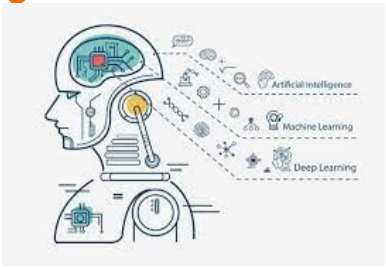
Falling Detection

Machine Learning project using kNN+Python+Colab

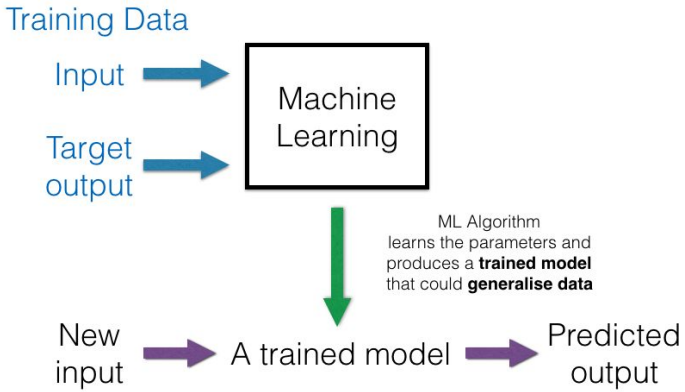


Presented by
Sai Harshinee Roopakula
19577

Table of Contents

Introduction - Machine Learning

- Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.
- Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.
- The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide.
- The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.



Introduction - Machine Learning Types

Machine learning algorithms are often categorized as Supervised Learning, Unsupervised Learning, Semi-supervised Learning and Reinforcement Learning.

1. Supervised machine learning algorithms - Need a lot of labelled data
Supervised learning is where you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X)$$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.
It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance.

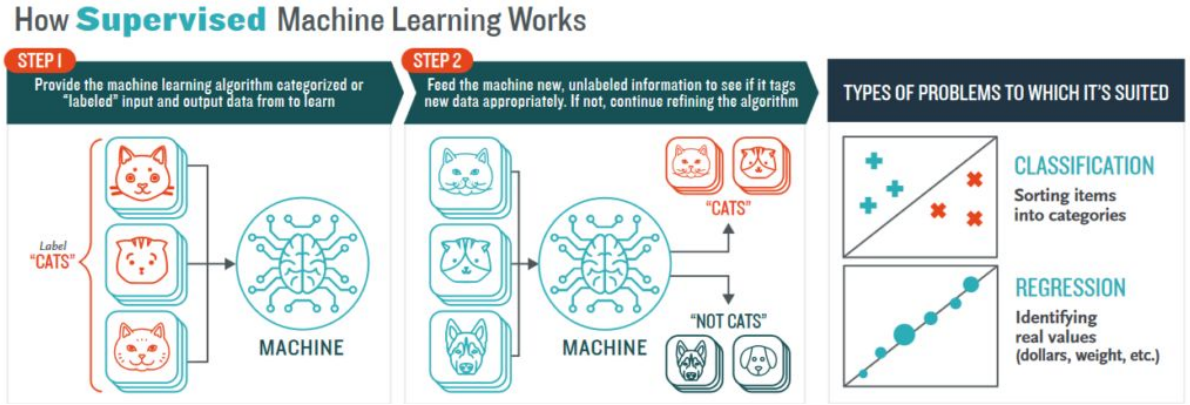
Supervised learning problems can be further grouped into regression and classification problems.

- **Classification**: A classification problem is when the output variable is a category, such as "red" or "blue" or "disease" and "no disease".
- **Regression**: A regression problem is when the output variable is a real value, such as "dollars" or "weight".

Introduction - Machine Learning Types

List of common Supervised Machine Learning Algorithms are:

- Linear Regression
- Logistic Regression
- k-Nearest Neighbors
- Support Vector Machines(SVMs)
- Decision Trees
- Random Forests
- Naive Bayers

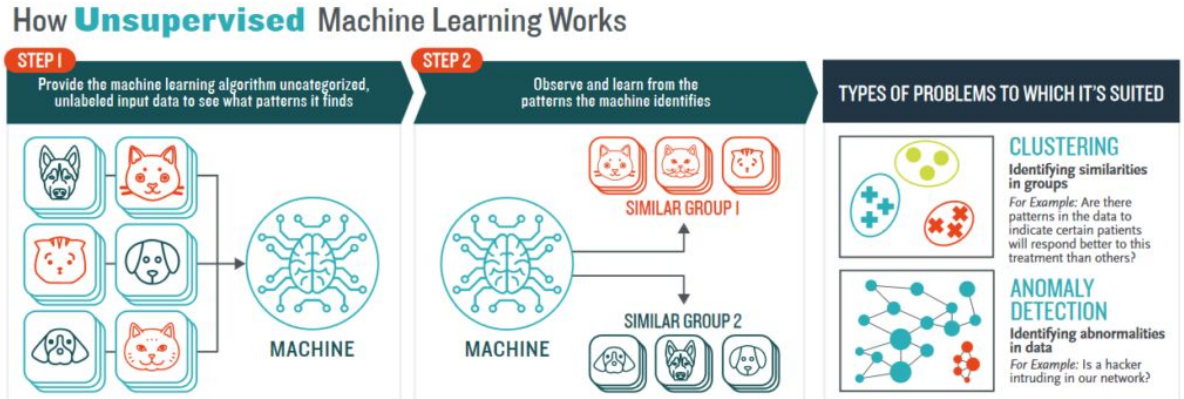


Introduction - Machine Learning Types

2. Unsupervised machine learning algorithms- Need no labelled data.
These are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data.
These are called unsupervised learning because unlike supervised learning there is no correct answers and there is no teacher. Algorithms are left to their own to discover and present the interesting structure in the data.

List of common Unsupervised Machine Learning Algorithms are:

- K-means Clustering
- Association Rules



Introduction - Machine Learning Types

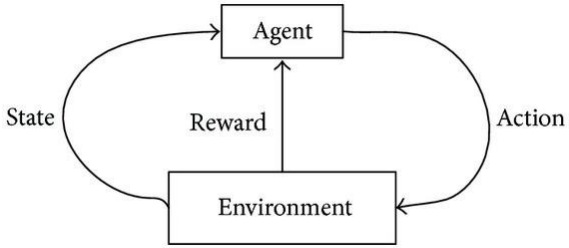
3. Semi-supervised machine learning algorithms
These fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data.
The systems that use this method are able to considerably improve learning accuracy.
A good example is a photo archive where only some of the images are labeled, (e.g. dog, cat, person) and the majority are unlabeled.

4. Reinforcement machine learning algorithms
This method aims at using observations gathered from the interaction with the environment to take actions that would maximize the reward or minimize the risk. Reinforcement learning algorithm (called the agent) continuously learns from the environment in an iterative fashion. In the process, the agent learns from its experiences of the environment until it explores the full range of possible states.

List of common Reinforcement machine Learning Algorithms are:

- Q-Learning
- Temporal Difference (TD)
- Deep Adversarial Networks

Self-driving cars, Robotic hands, Computer played board games (Chess, Go) use Reinforcement machine learning Algorithm.



Design-Understanding the project

Most mobile devices are equipped with different kind of sensors.

We can use the data sent from Gyroscope sensor and Accelerometer sensor to categorize any motion:

- **Gyroscope sensor** measures angular velocity. Angular Velocity is measurement of speed of rotation.
- **Accelerometer sensor** measures the vibration or acceleration of motion of a structure.

Injuries due to unintentional falls cause high social cost in which several systems have been developed to reduce them. Recently, two trends can be recognized.
Firstly, the market is dominated by fall detection systems, which activate an alarm after a fall occurrence, but the focus is moving towards predicting and preventing a fall, as it is the most promising approach to avoid a fall injury.
Secondly, personal devices, such as smartphones, are being exploited for implementing fall systems, because they are commonly carried by the user most of the day.

Here, in this project we are getting training and testing data from Gyroscope and Accelerometer sensor from a smartphone.
After collecting the data, we get 3 values of x,y,z axis positions from Accelerometer sensor and Gyroscope sensor each and also record where it's a fall or not

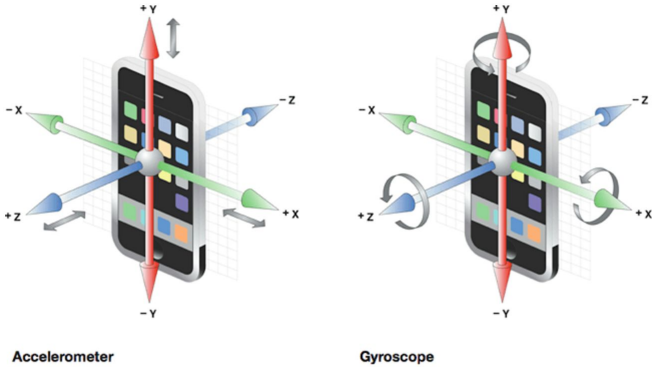
Fall meaning +
No Fall meaning -

Design-Project Dataset

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)
x	y	z	x	y	z	+/-
1	2	3	2	1	3	-
2	1	3	3	1	2	-
1	1	2	3	2	2	-
2	2	3	3	2	1	-
6	5	7	5	6	7	+
5	6	6	6	5	7	+
5	6	7	5	7	6	+
7	6	7	6	5	6	+
7	6	5	5	6	7	??

INPUT :The dataset given, has 8 sets of data, for which output is defined that is whether it is Fall(+) or Not Fall(-).

DESIRED OUTPUT: Assuming the smartphone provided with a new data [7,6,5,5,6,7], we need to predict if the person will fall or not.



Accelerometer Gyroscope

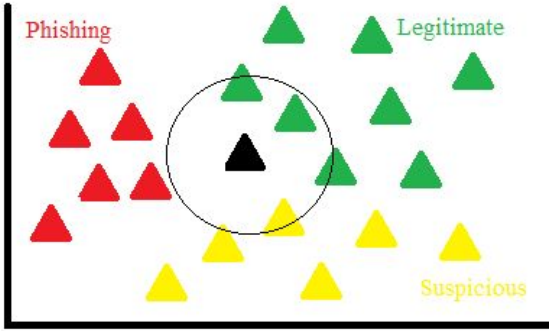
Design-kNN Algorithm

The k-nearest neighbors (kNN) is a simple supervised machine learning algorithm which can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. To evaluate any technique we generally look at 3 important aspects:

1. Ease to interpret output
2. Calculation time
3. Predictive Power

	Logistic Regression	CART	Random Forest	KNN
1. Ease to interpret output	2	3	1	3
2. Calculation time	3	2	1	3
3. Predictive Power	2	2	3	2

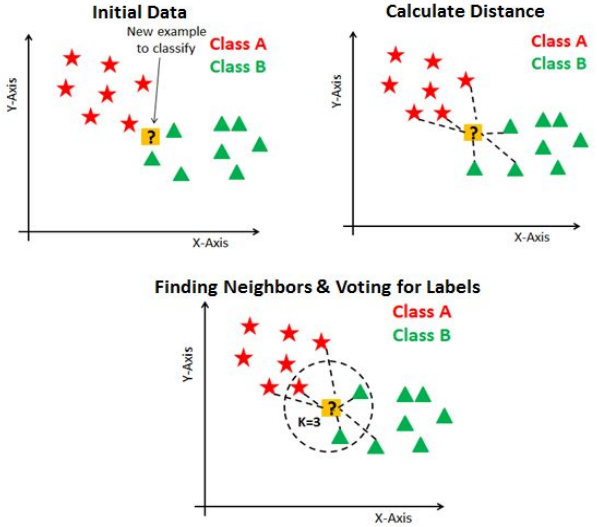
kNN algorithm fairs across all the parameters of considerations.
It is commonly used for its ease of interpretation and low calculation time.



Design-kNN Algorithm Steps

We can implement a KNN model by following the below steps:

1. Load the data
2. Initialise the value of k
3. For getting the predicted class, iterate from 1 to total number of training data points
    1. Calculate the distance between test data and each row of training data. Here we will use Euclidean distance as our distance metric since it's the most popular method. The other metrics that can be used are Chebyshev, cosine, etc.
    2. Sort the calculated distances in ascending order based on distance values
    3. Get top k rows from the sorted array
    4. Get the most frequent class of these rows
    5. Return the predicted class



Implementation - Using kNN manually to predict the result

Step 1 - Load the data

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)
x	y	z	x	y	z	+/-
1	2	3	2	1	3	-
2	1	3	3	1	2	-
1	1	2	3	2	2	-
2	2	3	3	2	1	-
6	5	7	5	6	7	+
5	6	6	6	5	7	+
5	6	7	5	7	6	+
7	6	7	6	5	6	+

In Step 1, we load the data.

```
Dataset = [[1, 2, 3, 2, 1, 3, -],
           [2, 1, 3, 3, 1, 2, -],
           [1, 1, 2, 3, 2, 2, -],
           [2, 2, 3, 3, 2, 1, -],
           [6, 5, 7, 5, 6, 7, +],
           [5, 6, 6, 6, 5, 7, +],
           [5, 6, 7, 5, 7, 6, +],
           [7, 6, 7, 6, 5, 6, +]]
```

Implementation - Using kNN manually to predict the result

Step 2 - Choosing the value of K

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)
x	y	z	x	y	z	+/-
1	2	3	2	1	3	-
2	1	3	3	1	2	-
1	1	2	3	2	2	-
2	2	3	3	2	1	-
6	5	7	5	6	7	+
5	6	6	6	5	7	+
5	6	7	5	7	6	+
7	6	7	6	5	6	+

In Step 2, we choose the value of K. .

K value is usually equal to the odd number closest to the square root of the number of instances. Here as number of instances/samples are 8.

$K = \sqrt{8} = 2.82 \sim 3$.

The K value chosen is equal to 3.

Implementation - Using kNN manually to predict the result

Step 3 - Calculating distance between test data and each training data

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)
x	y	z	x	y	z	+/-
1	2	3	2	1	3	-
2	1	3	3	1	2	-
1	1	2	3	2	2	-
2	2	3	3	2	1	-
6	5	7	5	6	7	+
5	6	6	6	5	7	+
5	6	7	5	7	6	+
7	6	7	6	5	6	+
7	6	5	5	6	7	???

In Step 3, we calculate the Euclidean distance between test data and each training data.

Formula

The distance to each neighbor =
(Targetx1-Datax1)^2+(Targety1-Datay1)^2+
(Targetz1-Dataz1)^2+(Targetx2-Datax2)^2+
(Targety2-Datay2)^2+(Targetz2-Dataz2)^2

The complete data calculation is continued on next page.

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)	Distance to each neighbour
x	y	z	x	y	z	+/-	(7-x1)^2+(6-y1)^2+(5-z1)^2+ (5-x2)^2+(6-y2)^2+(7-z2)^2+
1	2	3	2	1	3	-	106
2	1	3	3	1	2	-	108
1	1	2	3	2	2	-	115
2	2	3	3	2	1	-	101
6	5	7	5	6	7	+	6
5	6	6	6	5	7	+	7
5	6	7	5	7	6	+	10
7	6	7	6	5	6	+	7
7	6	5	5	6	7	???	

The distance to each neighbor =
(Targetx1-Datax1)^2+(Targety1-Datay1)^2+
(Targetz1-Dataz1)^2+(Targetx2-Datax2)^2+
(Targety2-Datay2)^2+(Targetz2-Dataz2)^2

(7-1)^2+(6-2)^2+(5-3)^2+(5-2)^2+(6-1)^2+(7-3)^2 = **106**

(7-2)^2+(6-1)^2+(5-3)^2+(5-3)^2+(6-1)^2+(7-2)^2 = **108**

(7-1)^2+(6-1)^2+(5-2)^2+(5-3)^2+(6-2)^2+(7-2)^2 = **115**

(7-2)^2+(6-2)^2+(5-3)^2+(5-3)^2+(6-2)^2+(7-1)^2 = **101**

(7-6)^2+(6-5)^2+(5-7)^2+(5-5)^2+(6-6)^2+(7-7)^2 = **6**

(7-5)^2+(6-6)^2+(5-6)^2+(5-6)^2+(6-5)^2+(7-7)^2 = **7**

(7-5)^2+(6-6)^2+(5-7)^2+(5-5)^2+(6-7)^2+(7-6)^2 = **10**

(7-7)^2+(6-6)^2+(5-7)^2+(5-6)^2+(6-5)^2+(7-6)^2 = **7**

Implementation - Using kNN manually to predict the result

Step 4 - Pick smallest K numbers in terms of distance calculated in Step 3

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)	Distance to each neighbour
x	y	z	x	y	z	+/-	(7-x1)^2+(6-y1)^2+(5-z1)^2+ (5-x2)^2+(6-y2)^2+(7-z2)^2+
6	5	7	5	6	7	+	6
5	6	6	6	5	7	+	7
7	6	7	6	5	6	+	7
7	6	5	5	6	7	???	

In Step 4, we pick K(3) smallest numbers in terms of Euclidean distance between test data and each training data.

We pick sample sets with the distance 6, 7, 7 respectively as they are the smallest 3 distances from the target/test test.

Implementation - Using kNN manually to predict the result

Step 5 - Make prediction based on the most frequent class from the K samples selected.

Accelerometer Data			Gyroscope Data			Fall (+), Not (-)	Distance to each neighbour
x	y	z	x	y	z	+/-	(7-x1)^2+(6-y1)^2+(5-z1)^2+ (5-x2)^2+(6-y2)^2+(7-z2)^2+
6	5	7	5	6	7	+	6
5	6	6	6	5	7	+	7
7	6	7	6	5	6	+	7
7	6	5	5	6	7	+	

In Step 5, we make prediction based on the most frequent class from the 3 samples selected.

Since all the sets selected from Step 4 i.e., the distance 6, 7, 7 respectively all have the Fall(+),Not(-) as "+", we predict that the test data [7, 6, 5, 5, 6, 7] belongs to the class **Fall(+)**

Implementation - Using Python to implement KNN

Step 1 - Importing necessary libraries

Importing necessary libraries

```
[1]  # Importing sqrt function from math module
     from math import sqrt
```

Step 2 - Calculating Euclidean distance

Calculating Euclidean distance

```
[2]  # calculate the Euclidean distance between two vectors
     # Formula - Euclidean Distance = sqrt(sum i to N (x1_i - x2_i)^2)
     def euclidean_distance(row1, row2):
         distance = 0.0
         for i in range(len(row1)):
             distance += (row1[i] - row2[i])**2
         return sqrt(distance)
```

Implementation - Using Python to implement KNN

Step 3 - Locating the most similar neighbors

Locating the most similar neighbors

```
[3]  # Locate the most similar neighbors
     # Result
     #    [6, 5, 7, 5, 6, 7, 1]
     #    [7, 6, 7, 6, 5, 6, 1]
     #    [5, 6, 6, 6, 5, 7, 1]

     def get_neighbors(train, test_row, num_neighbors):
         distances = list()
         for train_row in train:
             dist = euclidean_distance(test_row, train_row)
             distances.append((train_row, dist))
         distances.sort(key=lambda tup: tup[1])
         neighbors = list()
         for i in range(num_neighbors):
             neighbors.append(distances[i][0])
         return neighbors
```

Implementation - Using Python to implement KNN

Step 4 - Making classification prediction with neighbors

Making Classification prediction with neighbors

```
[4]  # Make a classification prediction with neighbors
     # test_row is [7, 6, 5, 5, 6, 7]
     # num_neighbors is 3
     def predict_classification(train, test_row, num_neighbors):
         neighbors = get_neighbors(train, test_row, num_neighbors)
         output_values = [row[-1] for row in neighbors]
         prediction = max(set(output_values), key=output_values.count)
         return prediction
```

Implementation - Using Python to implement KNN

Step 5 - Sending in the Dataset - Test Data

▾ Dataset - Train and test data

```
[5]  # Test distance function
     # Here, 0 means Not Fall(-), 1 means Fall(+)
     dataset = [[1,2,3,2,1,3,0],
                [2,1,3,3,1,2,0],
                [1,1,2,3,2,2,0],
                [2,2,3,3,2,1,0],
                [6,5,7,5,6,7,1],
                [5,6,6,6,5,7,1],
                [5,6,7,5,7,6,1],
                [7,6,7,6,5,6,1]]
```

Implementation - Using Python to implement KNN

Step 6 - Predicting for new Dataset

Predicting for new dataset

```
[6]  prediction = predict_classification(dataset, [7,6,5,5,6,7], 3)
     # Display
     # Expected 1, Got 1.
     print('Expected %d, Got %d.' % ([7,6,5,5,6,7,1][-1], prediction))
```

```
Expected 1, Got 1.
```

Using Python, we have predicted that the test data [7, 6, 5, 5, 6, 7] belongs to the class 1 i.e., **Fall(+)**

Implementation - Using scikit learn to implement KNN

Step 1: Importing necessary libraries and loading the sample dataset

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
df = pd.read_csv("sample.csv")
```

In [3]:
```
df.head(10)
```

Out[3]:

	x1	y1	z1	x2	y2	z2	FallOrNot
0	1	2	3	2	1	3	-
1	2	1	3	3	1	2	-
2	1	1	2	3	2	2	-
3	2	2	3	3	2	1	-
4	6	5	7	5	6	7	+
5	5	6	6	6	5	7	+
6	5	6	7	5	7	6	+
7	7	6	7	6	5	6	+

Implementation - Using scikit learn to implement KNN

Step 2: Preparing the X and y values

In [4]:
```
X=df.drop("FallOrNot",axis=1)
X
```

Out[4]:

	x1	y1	z1	x2	y2	z2
0	1	2	3	2	1	3
1	2	1	3	3	1	2
2	1	1	2	3	2	2
3	2	2	3	3	2	1
4	6	5	7	5	6	7
5	5	6	6	6	5	7
6	5	6	7	5	7	6
7	7	6	7	6	5	6

In [5]:
```
y=df["FallOrNot"]
y
```

Out[5]:
```
0    -
1    -
2    -
3    -
4    +
5    +
6    +
7    +
Name: FallOrNot, dtype: object
```

Implementation - Using scikit learn to implement KNN

Step 3: Generating the kNN Model using sklearn.model_selection

```
In [6]: from sklearn.model_selection import train_test_split
        X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=5)

In [7]: from sklearn.neighbors import KNeighborsClassifier

In [8]: knn_model = KNeighborsClassifier(n_neighbors=3)

In [9]: knn_model.fit(X_train,y_train)

Out[9]: KNeighborsClassifier(n_neighbors=3)

In [10]: knn_model.get_params()

Out[10]: {'algorithm': 'auto',
          'leaf_size': 30,
          'metric': 'minkowski',
          'metric_params': None,
          'n_jobs': None,
          'n_neighbors': 3,
          'p': 2,
          'weights': 'uniform'}
```

Implementation - Using scikit learn to implement KNN

Step 4: Predicting the test data and performing evaluation on the test data

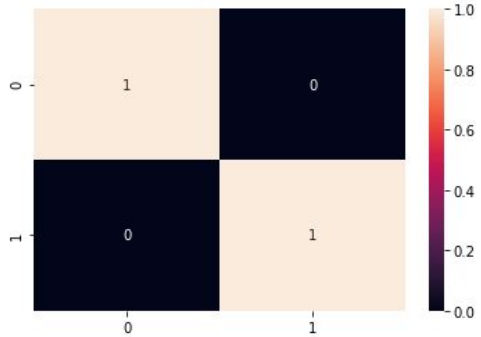
```
In [11]: y_pred=knn_model.predict(X_test)

In [12]: y_pred
Out[12]: array(['+', '-'], dtype=object)

In [13]: from sklearn.metrics import confusion_matrix

In [14]: cm = confusion_matrix(y_test,y_pred)

In [15]: sns.heatmap(cm,annot = True)
Out[15]: <AxesSubplot:>
```



Implementation - Using scikit learn to implement KNN

Step 5: Predicting new data.

```
In [16]: new_value = ([[7,6,5,5,6,7]])

In [17]: knn_model.predict(new_value)

Out[17]: array(['+'], dtype=object)
```

Using scikit-learn, we have predicted that the test data [7, 6, 5, 5, 6, 7] belongs to the class **Fall(+)**

Conclusion

We have implemented kNN Algorithm using 3 methods namely:

- Manually
- Python from scratch
- scikit-learn

It is found all the 3 methods above, gave the same result. It has predicted the new dataset as **Fall(+)**

Manual calculation are cumbersome in nature and has high scope of errors. (To avoid complicated calculations, we have not considered the square root while calculating Euclidean distance.

While, python could be easier and faster compared to manual calculation for large data.

The best method to be used to implement kNN algorithm is using scikit learn. We can import KNeighborsClassifier from sklearn.neighbors

Bibliography

https://www.expert.ai/blog/machine-learning-definition/

https://www.hindawi.com/journals/ahci/2019/9610567/

https://www.researchgate.net/post/How_can_we_find_the_optimum_K_in_K-Nearest_Neighbor

https://npu85.npu.edu/~henry/npu/classes/data_science/algorithm/slide/knn_from_scratch.html

Link to view the presentation

https://docs.google.com/presentation/d/1NJGIZIAUZujmJNI-y2ze3k0QHM1b9HGVgmqMqezgD8E/edit?usp=sharing