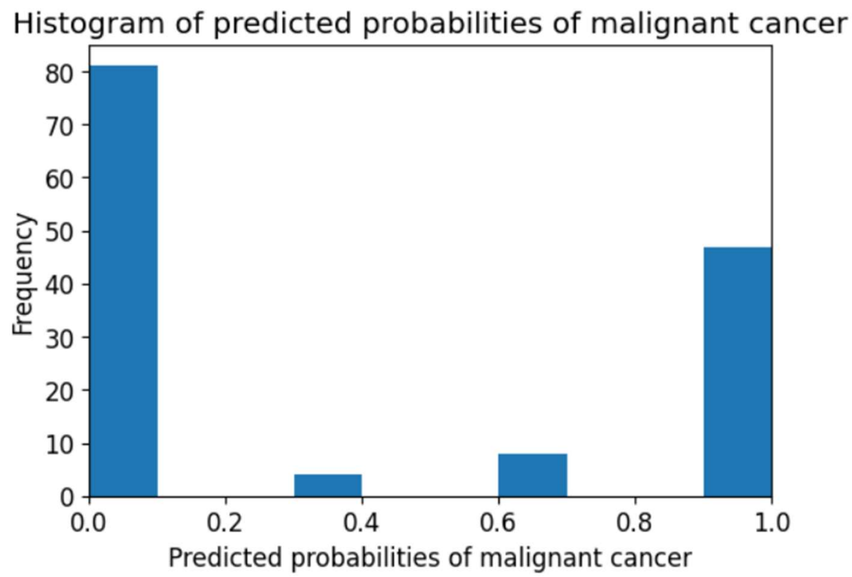
Out[99]:

	Prob of - benign cancer (2)	Prob of - malignant cancer (4)
0	1.000000	0.000000
1	1.000000	0.000000
2	0.333333	0.666667
3	1.000000	0.000000
4	0.000000	1.000000
5	1.000000	0.000000
6	0.000000	1.000000
7	1.000000	0.000000
8	0.000000	1.000000
9	0.666667	0.333333

- Histogram for predicted probabilities

```
In [118]: # plot histogram of predicted probabilities


# adjust figure size
plt.figure(figsize=(6,4))


# adjust the font size
plt.rcParams['font.size'] = 12


# plot histogram with 10 bins
plt.hist(y_pred_1, bins = 10)


# set the title of predicted probabilities
plt.title('Histogram of predicted probabilities of malignant cancer')


# set the x-axis limit
plt.xlim(0,1)


# set the title
plt.xlabel('Predicted probabilities of malignant cancer')
plt.ylabel('Frequency')
```

```
Out[118]: Text(0, 0.5, 'Frequency')
```

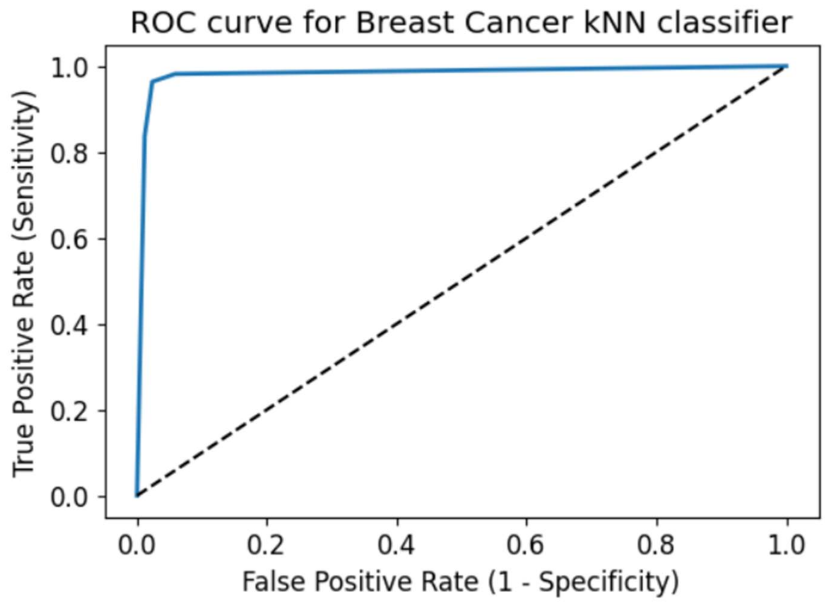


The histogram is positively skewed.
There are observations with 0 probability of malignant cancer.
Some are having the probability of >0.5 so that means these observations predict that there will be malignant cancer.

- ROC-AUC curve:

ROC AUC stands for Receiver Operating Characteristic - Area Under Curve. It is a technique to compare classifier performance. In this technique, we measure the area under the curve (AUC).

```
In [114]: # plot ROC Curve

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred_1, pos_label=4)

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Breast Cancer kNN classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```



- Computing area under curve

```
In [115]: # compute ROC AUC

          from sklearn.metrics import roc_auc_score

          ROC_AUC = roc_auc_score(y_test, y_pred_1)

          print('ROC AUC : {:.4f}'.format(ROC_AUC))

          ROC AUC : 0.9825
```

ROC AUC of our model is 0.9825 which approaches towards 1. So, we can conclude that our classifier does a good job in predicting whether it is benign or malignant cancer.

- Final Interpretation of the model:
  1. *The model yields very good performance as indicated by the model accuracy which was found to be 0.9786 with k=7.*
  2. *With k=3, the training-set accuracy score is 0.9821 while the test-set accuracy to be 0.9714. These two values are quite comparable. So, there is no question of overfitting.*
  3. *Our original model accuracy score with k=3 is 0.9714. Now, we can see that we get same accuracy score of 0.9714 with k=5. But, if we increase the value of k further, this would result in enhanced accuracy. With k=6,7,8 we get accuracy score of 0.9786. So, it results in performance improvement. If we increase k to 9, then accuracy decreases again to 0.9714. So, we can conclude that our optimal value of k is 7.*