K Nearest Neighbours Classifier

Learn how a Simple Lazy Learning Algorithm Works

Contents

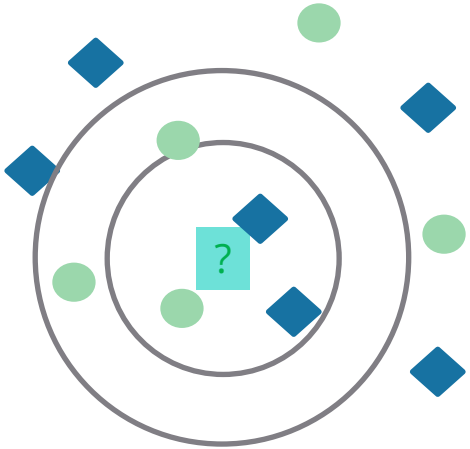
KNN for Classification

- Training dataset has 11 observations belonging to two categories.
- 12th observation is introduced, class of which is not known.
- Nearest neighbour algorithm classifies new observation to the class of the training observation closest to it.

When K=1, nearest one case is considered

As we go on increasing K, classification may vary

K	
1	Blue
3	Blue
5	Orange



Three most important components of this method are **Distance** between cases, **Value of K** and **Voting** criteria.

Simple Example To Understand KNN Method

Age	Current Debt	Default
25	40,000	N
35	60,000	N
45	80,000	N
20	20,000	N
35	120,000	N
52	18,000	N
23	95,000	Y
40	62,000	Y
60	100,000	Y
48	220,000	Y
33	150,000	Y
48	**142,000**	**?**

New observation will be classified as “N” or “Y” based on KNN method

Distance Based on Standardised Variables

$$X_s = \frac{X - Min}{Max - Min}$$

Alternatively, $\frac{(X - Mean)}{SD}$ can also be used

Age	Current Debt	Default	Distance
0.125	0.11	N	0.7652
0.375	0.21	N	0.5200
0.625	0.31	N	0.3160
0	0.01	N	0.9245
0.375	0.50	N	0.3428
0.8	0.00	N	0.6220
0.075	0.38	Y	0.6669
0.5	0.22	Y	0.4437
1	0.41	Y	0.3650
0.7	1.00	Y	0.3861
0.325	0.65	Y	0.3771
0.7	0.61	?	

New observation will be classified as “N”

Selection of K

The second component of KNN model is selecting the appropriate value for K

- If K = 1, the case is classified using the nearest neighbour
- However, K is usually greater than 1. **Consider the following when choosing K :**
  - Mostly odd numbered K is preferred to avoid tie.
  - For a very large K the classifier may result in misclassification, as group of nearest neighbours may include data points which are actually located far away from it.

Thumb Rule :

K = sqrt (n)

n is the number of observations in training data

Voting Criteria

Most common criteria for classification decision is **Majority Voting.**

Frequency of each class in K instances is measured. Class having the highest frequency is attributed to the new case.

Eg. Suppose for K = 7, 4 cases belong to class A and 3 to class B. New case is given class A

Drawback:

Classification is inappropriate when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number.

?

Is there a way to correct this?
One option to remove this drawback is to create training data with equal class frequency. However, this is possible only if data is very large.

Case Study – Predicting Loan Defaulters

Background

- The bank possesses demographic and transactional data of its loan customers. If the bank has a robust model to predict defaulters it can undertake better resource allocation.

Objective

- To predict whether the customer applying for the loan will be a defaulter

Available Information

- Sample size is 389
- **Age group, Years at current address, Years at current employer, Debt to Income Ratio, Credit Card Debts, Other Debts** are the independent variables
- **Defaulter** (=1 if defaulter, 0 otherwise) is the dependent variable

Data Snapshot

BANK LOAN KNN

Variables

SN	AGE	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER

Column	Description	Type	Measurement	Possible Values
SN	Serial Number		-	-
AGE	Age Groups	Categorical	1(<28 years),2(28-40 years),3(>40 years)	3
EMPLOY	Number of years customer working at current employer	Continuous	-	Positive value
ADDRESS	Number of years customer staying at current address	Continuous	-	Positive value
DEBTINC	Debt to Income Ratio	Continuous	-	Positive value
CREDDEBT	Credit Card Debt	Continuous	-	Positive value
OTHDEBT	Other Debt	Continuous	-	Positive value
DEFAULTER	Whether customer defaulted on loan	Binary	1(Defaulter), 0(Non-Defaulter)	2

KNN Classification in Python

```
# Importing the Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion_matrix, f1_score,
precision_score, recall_score, accuracy_score,
roc_curve, roc_auc_score
```

KNN Classification in Python

```
# Importing the Data
bankloan = pd.read_csv("BANK LOAN KNN.csv")

# Preparing data by removing unwanted variables
bankloan1 = bankloan.drop(['SN','AGE'], axis = 1)

bankloan1.head()
```

- **drop()** is used to remove unwanted variables. AGE is removed because it is a categorical variable.
- **axis = 1** drops columns.

```
# Output
```

	EMPLOY	ADDRESS	DEBTINC	CREDDEBT	OTHDEBT	DEFAULTER
0	17	12	9.3	11.36	5.01	1
1	2	0	17.3	1.79	3.06	1
2	12	11	3.6	0.13	1.24	0
3	3	4	24.4	1.36	3.28	1
4	24	14	10.0	3.93	2.47	0

KNN Classification in Python

```
# Creating Train and Test Datasets
X = bankloan1.loc[:,bankloan1.columns != 'DEFAULTER']
y = bankloan1.loc[:, 'DEFAULTER']



X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.30,
                                                    random_state = 999)
```

- **train_test_split()** from sklearn.model_selection is used to split dataset into random train and test sets.
- **test_size** represents the proportion of dataset to be included in the test set
- **random_state** sets the seed for the random number generator

KNN Classification in Python

```
# Preparing Variables
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
X_train
```

- **StandardScaler()** from sklearn.preprocessing is a generic function used for centering or scaling columns of a numeric matrix. The default method for scaling is (X-Mean)/SD.

```
# Output
```

```
array([[-0.89496854, -0.29977261,  1.11865332, -0.28678534,  0.00538471],
       [-0.23076269, -1.21765047,  1.07813931,  1.32686881, -0.1734379 ],
       [ 0.26739169,  0.00618668,  0.26785916, -0.20818328,  0.80582882],
       ...,
       [-0.06471123,  1.07704418, -0.69097236, -0.3191509 , -0.42889879],
       [-0.56286562,  0.15916632,  0.28136383, -0.18044137, -0.50553705],
       [-1.22707147, -0.14679297,  0.24084982,  0.2356872 , -0.40051425]])
```

- All the continuous predictors are now scaled to mean=0 and sd=1.
- Note: Test data is transformed using the train data parameters

KNN Classification in Python

```
# Building the KNN Classifier (Continuous Predictors)
KNNclassifier = KNeighborsClassifier(n_neighbors =
                                     int(np.sqrt(len(X)).round()))

# using thumb rule k ~= sqrt(n)
KNNclassifier.fit(X_train, y_train)
```

- **KNeighborsClassifier()** from sklearn.neighbors performs k-nearest neighbour classification
- **n_neighbors=** specifies the value of k.

```
# Output
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=20, p=2,
                     weights='uniform')
```

KNN Classification in Python

```
# Predictions on Test Data
y_pred = KNNclassifier.predict(X_test)
```

predict() predicts the class labels of the provided data. The default threshold is 0.5.

```
# Confusion Matrix
confusion_matrix(y_test, y_pred, labels=[0, 1])
array([[49, 7],
       [24, 37]])
accuracy_score(y_test, y_pred)
0.7350427350427351
precision_score(y_test, y_pred)
0.8409090909090909
recall_score(y_test, y_pred)
0.6065573770491803
```

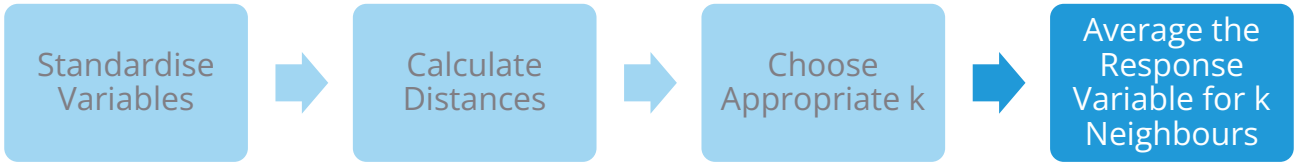
- **accuracy_score() =** number of correct predictions out of total predictions
- **precision_score()** = true positives / (true positives + false positives)
- **recall_score()** also known as 'Sensitivity' = true positives / (true positives + false negatives)

* Note : Output will be slightly different as observations are randomly assigned to train-test data.

KNN for Regression

KNN algorithm can also be extended to regression problems, i.e. **when the dependent variable is continuous**

Process flow for classification and regression is the same, except for the last step



Average value of the response variable for k neighbours is calculated and assigned to the new case.

`KNeighborsRegressor()` from sklearn.neighbors can be used to run k-nearest neighbour regression in Python.

Get an Edge!

- KNN can be used for categorical variables as well.
- Before executing knn on train-test data, categorical variables have to be converted to numeric variables by creating dummy variables.

Quick Recap

In this session, we learnt about **KNN Classifier** :L

KNN for Classification	• Three most important components of this method are **Distance** between cases, **Value of K** and **Voting** criteria.
KNN for Classification in Python	• `KNeighborsClassifier()` from `sklearn.neighbors`.
KNN for Regression	• KNN algorithm can also be extended to regression problems **when the dependent variable is continuous.**
KNN for Regression in Python	• `KNeighborsRegressor()` from `sklearn.neighbors`.