National University of Computer & Emerging Sciences, Karachi

FALL 2023, LAB MANUAL – 03

KNN

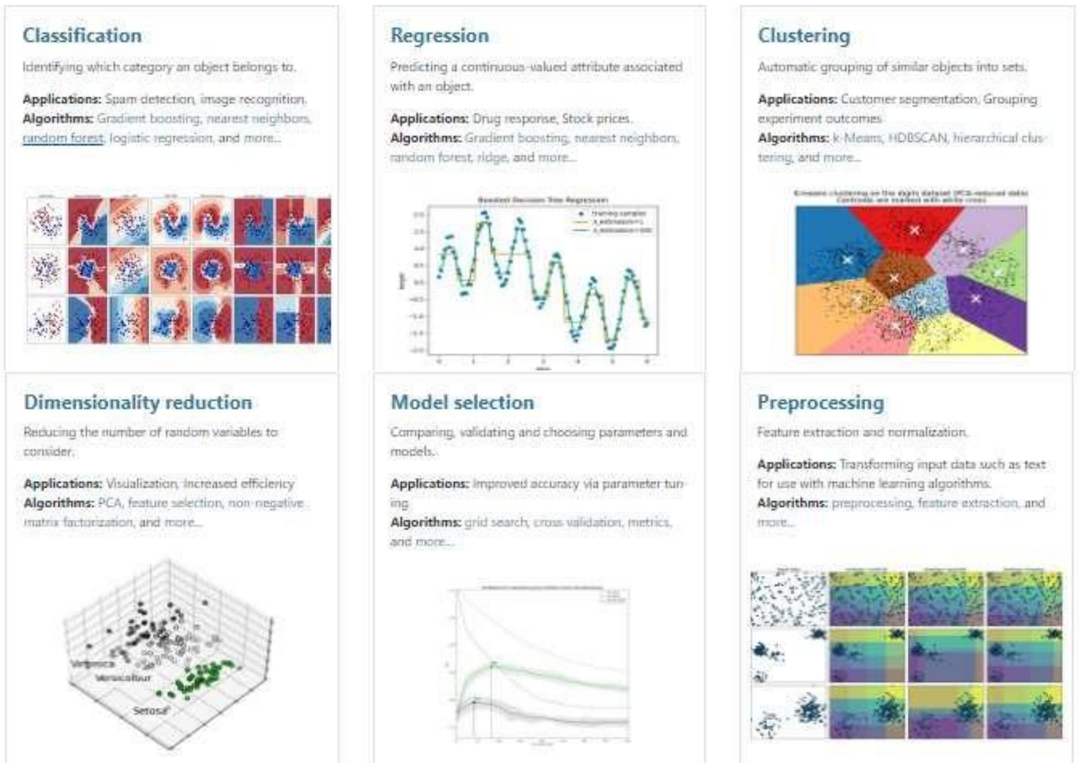
COURSE CODE :	**AL3002**
INSTRUCTOR :	**Usama Bin Umar**

OBJECTIVE

1. To know the type of Machine Learning Algorithms
2. Scikit Learn Library
3. KNN as a Classifier
4. How to implement KNN
5. Parameters in KNN

SCIKIT LEARN

Scikit-learn (sklearn) is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib.



Classification
Identifying which category an object belongs to.
Applications: Spam detection, image recognition.
Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...

Regression
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, Stock prices.
Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...

Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, HDBSCAN, hierarchical clustering, and more...

Dimensionality reduction
Reducing the number of random variables to consider.
Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization, and more...

Model selection
Comparing, validating and choosing parameters and models.
Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

Preprocessing
Feature extraction and normalization.
Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...

The functionality that scikit-learn provides include:

- **Regression,** including Linear and Logistic Regression algorithms etc
- **Classification,** including K-Nearest Neighbors algorithms etc
- **Clustering**, including K-Means and K-Means++ etc Model selection
- **Preprocessing**, including Min-Max Normalization

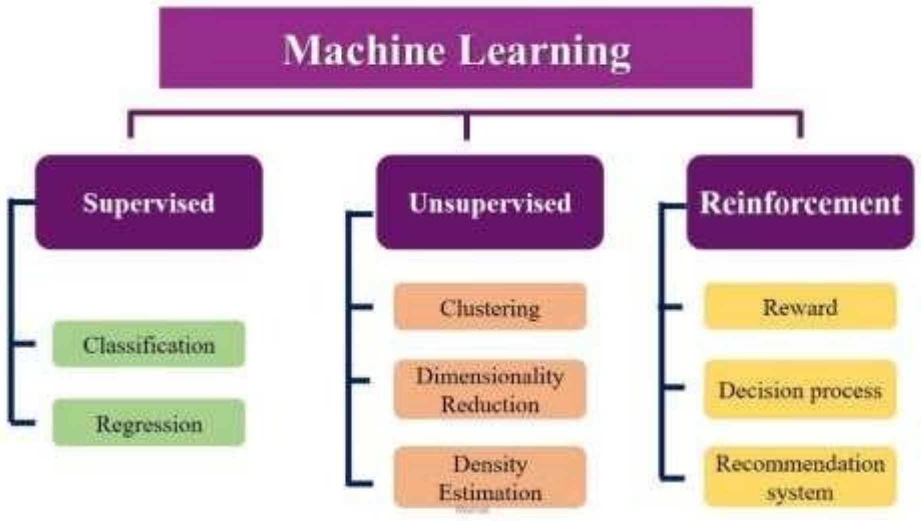


Figure 1 : Shows the Types of learning in ML

Machine learning has three types :

- **Supervised machine learning** (class/label/target is given)
- **Un-supervised machine learning** (class/label/target is not given)
- **Reinforcement Learning** (reward based)

SUPERVISED ALGORITHMS:

Classification algorithms (like KNN, Decision Tree, Naive Bayes etc)
Regression (like linear regression, logistic regression etc)

UNSUPERVISED ALGORITHMS:

Clustering Algorithms (like kmeans, kmeans++ etc)

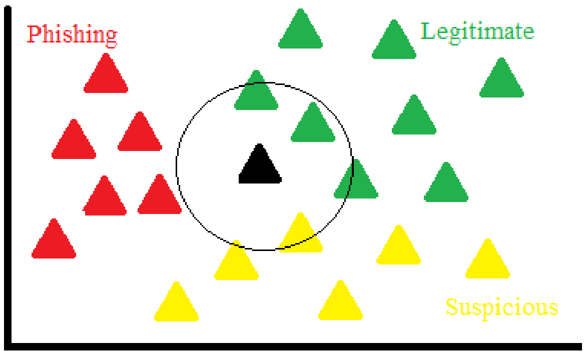
SUPERVISED ALGORITHMS:

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y)**.

K NEAREST NEIGBOURS

The K-Nearest Neighbor (KNN) algorithm is a popular machine learning technique used for classification and regression tasks. It relies on the idea that similar data points tend to have similar labels or values.



During the training phase, the KNN algorithm stores the entire training dataset as a reference. When making predictions, it calculates the distance between the input data point and all the training examples, using a chosen distance metric such as Euclidean distance.

STEPS TO IMPLEMENT KNN

1. Select the number K of the neighbors
2. Calculate the Euclidean distance of K number of neighbors
3. Take the K nearest neighbors as per the calculated Euclidean
4. Among these k neighbors, count the number of the data points in each category.
5. Assign the new data points to that category for which the number of the neighbor is maximum.
6. Our model is ready.

How to Calculate Distance:

The distance metric is the effective hyper-parameter through which we measure the distance between data feature values and new test inputs.

- Euclidean Distance
- Manhattan Distance
- Minkowsi Distance
- Hamming Distance
- Cosine Distance

Euclidean Distance

The Euclidean distance between all points to the centroid is calculated by measuring the distances of the Y and X coordinates to the centroid.

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Splitting Dataset into Input and Output

```python
y=df.pop('target')
```

Splitting Dataset into Training and Testing

```python
from sklearn.model_selection import train_test_split
xtrain, xtest , ytrain ,ytest=train_test_split(df,y,train_size=0.7)
print('x train : ', xtrain.shape)
print('x test : ', xtest.shape)
print('y train : ', ytrain.shape)
print('y test : ', ytest.shape)
```

Implementing KNN

Training model using KNN

```python
from sklearn.neighbors import KNeighborsClassifier
KNN=KNeighborsClassifier()
ModelKNN = KNN.fit(train_X,train_y)
```

Model Testing

```python
PredictionKNN = KNN.predict(test_X)
```

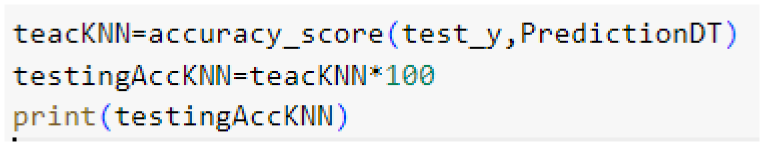
Training Accuracy:
This measures how well a machine learning model performs on the same data it was trained on. It shows how accurately the model can predict outcomes within its training dataset. High training accuracy suggests the model has learned to fit the training data well.

Model Training Accuracy

```python
tracKNN=KNN.score(train_X,train_y)
trainingAccKNN=tracKNN*100
print(trainingAccKNN)
```

Testing Accuracy:
This assesses how well a machine learning model generalizes its predictions to new, unseen data. It indicates how accurately the model can predict outcomes on data it hasn't encountered during training. High testing accuracy suggests the model can make accurate predictions on new, real-world data, making it a valuable measure of the model's generalization performance.

Model Testing Accuracy



```
teacKNN=accuracy_score(test_y,PredictionDT)
testingAccKNN=teacKNN*100
print(testingAccKNN)
```

We measure Training and Testing accuracy to check whether our model is Generalized or not. If the model is not generalized then it will either result in underfitting or overfitting, reason is bias or variance.

Generalization
Generalization in machine learning is the model's capacity to make accurate predictions or classifications on new, unseen data, beyond the examples it was trained on. Poor Generalization results in underfitting or overfitting.

- **Underfitting:**
  A condition where model's training and testing both accuracies are low.
  Model is too simple and fails to capture underlying patterns in the data, resulting in poor performance on both training and new data.
- **Overfitting:**
  A condition where model's training accuracy is good but testing accuracy is low.
  A model is overly complex, fitting the training data too closely, including noise, and performing poorly on new data.

Reason for Underfitting or Overfitting:
Let's discuss what is Bias and Variance Tradeoff
The bias-variance tradeoff in machine learning is a balance between two types of errors:
Bias: This error arises when a model is too simplistic and cannot capture the underlying patterns in the data, leading to underfitting.
Variance: This error occurs when a model is overly complex and fits the training data too closely, capturing noise and leading to overfitting.
Finding the right balance between bias and variance is crucial for building models that generalize well to new data. It involves choosing an appropriate model complexity and employing techniques like regularization and cross-validation to strike the right balance.

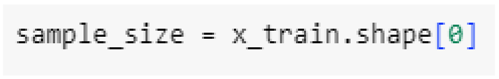
Small K: Low bias, High Variance
Large K: High Bias, Low Variance

How to Selecting K_Value (# of Nearest Neighbors)

- Square root method
- Experimentation

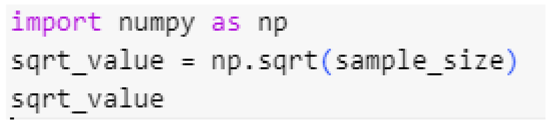
Square Root Method:
Count the total sample size and then apply square root , if the total samples are 64 then by taking its square root we can get 8. Make sure to choose an odd value. Using an odd value for k in KNN classification helps avoid ties in voting, simplifying class label determination, but even k values can be used with tie-breaking rules when necessary.

Checking Total X train Sample Size



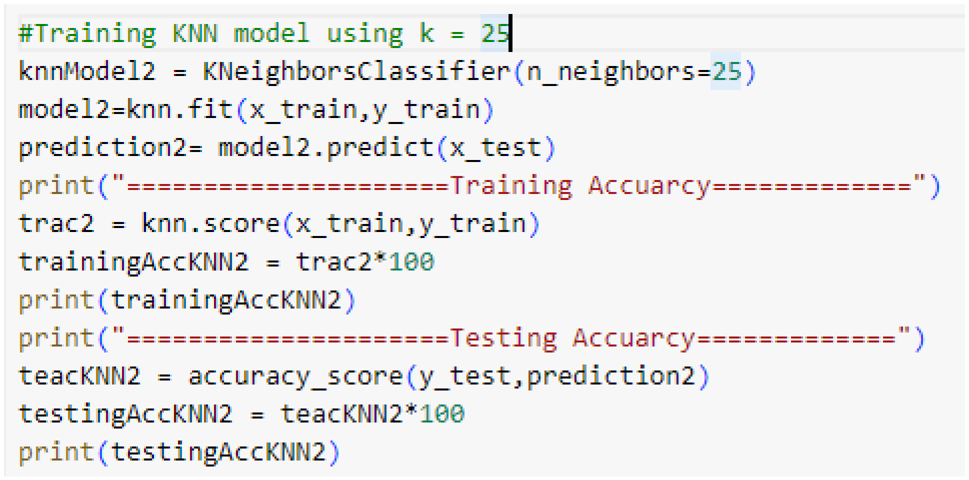
```
sample_size = x_train.shape[0]
```

Checking Total X train Size



```
import numpy as np
sqrt_value = np.sqrt(sample_size)
sqrt_value
```

Choosing K Value = sqrt_value (odd)



```
#Training KNN model using k = 25
knnModel2 = KNeighborsClassifier(n_neighbors=25)
model2=knn.fit(x_train,y_train)
prediction2= model2.predict(x_test)
print("====================Training Accuarcy============")
trac2 = knn.score(x_train,y_train)
trainingAccKNN2 = trac2*100
print(trainingAccKNN2)
print("=====================Testing Accuarcy============")
teacKNN2 = accuracy_score(y_test,prediction2)
testingAccKNN2 = teacKNN2*100
print(testingAccKNN2)
```

Experimentation:
Select a range of K values to test. This can be done by trying a variety of values, for example, from K=1 to K=20. For each K value in your range, train a KNN model on the training data and evaluate its performance on the validation set using a suitable evaluation metric (e.g., accuracy, F1-score, or another metric relevant to your problem). You can also use k-fold cross-validation to get a more robust estimate of performance.

Experimentation

```
#list that will store the accuracy rate value of k
acc=[]
for i in range (1,31):  #Took the range of k from 1 to 30
    clf=KNeighborsClassifier(n_neighbors=i)
    clf.fit(x_train,y_train)
    predict_i=clf.predict(x_test)
    accKNN=accuracy_score(y_test,predict_i)
    acc.append(accKNN)
acc
```

.

PARAMETERS for KNN

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5*, ***, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*) ¶

Read Sklearn Official Documentation, to know more about KNN Parameter.

Changing Metric

```
knn = KNeighborsClassifier(n_neighbors=14, metric='euclidean')
```

Changing Algorithm

```
knn = KNeighborsClassifier(n_neighbors=14, algorithm='ball_tree')
```

TASK 1:

Occupancy dataset contains four attributes i-e "Humidity, Light, CO2 and Humidity ratio".

- Apply KNN to find if occupancy is possible or not (0 or 1) based on "Humidity, Light and Humidity Ratio" only. Train on "Occupancy_train.txt" and Test on "Occupancy_test.txt". You need to do thefollowing then :

- Run•this KNN Algorithm for n_neighbors (K) from 1 to 10. You will get 10 different accuracies.Print all the accuracies. Then print the highest accuracy and also the value of K at which you got the highest accuracy.

TASK 2 :

Now instead of using built-in library, write your own code for kNN classifier from scratch. Runon iris dataset. Use 80/20 split. Print accuracy and confusion matrix at the end. You must use thefollowing chi squared distance function :

$$\sum_{i=1}^{n} \frac{(x_i - y_i)^2}{(x_i + y_i)}$$

TASK 3 :

Download the dataset

- Perform EDA
- Check the dataset is balance or not (using target variable "Label")
- Check whether there is any empty records, categorical feature, duplicate records, if yes then handle this and give a brief explanation why you have chosen this technique in a text cell or "jupyter/colab"
- Check the correlation of your dataset and perform feature selection using Pearson Correlation
- Analyze your dataset and think if feature scaling is required or not? If yes then apply any scaling technique based on your distribution.
- Split your dataset in training , testing and validation. Train split will be 80% and test will be 20% . In validation split your training samples will be 70% and validation set will be 30%. Briefly describe why we use validation set in a text cell. Declare Random_state=0
- Apply KNN and check model training and testing accuracy.
- Compare the accuracies by trying different metrics, combine all the training and testing accuracies of Euclidean, Manhattan etc. to compare their performance. Make a critical analysis what you have observed and where we have used different metrics?

TASK 5

- Apply KNN and check model training and testing accuracy.
- Compare the accuracies by trying different algorithms, combine all the training and testing accuracies of auto, KDTress , brute etc. to compare their performance. Do analyze what you have observed and where we have used different algorithms? What is the impact of changing algorithm on accuracy and testing time?