Tech Tunnel

https://ashutoshtripathi.com

A Complete Guide to K-Nearest Neighbours Algorithm – KNN using Python

K-Nearest Neighbours or KNN algorithm is very easy and powerful Machine Learning algorithm. It can be used for both classification as well as regression that is predicting a continuous value. The very basic idea behind KNN is that it starts with finding out the k-nearest data points known as neighbours of the new data point for which we need to make the prediction. And then if it is regression then take the conditional mean of the neighbour's y-value and that is the predicted value for new data point. If it is classification then it takes the mode (majority value) of the neighbours y value and that becomes the predicted class of the new data point.

In this article you will learn how to implement k-Nearest Neighbours or KNN algorithm from scratch using python. Problem described is to predict whether a person will take the personal loan or not. Data set used is from universal bank data set.

Table of Contents

The intuition behind KNN – understand with the help of a graph

Let's start with one basic example and try to understand what the intuition behind the KNN algorithm is. Consider the following example containing a data frame with three columns Height, Age and weight. The values are randomly chosen.

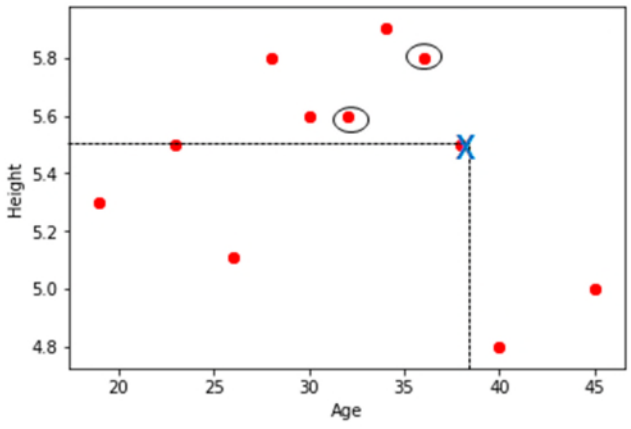
In [28]:
```
import pandas as pd
import matplotlib.pyplot as plt
Age = [45,26,30,34,40,36,19,28,23,32,38]
Height = [5,5.11,5.6,5.9,4.8,5.8,5.3,5.8,5.5,5.6,5.5]
Weight = [77,47,55,59,72,60,40,60,45,58,"?"]
Height_Age_Weight_df = pd.DataFrame({"Height":Height,"Age":Age,"Weight":Weight})
Height_Age_Weight_df
```

Out[28]:

	Height	Age	Weight
0	5.00	45	77
1	5.11	26	47
2	5.60	30	55
3	5.90	34	59
4	4.80	40	72
5	5.80	36	60
6	5.30	19	40
7	5.80	28	60
8	5.50	23	45
9	5.60	32	58
10	5.50	38	?

So now the problem is to predict the weight of a person whose height is 5.50 feet and his age is 38 years (see the 10th row in data set). In the below scatter plot between Height and Age this test point is marked as “x” in blue colour.

In [25]:
```
plt.scatter(Age, Height,color = 'r')
plt.xlabel('Age')
plt.ylabel('Height')
plt.show()
```



So if you look carefully the above scatter plot and observe that this test point is closer to the circled point. And hence its weight will be closer to the weight of these two persons. This is fair enough answer. So these circled points become the neighbours of the test data point. This is the exact idea behind the KNN algorithm. How KNN as an algorithm works? How KNN as an algorithm works?

How KNN as an algorithm works?

Let's take one more example: Consider one Predictor variable x and Target variable y. And we want to predict value of y for x = 13. (See the data below)

```
In [30]: x = [10,13.8,12.5,16,34,23,27,18,30,13]
         y = [11,14.8,13.5,17,35,24,28,19,31,"?"]

         random_df = pd.DataFrame({"X":x,"Y":y})
         random_df
```

Out[30]:

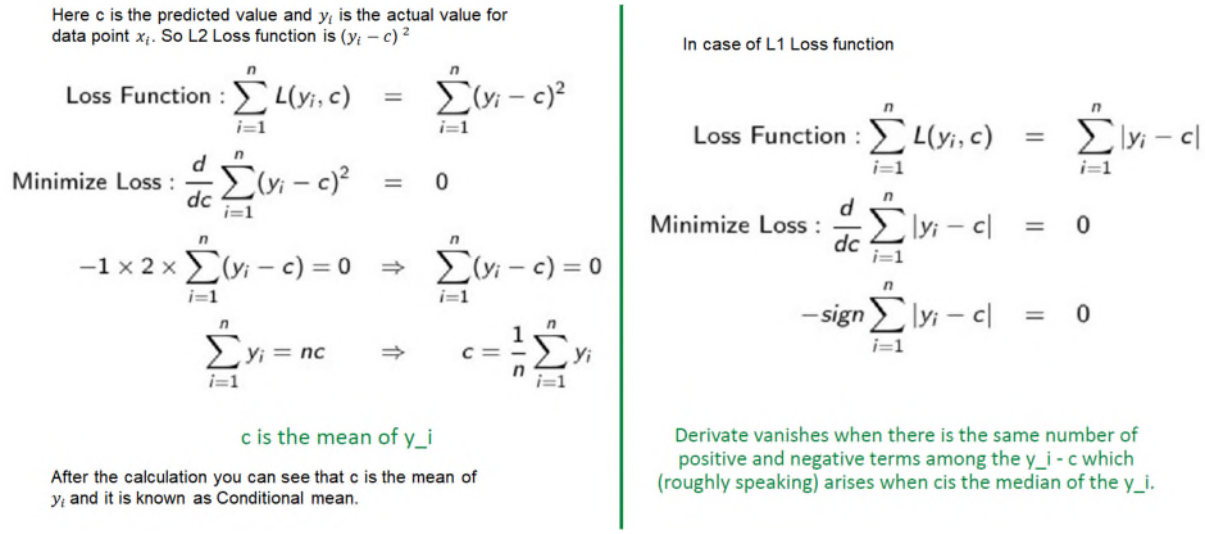
	X	Y
0	10.0	11
1	13.8	14.8
2	12.5	13.5
3	16.0	17
4	34.0	35
5	23.0	24
6	27.0	28
7	18.0	19
8	30.0	31
9	13.0	?

So we will look data points in x which are equal or closer to x= 13. Those are known as neighbours to the new data point. So these points are 12.5, 13.8 and 10 if we take k = 3 nearest neighbours. Now find selected neighbours corresponding y value those are 13.5, 14.8 and 11. Note k is hyper parameter and decision to take how many k will discuss in next heading.

	X	Y
0	10.0	11
1	13.8	14.8
2	12.5	13.5
3	16.0	17
4	34.0	35
5	23.0	24
6	27.0	28
7	18.0	19
8	30.0	31
9	13.0	?

And take mean of those y values as (11+14.8+13.5)/3 = 13.1. So this will be the predicted value for new data point x = 13. Whether we will take mean or median or some other measures it depends on the Loss function. In case of L2 loss that is minimizing the squared error values, we take mean of y values and it is known as conditional mean. If our loss function is of L1 loss then we go with finding median of neighbour's y values.

According to statistics ***"The best prediction of y at an point X = x is the conditional mean in case of L1 Loss and is conditional median in case of L1 Loss"***.



This was the example of predicting a continuous value that is regression problem. KNN can also be used for classification problem. Only the difference will be in this case, we will take the mode of neighbour's y values that is taking the majority of y. For example in above case if we have neighbour's y values as 1, 0, 1 then majority is 1 and hence we will predict our data point x = 13 will belong to class 1. This is how KNN can also be used for classification problems.

How to find the k-Nearest Neighbours?

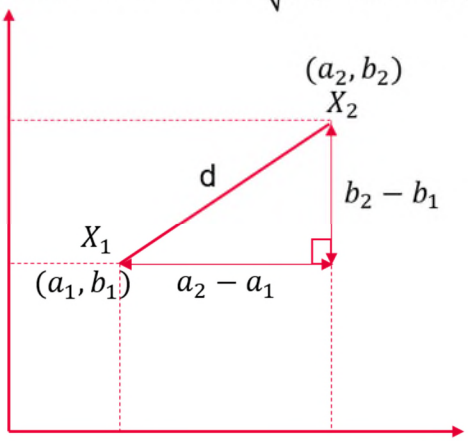
To find the nearest neighbour's, Algorithm calculates the distance between the new data point and training data points. And then shortlist the points which are closer to the new data point. These shortlisted points are known as the nearest neighbours to the new data point. Now two question arises how to calculate the distance between points and how many (k) closer points to be shortlisted. Decision of k will discuss in next heading, now let's understand how to calculate the distance.

The most commonly used distance measures are Euclidean and Manhattan for continuous value prediction that is regression and Hamming Distance for categorical or classification problems.

1. Euclidean Distance

Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (X2) and an existing point (X1).

$$\textbf{Euclidean Distance } d = \sqrt{(a_2 - a_1)^2 + (b_2 - b_1)^2}$$



- X_1 and X_2 are two data points means two different rows in data set. And Data set is two dimensional feature space
- For three dimensional feature space suppose two data points are $X_1(a_1, b_1, c_1)$ *and* $X_2(a_2, b_2, c_2)$ then distance can be calculated as below:

$$\boldsymbol{d} = \sqrt{(\boldsymbol{a_2} - \boldsymbol{a_1})^2 + (\boldsymbol{b_2} - \boldsymbol{b_1})^2 + (\boldsymbol{c_2} - \boldsymbol{c_1})^2}$$

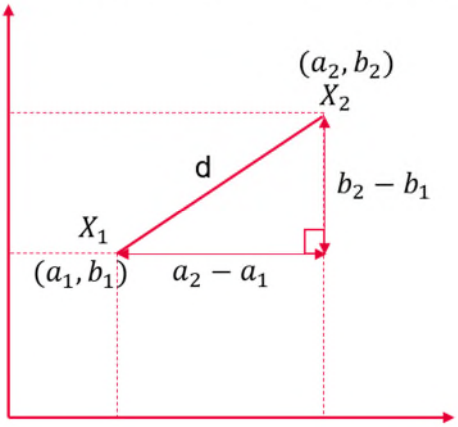
- In case of multi-dimensional vector space, More generic formula will be

$$\textbf{Euclidean Distance } d = \sum_{i=1}^{k} \sqrt{(\boldsymbol{X1_i} - \boldsymbol{X2_i})^2}$$

2. Manhattan Distance

This is the distance between real vectors using the sum of their absolute difference.

$$\textbf{Manhattan Distance } d = |a_2 - a_1| + |b_2 - b_1|$$



- X_1 and X_2 are two data points means two different rows in data set. And Data set is two dimensional feature space
- For three dimensional feature space suppose two data points are $X_1(a_1, b_1, c_1)$ *and* $X_2(a_2, b_2, c_2)$ then distance can be calculated as below:

$$\boldsymbol{d} = |\boldsymbol{a_2} - \boldsymbol{a_1}| + |\boldsymbol{b_2} - \boldsymbol{b_1}| + |\boldsymbol{c_2} - \boldsymbol{c_1}|$$

- In case of multi-dimensional vector space, More generic formula will be

$$\textbf{Manhattan Distance } d = \sum_{i=1}^{k} |\boldsymbol{X1_i} - \boldsymbol{X2_i}|$$

3. Hamming Distance

It is used for categorical variables. If the value (x) and the value (y) are same, the distance D will be equal to 0. Otherwise D=1.

$$D_H = \sum_{i=1}^{k} |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

The Hamming distance between:
- "karolin" and "kathrin" is 3.
- "karolin" and "kerstin" is 3.
- 1011101 and 1001001 is 2.
- 2173896 and 2233796 is 3.

Source: Wikipedia

4. Deciding k – The hyper parameter in KNN

k is nothing but the number of nearest neighbours to be selected to finally predict the outcome of new data point. Decision of choosing the k is very important, although there is no mathematical formula to decide the k.

We start with some random value of k and then start increasing until it is reducing the error in predicted value. Once it start increasing the error we stop there. Also overfitting case need to be taken care here. Sometime we end up choosing large value of k which best suited in training data but drastically increases the error in test or live data. Hence we divide the data in three parts train, validation and test. We select k based on train data and check if it is not overfitting by validating it against validation data.

This procedure required multiple iteration and then finally we get the best suited value of k. However this all we no need to do manually, we can write a function or utilize the inbuilt libraries in python which produces the final k value.

5. Complete end to end example using python

Problem Description

- In the following Supervised Learning activity, we try to predict those who will likely accept the offer of a new personal loan.
- ID: Customer ID
- Age: Customer's age in completed years
- Experience: #years of professional experience
- Income: Annual income of the customer ($000)

- ZIP Code: Home Address ZIP code. Do not use ZIP code Family: Family size of the customer
- CCAvg: Avg. spending on credit cards per month ($000)
- Education: Education Level. 1: Undergrad; 2: Graduate; 3: Advanced/Professional
- Mortgage: Value of house mortgage if any. ($000)
- Personal Loan: Did this customer accept the personal loan offered in the last campaign?
- Securities Account: Does the customer have a securities account with the bank?
- CD Account: Does the customer have a certificate of deposit (CD) account with the bank?
- Online: Does the customer use internet banking facilities?
- Credit Card: Does the customer use a credit card issued by Universal Bank?

5.2 Exploratory data analysis

In [14]:
```
data = pd.read_csv("../UnivBank.csv",na_values = ("?","#","","  ") )
```

In [15]:
```
data.head()
```

Out[15]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0.0	0	1.0	0.0	0	0
1	2	45	19	34	90089	3	1.5	1	0.0	0	1.0	0.0	0	0
2	3	39	15	11	94720	1	1.0	1	0.0	0	0.0	0.0	0	0
3	4	35	9	100	94112	1	2.7	2	0.0	0	0.0	NaN	0	0
4	5	35	8	45	91330	4	1.0	2	0.0	0	0.0	0.0	0	1

In [18]:
```
data.tail()
```

Out[18]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditCard
4995	4996	29	3	40	92697	1	1.9	3	0.0	0	0.0	0.0	1	0
4996	4997	30	4	15	92037	4	0.4	1	85.0	0	0.0	0.0	1	0
4997	4998	63	39	24	93023	2	0.3	3	0.0	0	0.0	0.0	0	0
4998	4999	65	40	49	90034	3	0.5	2	0.0	0	0.0	0.0	1	0
4999	5000	28	4	83	92612	3	0.8	1	0.0	0	0.0	0.0	1	1

In [19]:
```
data.describe()
```

Out[19]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD A
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	4998.000000	5000.000000	4998.000000	4999.
mean	2500.500000	45.338400	20.104600	73.774200	93152.503000	2.396400	1.937938	1.881000	56.521409	0.096000	0.104442	0.
std	1443.520003	11.463166	11.467954	46.033729	2121.852197	1.147663	1.747659	0.839869	101.727873	0.294621	0.305863	0.
min	1.000000	23.000000	-3.000000	8.000000	9307.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.
25%	1250.750000	35.000000	10.000000	39.000000	91911.000000	1.000000	0.700000	1.000000	0.000000	0.000000	0.000000	0.
50%	2500.500000	45.000000	20.000000	64.000000	93437.000000	2.000000	1.500000	2.000000	0.000000	0.000000	0.000000	0.
75%	3750.250000	55.000000	30.000000	98.000000	94608.000000	3.000000	2.500000	3.000000	101.000000	0.000000	0.000000	0.
max	5000.000000	67.000000	43.000000	224.000000	96651.000000	4.000000	10.000000	3.000000	635.000000	1.000000	1.000000	1.

In [17]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
ID                    5000 non-null int64
Age                   5000 non-null int64
Experience            5000 non-null int64
Income                5000 non-null int64
ZIP Code              5000 non-null int64
Family                5000 non-null int64
CCAvg                 5000 non-null float64
Education             5000 non-null int64
Mortgage              4998 non-null float64
Personal Loan         5000 non-null int64
Securities Account    4998 non-null float64
CD Account            4999 non-null float64
Online                5000 non-null int64
CreditCard            5000 non-null int64
dtypes: float64(4), int64(10)
memory usage: 547.0 KB
```

In [22]:
```
data.isnull().sum()
```

Out[22]:
```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              2
Personal Loan         0
Securities Account    2
CD Account            1
Online                0
CreditCard            0
dtype: int64
```

5.3 Imputing missing values

In [ ]:
```
#Fill NA values with the mean of that column values.
#We chosen mean because column is of Float data type.
#If it is categorical then we can go with Mode instead of mean
```

In [141]:
```
mean_mortgage = data['Mortgage'].mean()
data['Mortgage'].fillna(mean_mortgage, inplace = True)
```

In [142]:
```
mode_Securities_Account = data['Securities Account'].mode()
mode_Securities_Account
data['Securities Account'].fillna(mode_Securities_Account[0], inplace = True)
```

In [143]:
```
mode_CD_Account = data['CD Account'].mode()
data['CD Account'].fillna(mode_CD_Account[0], inplace = True)
```

In [127]:
```
data.isnull().sum()
```

Out[127]:
```
ID                    0
Age                   0
Experience            0
Income                0
ZIP Code              0
Family                0
CCAvg                 0
Education             0
Mortgage              0
Personal Loan         0
Securities Account    0
CD Account            0
Online                0
CreditCard            0
dtype: int64
```

5.4 Data Pre-processing

```
In [290]: data.describe() #will describe only numeric attributes
```

Out[290]:

	ID	Age	Experience	Income	ZIP Code	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	Ac
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.
mean	2500.500000	45.338400	20.104600	73.774200	93152.503000	2.396400	1.937938	1.881000	56.521409	0.096000	0.104400	0.
std	1443.520003	11.463166	11.467954	46.033729	2121.852197	1.147663	1.747659	0.839869	101.707521	0.294621	0.305809	0.
min	1.000000	23.000000	-3.000000	8.000000	9307.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.
25%	1250.750000	35.000000	10.000000	39.000000	91911.000000	1.000000	0.700000	1.000000	0.000000	0.000000	0.000000	0.
50%	2500.500000	45.000000	20.000000	64.000000	93437.000000	2.000000	1.500000	2.000000	0.000000	0.000000	0.000000	0.
75%	3750.250000	55.000000	30.000000	98.000000	94608.000000	3.000000	2.500000	3.000000	101.000000	0.000000	0.000000	0.
max	5000.000000	67.000000	43.000000	224.000000	96651.000000	4.000000	10.000000	3.000000	635.000000	1.000000	1.000000	1.

```
In [240]: # In above description see Experience min value is negative
          # and experience can not be less than 0
          # hence will replace all less than 0 values with 0.
```

```
In [275]: data[data['Experience'] < 0] = 0
```

```
In [242]: data.describe()
```

Out[242]:

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	Personal Loan	Securities Account	CD Account	Online	CreditC
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5000.00000	5000.00000	5000.000000	5000.000
mean	45.083400	20.119600	73.046800	2.366600	1.915792	1.859400	56.068009	0.096000	0.10320	0.06040	0.590800	0.291
std	12.173351	11.440484	46.479457	1.167937	1.749641	0.856842	101.454529	0.294621	0.30425	0.23825	0.491735	0.454
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.00000	0.000000	0.000
25%	35.000000	10.000000	38.000000	1.000000	0.670000	1.000000	0.000000	0.000000	0.00000	0.00000	0.000000	0.000
50%	45.000000	20.000000	63.000000	2.000000	1.500000	2.000000	0.000000	0.000000	0.00000	0.00000	1.000000	0.000
75%	55.000000	30.000000	98.000000	3.000000	2.500000	3.000000	100.000000	0.000000	0.00000	0.00000	1.000000	1.000
max	67.000000	43.000000	224.000000	4.000000	10.000000	3.000000	635.000000	1.000000	1.00000	1.00000	1.000000	1.000

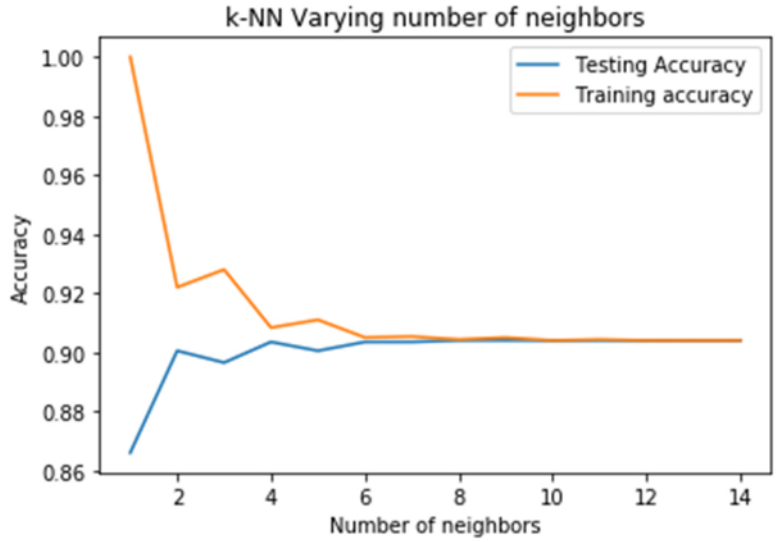
5.5 Train Test split of data

```
1  from sklearn.model_selection import train_test_split
2  X = data.drop('Personal Loan',axis=1).values
3  y = data['Personal Loan'].values
4  X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.4,random_state=42, stratify=y)
```

5.6 Decide K the number of nearest neighbours

```
import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
Setup arrays to store training and test accuracies
neighbors = np.arange(1,15)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
for i,k in enumerate(neighbors):
    #Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)
    #Fit the model
    knn.fit(X_train, y_train)
    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)
    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test)
```

```
#Generate plot
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



From the above plot we can say that we get maximum test accuracy for k = 8 and after that it is constant. Hence we will finalize k as 8 and train the model for 8 nearest neighbours.

5.7 Training the model using KNN

```
In [291]: knn = KNeighborsClassifier(n_neighbors=8)

          #Fit the model
          knn.fit(X_train,y_train)
```

```
Out[291]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                               metric_params=None, n_jobs=1, n_neighbors=8, p=2,
                               weights='uniform')
```

5.8 Predicting on test data

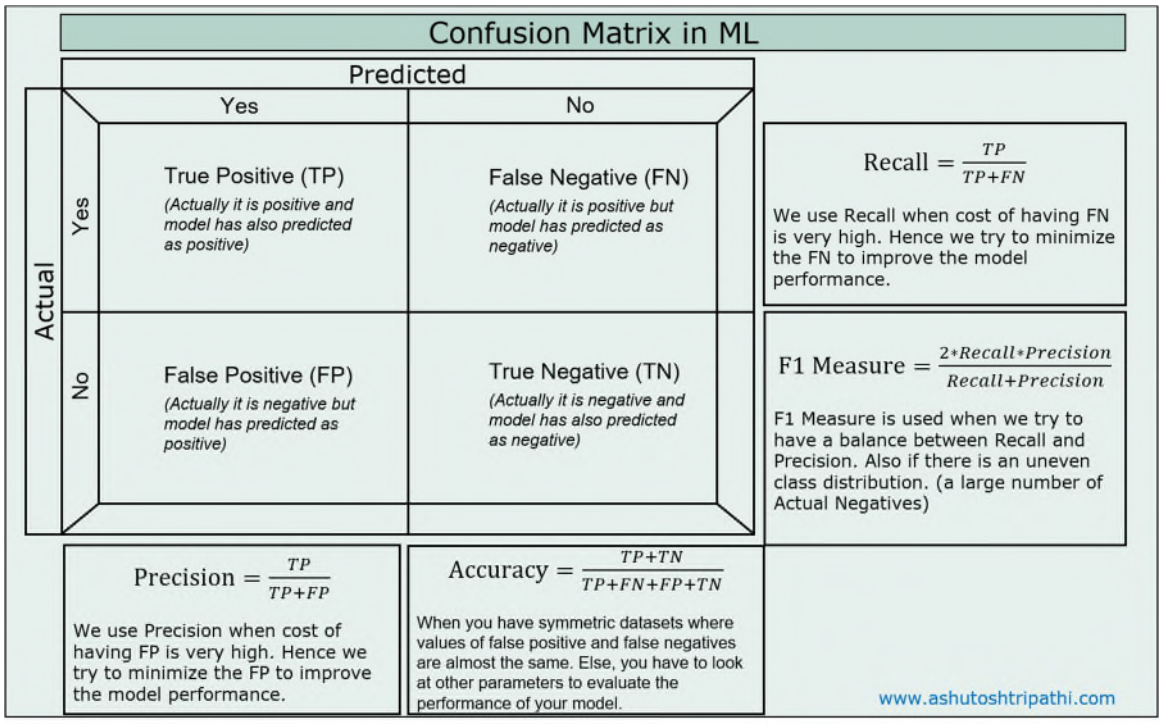
```
1  #Get accuracy.
2  #Note: In case of classification algorithms score method
3  #represents accuracy.
4  knn.score(X_test,y_test)
```

```
In [292]: knn.score(X_test,y_test)
```

```
Out[292]: 0.904
```

5.9 Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. Scikit-learn provides facility to calculate confusion matrix using the confusion matrix method.



```
In [294]: from sklearn.metrics import confusion_matrix

          #let us get the predictions using the classifier we had fit above
          y_pred = knn.predict(X_test)

          confusion_matrix(y_test,y_pred)

Out[294]: array([[1808,    0],
                 [ 192,    0]], dtype=int64)
```

5.10 Classification Report

Another important report is the Classification report. It is a text summary of the precision, recall, F1 score for each class. Scikit-learn provides facility to calculate Classification report using the classification report method.

```
In [295]: from sklearn.metrics import classification_report
          print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
0	0.90	1.00	0.95	1808
1	0.00	0.00	0.00	192
avg / total	0.82	0.90	0.86	2000

Some Important Points

- KNN is Model free algorithm. As there is no assumption on form of function f. You see in regression at last we get a function in terms of x and y including coefficients and constant values.
- Computational complexity increases with increase in data set size
- It suffers with “Curse of Dimensionality” problem. Because when dimensions increases it produces less accuracy.
- Increase the neighbours, it creates smoother boundaries to classify correctly.

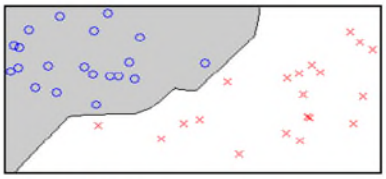
Improving (Speeding up) KNN

Clustering as a Pre-processing Step

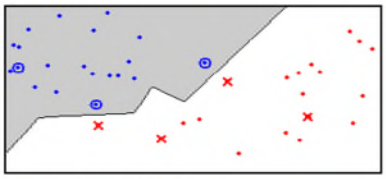
- Eliminate most points (keep only cluster centroids)
- Apply KNN

Condensed NN

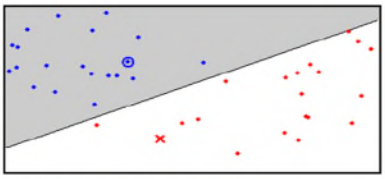
- Retain samples closest to “decision boundaries”
- Decision Boundary Consistent – a subset whose nearest neighbour decision boundary is identical to the boundary of the entire training set
- Minimum Consistent Set – the smallest subset of the training data that correctly classifies all of the original training data



Original data



Condensed data



Minimum Consistent Set

Reduced NN

- Remove a sample if doing so does not cause any incorrect classifications
- Initialize subset with a single training example
- Classify all remaining samples using the subset, and transfer any incorrectly classified samples to the subset
- Return to 2 until no transfers occurred or the subset is full

Thank You

For more Articles please visit -> https://ashutoshtripathi.com