

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology



Department of Computer Science and Engineering

Course no: CSE 4204

Course Title: Sessional Based on CSE 4203

Experiment no: 1

Name of the experiment:

Implementation of Nearest Neighbor classification algorithms with
and without distorted pattern

Submitted by

Partho Kumar Rajvor

Roll: 1803119

Section: B

Department of Computer Science and Engineering

Rajshahi University of Engineering and Technology

Submitted to

Rizoan Toufiq

Assistant Professor

Department of Computer Science and Engineering

Rajshahi University of Engineering and Technology

Table of Contents

1	Implementation of Nearest Neighbor classification algorithm	3
1.1	Introduction	3
1.2	Preparing the dataset	4
1.2.1	Foreword	4
1.2.2	Selecting dataset	4
1.2.3	Loading the dataset	4
1.2.4	Characteristics of the dataset	5
1.2.5	Preprocessing the dataset	6
1.2.6	Selecting the features and the output	7
1.2.7	Splitting the dataset into training and test sets	7
1.3	Implementing the k-NN algorithm	8
1.3.1	Selecting the value of k	8
1.3.2	Calculating the distance between two samples	8
1.3.3	Training the model	8
1.3.4	Predicting the output	9
1.4	Calculating the accuracy of the model	9
1.4.1	Confusion matrix	9
1.4.2	Accuracy score and f1 score	10
1.4.3	Relation between the value of k and the accuracy of the model	10
1.5	Drawbacks of the k-NN algorithm	11
1.6	Conclusion	12

Chapter 1

Implementation of Nearest Neighbor classification algorithm

1.1 Introduction

In statistics, the k-nearest neighbors algorithm (k-NN) is a non-parametric classification method first developed by Evelyn Fix and Joseph Hodges in 1951, and later expanded by Thomas Cover. It is used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space.

- In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.
- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

1.2 Preparing the dataset

1.2.1 Foreword

We will be using the following libraries in this lab:

- *pandas* for loading and preprocessing the dataset.
- *scikit-learn* for splitting the dataset into training and test sets and for implementing the k-NN algorithm as well as for calculating the accuracy of the model.
- *math* for calculating the square root of a number.

1.2.2 Selecting dataset

In this lab, we will use the *Breast Cancer*¹ dataset. This dataset contains 569 samples of malignant and benign tumor cells. The first two columns in the dataset store the unique ID numbers of the samples and the corresponding diagnosis (M=malignant, B=benign), respectively. Columns 3-32 contain 30 real-valued features that have been computed from digitized images of the cell nuclei, which can be used to build a model to predict whether a tumor is benign or malignant.

From now on, we will use *dataset* to refer to this dataset.

1.2.3 Loading the dataset

We will be using the *pandas* library to load the dataset. The following code snippet shows how we can load the dataset using *pandas*:

```
import pandas as pd
df = pd.read_csv(
    'breast-cancer-wisconsin-data_data.csv',
    usecols=lambda c: not c.startswith('Unnamed'))
```

¹<https://data.world/health/breast-cancer-wisconsin>

```
[8]: df = pd.read_csv('breast-cancer-wisconsin-data_data.csv', usecols=lambda c: not c.startswith('Unnamed'))
print(df.head(5))
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	\
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

	smoothness_mean	compactness_mean	concavity_mean	concave	points_mean	\
0	0.11840	0.27760	0.3001		0.14710	
1	0.08474	0.07864	0.0869		0.07017	
2	0.10960	0.15990	0.1974		0.12790	
3	0.14250	0.28390	0.2414		0.10520	
4	0.10030	0.13280	0.1980		0.10430	

	...	radius_worst	texture_worst	perimeter_worst	area_worst	\
0	...	25.38	17.33	184.60	2019.0	
1	...	24.99	23.41	158.80	1956.0	
2	...	23.57	25.53	152.50	1709.0	
3	...	14.91	26.50	98.87	567.7	
4	...	22.54	16.67	152.20	1575.0	

	smoothness_worst	compactness_worst	concavity_worst	concave	points_worst	\
0	0.1622	0.6656	0.7119		0.2654	
1	0.1238	0.1866	0.2416		0.1860	
2	0.1444	0.4245	0.4504		0.2430	
3	0.2098	0.8663	0.6869		0.2575	
4	0.1374	0.2050	0.4000		0.1625	

	symmetry_worst	fractal_dimension_worst
0	0.4601	0.11890
1	0.2750	0.08902
2	0.3613	0.08758
3	0.6638	0.17300
4	0.2364	0.07678

[5 rows x 32 columns]

Figure 1.1: A sneak peek into the dataset

1.2.4 Characteristics of the dataset

There are some key characteristics of the dataset that we need to know before we start working with it. To mention a few of them:

- The dataset contains 569 samples.
- Each sample has 30 features.
- The dataset contains 212 malignant and 357 benign samples.
- The dataset contains no missing values.

This narrows down our task to the following, viz., our dataset has the following characteristics:

- Size: 569 (i.e., 569 samples)
- Dimensionality: 30 (i.e., 30 features)

- Classes: 2 (i.e., 2 classes, malignant and benign)
- Ten real valued features are computed for each cell nucleus:
 - Radius (mean of distances from center to points on the perimeter)
 - Texture (standard deviation of gray-scale values)
 - Perimeter
 - Area
 - Smoothness (local variation in radius lengths)
 - Compactness ($\text{perimeter}^2 / \text{area} - 1.0$)
 - Concavity (severity of concave portions of the contour)
 - Concave points (number of concave portions of the contour)
 - Symmetry
 - Fractal dimension ("coastline approximation" - 1)
- Missing values: None
- Class distribution: 357 benign, 212 malignant

1.2.5 Preprocessing the dataset

To ease the process of working with the dataset, we will specifically preprocess the *disagnosis* column of the dataset. We will replace the values *M* and *B* with 1 and 0 respectively. This will help us to work with the dataset more easily.

```
df[ 'diagnosis ' ] = df[ 'diagnosis ' ]
                      .replace( 'M', 1)
df[ 'diagnosis ' ] = df[ 'diagnosis ' ]
                      .replace( 'B', 0)
```

1.2.6 Selecting the features and the output

We will be using the first 30 columns of the dataset as the features and the last column as the output. We will use the following code snippet to select the features and the output:

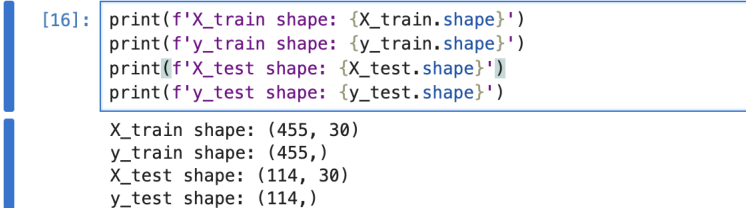
```
x = df.iloc[:, 2:32]
y = df.iloc[:, 1]
```

1.2.7 Splitting the dataset into training and test sets

We will be using the *scikit-learn* library to split the dataset into training and test sets. We will use the following code snippet to split the dataset into training and test sets:

```
X_train ,
X_test ,
y_train ,
y_test = train_test_split(x, y,
                           test_size=0.2)
```

Here we have split the dataset into 80% training set and 20% test set.



```
[16]: print(f'X_train shape: {X_train.shape}')
      print(f'y_train shape: {y_train.shape}')
      print(f'X_test shape: {X_test.shape}')
      print(f'y_test shape: {y_test.shape}')

      X_train shape: (455, 30)
      y_train shape: (455,)
      X_test shape: (114, 30)
      y_test shape: (114,)
```

Figure 1.2: Shape of the training and test sets

1.3 Implementing the k-NN algorithm

1.3.1 Selecting the value of k

Value of k is a hyperparameter of the k-NN algorithm. Generally the value of k is selected as an odd number adjacent to the square root of the number of samples in the training set. In our case, the number of samples in the training set is 455. So, we will select the value of k as 21.

```
[57]: k = int(math.sqrt(len(X_train)))  
      print(f'k = {k}')
```

```
k = 21
```

Figure 1.3: Selecting the value of k

1.3.2 Calculating the distance between two samples

There are several ways to calculate the distance between two samples. We will be using the Euclidean distance to calculate the distance between two samples. For Euclidean distance, we have to pass the following parameters to *KNeighborsClassifier* class: parameter *metric = euclidean*

```
classifier = KNeighborsClassifier(  
    n_neighbors=21,  
    metric='euclidean')
```

1.3.3 Training the model

Now we can train the model using the following code snippet:


```
[78]: classifier = KNeighborsClassifier(n_neighbors=k, p=2, metric='euclidean')
      classifier.fit(X_train, y_train)

[78]: KNeighborsClassifier
      KNeighborsClassifier(metric='euclidean', n_neighbors=21)
```

Figure 1.4: Fitting the model

1.3.4 Predicting the output

Now we can predict the output using the following code snippet:

```
y_pred = classifier.predict(X_test)
```

1.4 Calculating the accuracy of the model

There are several ways to calculate the accuracy of the model. We will be using the *confusion matrix*, *accuracy score* and *f1 score* to calculate the accuracy of the model.

1.4.1 Confusion matrix

The confusion matrix is a technique used for summarizing the performance of a classification algorithm i.e. it has binary outputs. It is a table with 4 different combinations of predicted and actual values.

```
cm = confusion_matrix(y_test, y_pred)
cm_display = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = [False, True])
cm_display.plot()
```

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29fb3bed0>

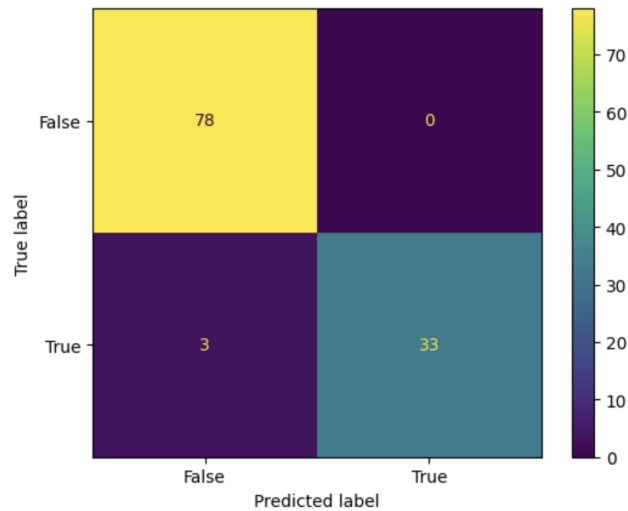


Figure 1.5: Confusion matrix

1.4.2 Accuracy score and f1 score

```
[93]: print(accuracy_score(y_test, y_pred))
0.9736842105263158
```

```
[92]: print(f1_score(y_test, y_pred))
0.9565217391304348
```

Figure 1.6: Score

1.4.3 Relation between the value of k and the accuracy of the model

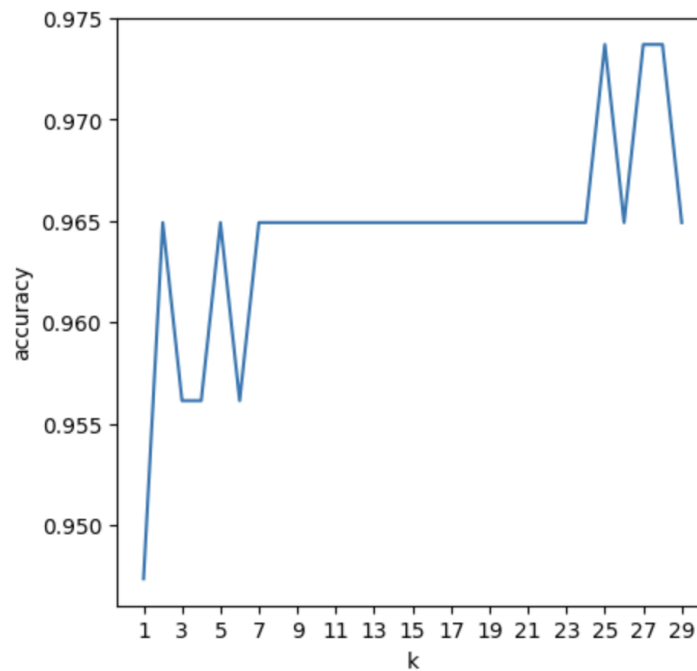


Figure 1.7: Relation between the value of k and the accuracy of the model

1.5 Drawbacks of the k-NN algorithm

Some salient drawbacks of the k-NN algorithm are:

- The k-NN algorithm is very sensitive to outliers.
- The k-NN algorithm is computationally expensive.
- The k-NN algorithm is not suitable for high dimensional data.
- The k-NN algorithm is not suitable for imbalanced data.
- The k-NN algorithm is not suitable for missing data.
- The k-NN algorithm is not suitable for large datasets.
- The k-NN algorithm is not suitable for noisy data.

1.6 Conclusion

In this lab, we have implemented the k-NN algorithm using the *scikit-learn* library. We have also calculated the accuracy of the model using the *confusion matrix*, *accuracy score* and *f1 score*. We have also discussed the drawbacks of the k-NN algorithm.