GHARDA INSTITUTE OF TECHNOLOGY

Department of Computer Engineering

Machine Learning Lab BE Computer (Semester-VII)

Experiment No.4 : k-Nearest Neighbor Classifier

Aim- To study, understand and implement a kNN classification algorithm.

Theory-

K Nearest Neighbors Classification is one of the classification techniques based on instance-based learning. Models based on instance-based learning to generalize beyond the training examples. To do so, they store the training examples first. When it encounters a new instance (or test example), then they instantly build a relationship between stored training examples and this new instant to assign a target function value for this new instance. Instance-based methods are sometimes called lazy learning methods because they postponed learning until the new instance is encountered for prediction.

Instead of estimating the hypothetical function (or target function) once for the entire space, these methods will estimate it locally and differently for each new instance to be predicted.

K-Nearest Neighbors Classifier Learning

Basic Assumption:

1. All instances correspond to points in the n-dimensional space where n represents the number of features in any instance.
2. The nearest neighbors of an instance are defined in terms of the Euclidean distance.

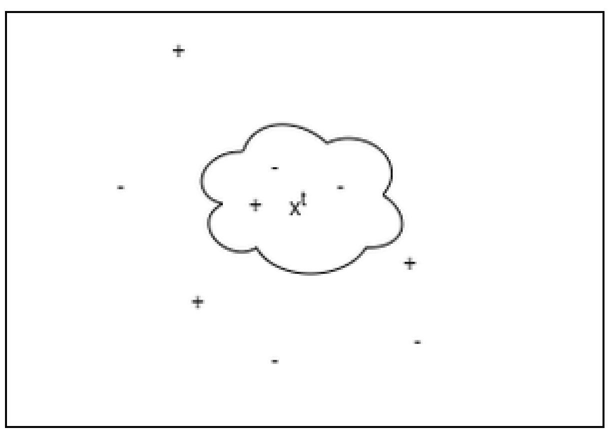
An instance can be represented by < x1, x2, ..., xn >. Euclidean distance between two instances xa and xb is given by d(xa, xb) :

$$\sqrt{\sum_{j=1}^{n}\left(x_j^a - x_j^b\right)^2}$$

How does it work?

K-Nearest Neighbors Classifier first stores the training examples. During prediction, when it encounters a new instance (or test example) to predict, it finds the K number of training instances nearest to this new instance. Then assigns the most common class among the K-Nearest training instances to this test instance.

The optimal choice for K is by validating errors on test data. K can also be chosen by the square root of m, where m is the number of examples in the dataset.



KNN Graphical Working Representation

In the above figure, “+” denotes training instances labeled with 1. “-” denotes training instances with 0. Here we classified for the test instance x_t as the most common class among K-Nearest training instances to it. Here we choose K = 3, so x_t is classified as “-” or 0.

Pseudocode:

1. Store all training examples.
2. Repeat steps 3, 4, and 5 for each test example.
3. Find the K number of training examples nearest to the current test example.
4. *y_pred* for current test example = most common class among K-Nearest training instances.
5. Go to step 2.

Code :

```
from google.colab import files
a=files.upload()
import pandas
b = pandas.read_csv("Default.csv")
print(b)
x = b[["balance", "income"]]
y = b["default"]
from sklearn.model_selection import train_test_split
xtr,xte,ytr,yte = train_test_split(x,y, test_size=0.4)
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()
mm.fit(xtr)
trt = mm.transform(xtr)
tet = mm.transform(xte)
```

```
xtr["balance normalised"] = trt[:,0]
xtr["income normalised"] = trt[:,1]
xte["balance normalised"] = tet[:,0]
xte["income normalised"] = tet[:,1]
print(xtr)
print(xte)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3, metric="euclidean")
knn.fit(xtr[["balance normalised","income normalised"]], ytr)
p = knn.predict(xte[["balance normalised","income normalised"]])
from sklearn.metrics import accuracy_score
a = accuracy_score(yte,p)
print("Accuracy is : ", a)

from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()
mm.fit(xtr)
trt = mm.transform(xtr)
tet = mm.transform(xte)
xtr["balance normalised"] = trt[:,0]
xtr["income normalised"] = trt[:,1]
xte["balance normalised"] = tet[:,0]
xte["income normalised"] = tet[:,1]
print(xtr)
print(xte)
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3, metric="euclidean")
knn.fit(xtr[["balance normalised","income normalised"]], ytr)
p = knn.predict(xte[["balance normalised","income normalised"]])
from sklearn.metrics import accuracy_score
a = accuracy_score(yte,p)
print("Accuracy is : ", a)
```

Output :

```
from google.colab import files
a=files.upload()
```

Choose Files No file chosen   Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Default.csv to Default.csv

```
x = b[["balance", "income"]]
y = b["default"]
from sklearn.model_selection import train_test_split
xtr,xte,ytr,yte = train_test_split(x,y, test_size=0.4)
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()
mm.fit(xtr)
trt = mm.transform(xtr)
tet = mm.transform(xte)
xtr["balance normalised"] = trt[:,0]
xtr["income normalised"] = trt[:,1]
xte["balance normalised"] = tet[:,0]
xte["income normalised"] = tet[:,1]
print(xtr)
print(xte)
```

```
          balance        income  balance normalised  income normalised
5430  1335.935254  19482.20628            0.503306           0.257071
4828  1462.223173  29574.23457            0.550884           0.395732
9855   761.553725  46836.67227            0.286911           0.632911
2106  1277.793381  35620.15600            0.481401           0.478801
7561   660.244842  40885.84695            0.248743           0.551149
...           ...           ...                 ...                ...
3226   224.742276  25386.99487            0.084670           0.338201
6571     0.000000  54298.85961            0.000000           0.735439
3515   738.194410  35134.26535            0.278110           0.472125
5416   542.765602  52625.41571            0.204484           0.712446
1476  1319.187613  35466.24186            0.496996           0.476686

[6000 rows x 4 columns]
          balance        income  balance normalised  income normalised
2352   993.302551  28501.85601            0.374221           0.380998
9654  2128.795992  42096.50237            0.802011           0.567783
```

```
[6000 rows x 4 columns]
          balance        income  balance normalised  income normalised
2352   993.302551  28501.85601            0.374221           0.380998
9654  2128.795992  42096.50237            0.802011           0.567783
8775   154.875532  20685.45742            0.058348           0.273604
634    304.599224  40785.98918            0.114756           0.549777
936    824.731704  42313.51575            0.310713           0.570765
...           ...          ...                 ...                ...
7514   388.609229  23040.44174            0.146406           0.305960
9979   173.249172  30697.24506            0.065271           0.411162
656     49.209666  30451.15286            0.018539           0.407780
6034  1240.664250  30938.53745            0.467413           0.414477
6038     0.000000  34701.19596            0.000000           0.466174

[4000 rows x 4 columns]
```

```python
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3, metric="euclidean")
knn.fit(xtr[["balance normalised","income normalised"]], ytr)
p = knn.predict(xte[["balance normalised","income normalised"]])
from sklearn.metrics import accuracy_score
a = accuracy_score(yte,p)
print("Accuracy is : ", a)
```

```
Accuracy is :  0.9685
```

Results -

Testing Samples	K value	Metric	Acuuracy

Discussion -

Disadvantage

Instance Learning models are computationally very costly because all the computations are done during prediction. It also considers all the training examples for the prediction of every test example.

Conclusion-

The k-NN algorithm is used as a classifier and may also be used as regression.