LABORATORY MANUAL

FOR

Laboratory Practice III(ML) (2019) (BE SEM I)

Department of Computer Engineering

International Institute of Information Technology

Hinjawadi,Pune - 411 057

www.isquareit.edu.in

Work Load	**Exam Schemes**		
Practical	**Term Work**	**Practical**	**Oral**
04 Hours/Week	50	50	--
List Of Assignments			
Sr. No.	**Title of Assignment**		**Time Span (No. of weeks)**

1)	Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks: 1. Pre-process the dataset. 2. Identify outliers. 3. Check the correlation. 4. Implement linear regression and random forest regression models. 5. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset	02
2)	Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method.Dataset link: https://www.kaggle.com/datasets/kyanyoga/sample-sales-data	01
3)	Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset. Dataset link: https://www.kaggle.com/datasets/abdallamahgoub/diabetes	01
4)	Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months. Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, Credit Score, Geography, Gender, Age, Tenure, Balance, etc. Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps: 1. Read the dataset. 2. Distinguish the feature and target set and divide the data set into training and test sets. 3. Normalize the train and test data. 4. Initialize and build the model. Identify the points of improvement and implement the same. 5. Print the accuracy score and confusion matrix (5 points).	01
5)	Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance. Dataset link: The emails.csv dataset on the Kaggle ttps://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv	02
6)	Mini Project Mini Project - Use the following dataset to analyze ups and downs in the market and predict future stock price returns based on Indian Market data from 2000 to 2020. Dataset Link: ttps://www.kaggle.com/datasets/sagara9595/stock-data	02
7)	Installation of MetaMask and study spending Ether per transaction.	01

8)	Create your own wallet using Metamask for crypto transactions.	01
9)	Write a smart contract on a test network, for Bank account of a customer for following operations: □ Deposit money □ Withdraw Money □ Show balance	02
10)	Write a program in solidity to create Student data. Use the following constructs: □ Structures □ Arrays □ Fallback Deploy this as smart contract on Ethereum and Observe the transaction fee and Gas values.	02
11)	Write a survey report on types of Blockchains and its real time use cases	02
12)	Mini Project - Develop a Blockchain based application for health related medical records	

Reference Sites:

https://builtin.com/machine-learning/how-to-preprocess-data-python

https://levelup.gitconnected.com/random-forest-regression

Virtual Laboratory:

1. http://cse01-iiith.vlabs.ac.in/
2. http://vlabs.iitb.ac.in/vlabs-dev/labs/blockchain/labs/index.php
3. http://vlabs.iitb.ac.in/vlabs-dev/labs/machine_learning/labs/index.php

ASSIGNMENT NO. 1

TITLE: PREDICT THE PRICE OF THE UBER RIDE FROM A GIVEN PICKUP POINT TO THE AGREED DROP-OFF LOCATION.

Problem Statement: Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks: 1. Pre-process the dataset. 2. Identify outliers. 3. Check the correlation. 4. Implement linear regression and random forest regression models. 5. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset.

Objective:

1. Pre-process the dataset
2. Identify outliers
3. Check the correlation
4. Implement linear regression and random forest regression models
5. Predict the price of the uber ride from a given pickup point to the agreed drop-off location
6. Evaluate the models and compare their respective scores like R2, RMSE

Theory:

What Is Data Preprocessing and Why Do We Need It?

For machine learning algorithms to work, it's necessary to convert raw data into a clean data set, which means we must convert the data set to numeric data. We do this by encoding all the categorical labels to column vectors with binary values. Missing values, or NaNs (not a number) in the data set is an annoying problem. You must either drop the missing rows or fill them up with a mean or interpolated values.

HOW TO PREPROCESS DATA IN PYTHON STEP-BY-STEP

1. Load data in Pandas.
2. Drop columns that aren't useful.
3. Drop rows with missing values.
4. Create dummy variables.
5. Take care of missing data.
6. Convert the data frame to NumPy.
7. Divide the data set into training data and test data.

Identify outliers

An Outlier is a data-item/object that deviates significantly from the rest of the (so-called normal) objects. They can be caused by measurement or execution errors. The analysis for outlier detection is referred to as outlier mining. There are many ways to detect the outliers, and the removal process is the data frame same as removing a data item from the panda's data frame.

Here pandas' data frame is used for a more realistic approach as in real-world project need to detect the outliers arouse during the data analysis step, the same approach can be used on lists and series-type objects.

Detecting the outliers

Outliers can be detected using visualization, implementing mathematical formulas on the dataset, or using the statistical approach.

1. Visualization a)Using Box Plot b)Using ScatterPlot.

2. Z-score

Z- Score is also called a standard score. This value/score helps to understand that how far is the data point from the mean. And after setting up a threshold value one can utilize z score values of data points to define the outlier
Zscore = (data_point -mean) / std. deviation

```
# Z score
from scipy import stats
import numpy as np

z = np.abs(stats.zscore(df_boston['DIS']))
print(z)
```

3. IQR (Inter Quartile Range)

IQR (Inter Quartile Range) Inter Quartile Range approach to finding the outliers is the most commonly used and most trusted approach used in the research field.
IQR = Quartile3 – Quartile1

What is Correlation?

Variables within a dataset can be related for lots of reasons.

For example:

- One variable could cause or depend on the values of another variable.
- One variable could be lightly associated with another variable.
- Two variables could depend on a third unknown variable.

It can be useful in data analysis and modeling to better understand the relationships between variables.

The statistical relationship between two variables is referred to as their correlation.

- **Positive Correlation**: both variables change in the same direction.
- **Neutral Correlation**: No relationship in the change of the variables.
- **Negative Correlation**: variables change in opposite directions.

The performance of some algorithms can deteriorate if two or more variables are tightly related, called multicollinearity. An example is linear regression, where one of the offending correlated variables should be removed in order to improve the skill of the model.

Covariance

Variables can be related by a linear relationship. This is a relationship that is consistently additive across the two data samples.
This relationship can be summarized between two variables, called the covariance. It is calculated as the average of the product between the values from each sample, where the values haven been centered (had their mean subtracted).
The calculation of the sample covariance is as follows:
cov(X, Y) = (sum (x - mean(X)) * (y - mean(Y))) * 1/(n-1)
The use of the mean in the calculation suggests the need for each data sample to have a Gaussian or Gaussian-like distribution.
The sign of the covariance can be interpreted as whether the two variables change in the same direction (positive) or change in different directions (negative). The magnitude of the covariance is not easily interpreted. A covariance value of zero indicates that both variables are completely independent.
The *cov()* NumPy function can be used to calculate a covariance matrix between two or more variables.
There are several statistics that you can use to quantify correlation. Three correlation coefficients are :

- Pearson's r
- Spearman's rho
- Kendall's tau

Pearson's coefficient measures linear correlation, while the Spearman and Kendall coefficients compare the ranks of data. There are several NumPy, SciPy, and Pandas correlation functions and methods that you can use to calculate these coefficients.

Pearson's Correlation

The Pearson correlation coefficient (named for Karl Pearson) can be used to summarize the strength of the linear relationship between two data samples.
The Pearson's correlation coefficient is calculated as the covariance of the two variables divided by the product of the standard deviation of each data sample. It is the normalization of the covariance between the two variables to give an interpretable score.
Pearson's correlation coefficient = covariance(X, Y) / (stdv(X) * stdv(Y))

Pandas Correlation Calculation

Pandas is, in some cases, more convenient than NumPy and SciPy for calculating statistics. It offers statistical methods for Series and DataFrame instances. For example, given two Series objects with the same number of items, you can call **.corr()** on one of them with the other as the first argument:

3. Implement linear regression and random forest regression models

What is Regression?

Regression analysis is a statistical method that helps us to understand the relationship between dependent and one or more independent variables,

Dependent Variable

This is the Main Factor that we are trying to predict.

Independent Variable

These are the variables that have a relationship with the dependent variable.

Types of Regression Analysis

There are many types of regression analysis, but in this article, we will deal with,

1. Simple Linear Regression
2. Multiple Linear Regression

What is Linear Regression?

In Machine Learning lingo, Linear Regression (LR) means simply finding the best fitting line that explains the variability between the dependent and independent features very well or we can say it describes the linear relationship between independent and dependent features, and in linear regression, the algorithm predicts the continuous features (e.g. Salary, Price), rather than deal with the categorical features (e.g. cat, dog).

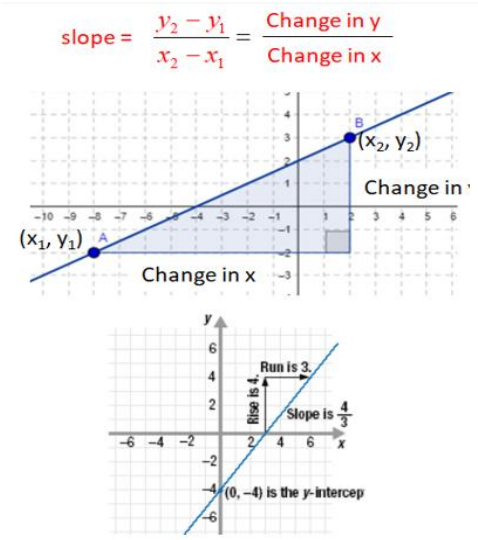
Simple Linear Regression

Simple Linear Regression uses the slope-intercept (weight-bias) form, where our model needs to find the optimal value for both slope and intercept. So with the optimal values, the model can find the variability between the independent and dependent features and produce accurate results. In simple linear regression, the model takes a single independent and dependent variable.

There are many equations to represent a straight line, we will stick with the common equation,

$$y = b_o + b_1 x$$

Here, y and x are the dependent variables, and independent variables respectively. b1(m) and b0(c) are slope and y-Intercept.



Slope(m) tells, for one unit of increase in *x*, How many units does it increase in *y*. When the line is steep, the slope will be higher, the slope will be lower for the less steep line.

Constant(c) means, what is the value of y when the x is zero.

How the Model will Select the Best Fit Line?

First, our model will try a bunch of different straight lines from that it finds the optimal line that predicts our data points well.

From the above picture, you can notice there are 4 lines, and any guess which will be our best fit line?

Ok, For finding the best fit line our model uses the cost function. In machine learning, every algorithm has a cost function, and in simple linear regression, the goal of our algorithm is to find a minimal value for the cost function.
And in linear regression (LR), we have many cost functions, but mostly used cost function is MSE(Mean Squared Error). It is also known as a Least Squared Method.

$$\mathbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Yi – Actual value,
Y^i – Predicted value,
n – number of records.
(yi – yi_hat) is a Loss Function. And you can find in most times people will interchangeably use the word loss and cost function. But they are different, and we are squaring the terms to neglect the negative value.

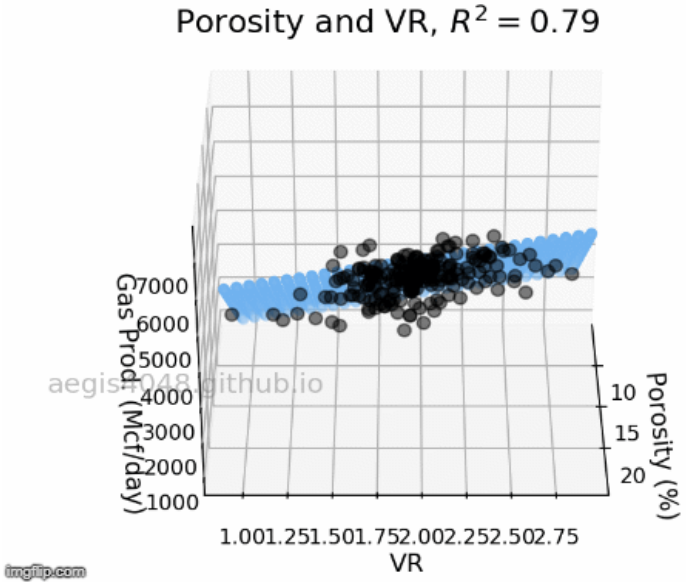
Loss Function
It is a calculation of loss for single training data.
Cost Function
It is a calculation of average loss over the entire dataset.

Multiple Linear Regression

In multiple linear regression instead of having a single independent variable, the model has multiple independent variables to predict the dependent variable.

$$y = b_o + b_1x_1 + b_2x_2 + b_3x_3 \ldots + b_nx_n$$

here bo is the y-intercept, $b_1,b_2,b_3,b_4\ldots,b_n$ are slopes of the independent variables $x_1,x_2,x_3,x_4\ldots,x_n$ and y is the dependent variable.
Here instead of finding a line, our model will find the best plane in 3-Dimension, and in n-Dimension, our model will find the best hyperplane. The below diagram is for demonstration purposes.



Python code for linear regression

After splitting the data into training and testing sets, finally, the time is to train our algorithm. For that, we need to import LinearRegression class, instantiate it, and call the fit() method along with our training data.

```
regressor = LinearRegression()
regressor.fit(X_train, y_train) #training the algorithm
```

he linear regression model basically finds the best value for the intercept and slope, which results in a line that best fits the data. To see the value of the intercept and slop calculated by the linear regression algorithm for our dataset, execute the following code.

```
#To retrieve the intercept:
print(regressor.intercept_)

#For retrieving the slope:
print(regressor.coef_)
```

To make predictions on the test data, execute the following script:

```
y_pred = regressor.predict(X_test)
```

Let's plot our straight line with the test data:

```
plt.scatter(X_test, y_test,  color='gray')
plt.plot(X_test, y_pred, color='red', linewidth=2)
plt.show()
```

The final step is to evaluate the performance of the algorithm. This step is particularly important to compare how well different algorithms perform on a particular dataset. For regression algorithms, three evaluation metrics are commonly used:

1. Mean Absolute Error (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$\mathbf{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - y_j|$$

2. Mean Squared Error (MSE) is the mean of the squared errors and is calculated as:

$$MSE = \frac{1}{N}\sum_{i}^{n}(Y_i - y_i)^2$$

3. Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

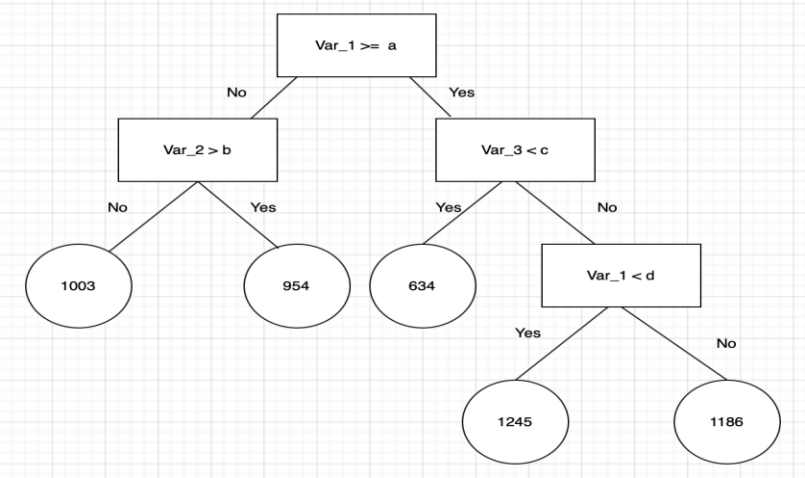
we don't have to perform these calculations manually. The Scikit-Learn library comes with pre-built functions that can be used to find out these values for us.

Let's find the values for these metrics using our test data.

```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

Random Forest Regression Models

Decision Trees are used for both regression and classification problems. They visually flow like trees, hence the name, and in the regression case, they start with the root of the tree and follow splits based on variable outcomes until a leaf node is reached and the result is given. An example of a decision tree is below



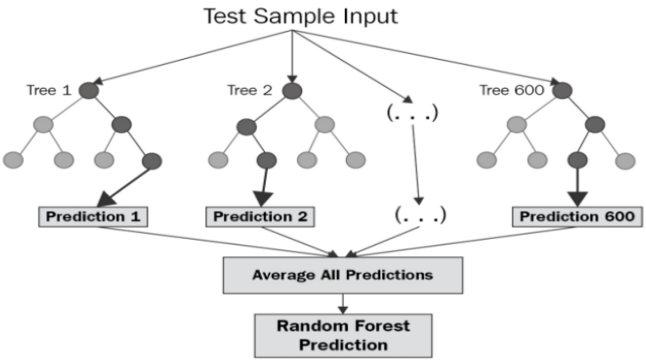
Here we see a basic decision tree diagram which starts with the Var_1 and splits based off of specific criteria. When 'yes', the decision tree follows the represented path, when 'no', the decision tree goes down the other path. This process repeats until the decision tree reaches the leaf node and the resulting outcome is decided. For the example above, the values of a, b, c, or d could be representative of any numeric or categorical value.

Ensemble learning is the process of using multiple models, trained over the same data, averaging the results of each model ultimately finding a more powerful predictive/classification result. Our hope, and the requirement, for ensemble learning is that the errors of each model (in this case decision tree) are independent and different from tree to tree.

Bootstrapping is the process of randomly sampling subsets of a dataset over a given number of iterations and a given number of variables. These results are then averaged together to obtain a more powerful result. Bootstrapping is an example of an applied ensemble model.

The bootstrapping **Random Forest** algorithm combines ensemble learning methods with the decision tree framework to create multiple randomly drawn decision trees from the data, averaging the results to output a new result that often leads to strong predictions/classifications.

Random Forest Regression is a supervised learning algorithm that uses **ensemble learning** method for regression. Ensemble learning method is a technique that combines predictions from multiple machine learning algorithms to make a more accurate prediction than a single model.



The diagram above shows the structure of a Random Forest. You can notice that the trees run in parallel with no interaction amongst them. A Random Forest operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees. To get a better understanding of the Random Forest algorithm, let's walk through the steps:

1. Pick at random *k* data points from the training set.
2. Build a decision tree associated to these *k* data points.
3. Choose the number *N* of trees you want to build and repeat steps 1 and 2.
4. For a new data point, make each one of your *N*-tree trees predict the value of *y* for the data point in question and assign the new data point to the average across all of the predicted *y* values.

A Random Forest Regression model is powerful and accurate. It usually performs great on many problems, including features with non-linear relationships. Disadvantages, however, include the following: there is no interpretability, overfitting may easily occur, we must choose the number of trees to include in the model.

Random Forest Regression Model:

We will use the sklearn module for training our random forest regression model, specifically the RandomForestRegressor function. The RandomForestRegressor documentation shows many different parameters we can select for our model. Some of the important parameters are highlighted below:

- **n_estimators —** the number of decision trees you will be running in the model
- **criterion —** this variable allows you to select the criterion (loss function) used to determine model outcomes. We can select from loss functions such as mean squared error (MSE) and mean absolute error (MAE). The default value is MSE.
- **max_depth —** this sets the maximum possible depth of each tree
- **max_features —** the maximum number of features the model will consider when determining a split
- **bootstrap —** the default value for this is True, meaning the model follows bootstrapping principles (defined earlier)
- **max_samples —** This parameter assumes bootstrapping is set to True, if not, this parameter doesn't apply. In the case of True, this value sets the largest size of each sample for each tree.
- Other important parameters are **min_samples_split, min_samples_leaf, n_jobs**, and others that can be read in the sklearn's.
- Let's see Random Forest Regression in action!
-

After importing the libraries, importing the dataset, addressing null values, and dropping any necessary columns, we are ready to create our Random Forest Regression model!

Step 1: Identify your dependent (y) and independent variables (X)

Step 2: Split the dataset into the Training set and Test set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                    random_state = 0)
```

Step 3: Training the Random Forest Regression model on the whole dataset

```
rf = RandomForestRegressor(n_estimators = 300, max_features = 'sqrt', max_depth = 5, random_state = 18).fit(x_train, y_train)
```

Looking at our base model above, we are using 300 trees; max_features per tree is equal to the squared root of the number of parameters in our training dataset. The max depth of each tree is set to 5. And lastly, the random_state was set to 18 just to keep everything standard.

Step 4: Predicting the Test set results

```
y_pred = regressor.predict(X_test)
y_pred
```

when we solve classification problems, we can view our performance using metrics such as accuracy, precision, recall, etc. When viewing the performance metrics of a regression model, we can use factors such as mean squared error, root mean squared error, R^2, adjusted R^2, and others.

```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

R^2 score tells us how well our model is fitted to the data by comparing it to the average line of the dependent variable. If the score is closer to 1, then it indicates that our model performs well versus if the score is farther from 1, then it indicates that our model does not perform so well.

Conclusion: We studied how to Predict the price of the Uber ride from a given pickup point to the agreed drop-off location using linear regression and random forest regression models. Also Evaluated the models and compare their respective scores like R2, RMSE

ASSIGNMENT NO.2

TITLE: IMPLEMENTATION OF K-MEANS CLUSTERING/ HIERARCHICAL CLUSTERING ON SALES_DATA.

Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method. Dataset link:
https://www.kaggle.com/datasets/kyanyoga/sample-sales-data

Objective:

To understand and implement:

1. K-Means clustering
2. Determine the number of clusters using the elbow method

Theory:

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

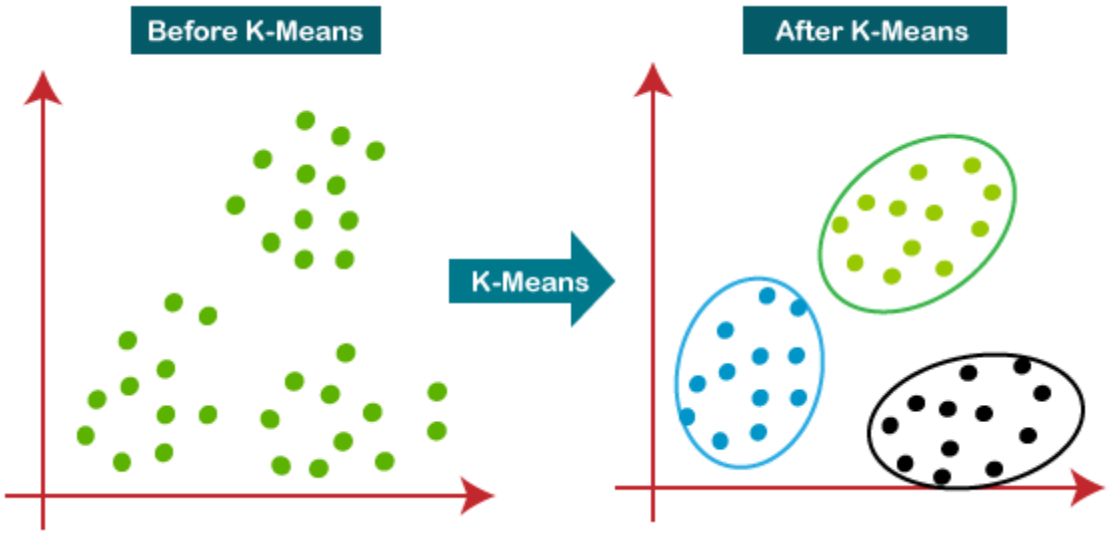
It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.
The below diagram explains the working of the K-means Clustering Algorithm:



Algorithms:

How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS= \sum_{Pi in Cluster1} distance(P_i C_1)^2 + \sum_{Pi in Cluster2} distance(P_i C_2)^2 + \sum_{Pi in CLuster3} distance(P_i C_3)^2$$

In the above formula of WCSS,

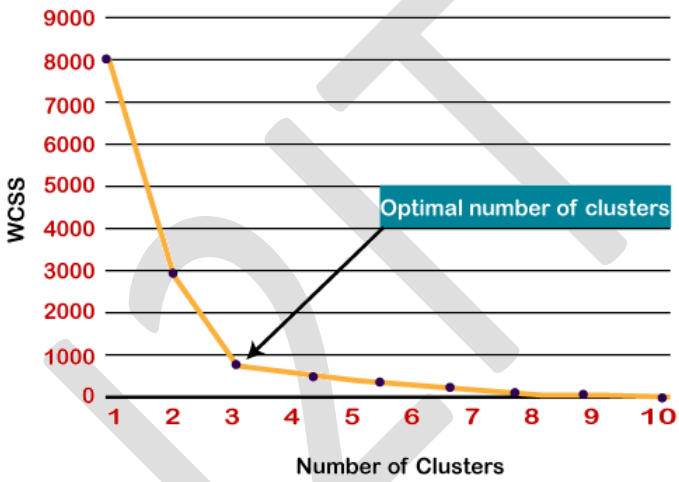
∑Pi in Cluster1 distance(Pi C1)2: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Applications of K-means Clustering

The concern of the fact is that the data is always complicated, mismanaged, and noisy. Let's learn where we can implement k-means clustering among various

1. K-means clustering is applied in the **Call Detail Record (CDR) Analysis**. It gives in-depth vision about customer requirements and satisfaction based on call-traffic during the time of the day and demographic of a particular location.
2. It is used in **the clustering of documents** to identify the compatible documents in the same place.

3. It is deployed **to classify the sounds** based on their identical patterns and segregate malformation in them.
4. It serves as the **model of lossy images compression technique**, in the confinement of images, K-means makes clusters pixels of an image in order to decrease the total size of it.
5. It is helpful in the business sector for recognizing the portions of purchases made by customers, also to cluster movements on apps and websites.
6. In the field of insurance and fraud detection on the basis of prior data, it is plausible to cluster fraudulent consumers to demand based on their proximity to clusters as the patterns indicate.

K-means vs Hierarchical Clustering

1. K-means clustering **produces a specific number of clusters** for the disarranged and flat dataset, where Hierarchical clustering **builds a hierarchy of clusters**, not for just a partition of objects under various clustering methods and applications.
2. K-means can be used for categorical data and first converted into numeric by assigning rank, where Hierarchical clustering was selected for categorical data but due to its complexity, a new technique is considered to assign rank value to categorical features.
3. K-means are **highly sensitive to noise** in the dataset and perform well than Hierarchical clustering where it is **less sensitive to noise** in a dataset.
4. ***Performance of the K-Means algorithm increases as the RMSE decreases and the RMSE decreases as the number of clusters increases*** so the time of execution increases, in contrast to this, the performance of Hierarchical clustering is less.
5. K-means are **good for a large dataset** and Hierarchical clustering is **good for small datasets**.

Conclusion: Thus, we have understood what Clustering is, K-mean clustering. We also Determine the number of clusters using the elbow method and Implemented K-mean clustering.

ASSIGNMENT NO.3

TITLE: IMPLEMENTATION OF K-NEAREST NEIGHBORS ALGORITHM.

Problem Statement: Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset. Dataset link: https://www.kaggle.com/datasets/abdallamahgoub/diabetes

Objective:

To understand and implement:

1. K-Nearest Neighbors.
2. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

Theory:

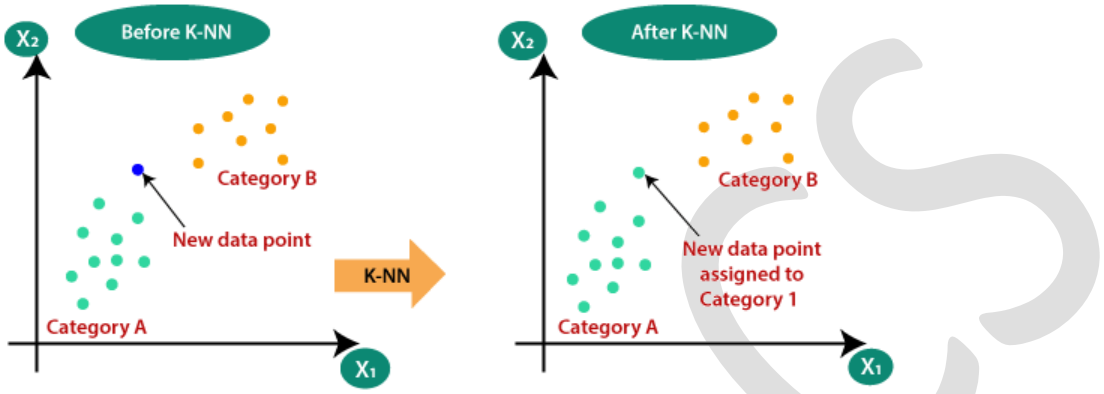
What is K-Nearest Neighbors Algorithm?

K-Nearest Neighbour (KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most like the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much like the new data.

Why do we need a K-NN Algorithm?

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



How does K-NN work?

The K-NN working can be explained based on the below algorithm:

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

How to select the value of K in the K-NN Algorithm?

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Steps to implement the K-NN algorithm:

- Data Pre-processing step
- Fitting the K-NN algorithm to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)
- Visualizing the test set result.

Python Code for K-NN

Fitting K-NN classifier to the Training data:

Now we will fit the K-NN classifier to the training data. To do this we will import the KNeighborsClassifier class of Sklearn Neighbors library. After importing the class, we will create the Classifier object of the class. The Parameter of this class will be

- n_neighbors: To define the required neighbors of the algorithm. Usually, it takes 5.
- metric='minkowski': This is the default parameter and it decides the distance between the points.
- p=2: It is equivalent to the standard Euclidean metric.

#Fitting K-NN classifier to the training set

```
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

Predicting the Test Result:

To predict the test set result, we will create a y_pred vector. Below is the code for it:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

Creating the Confusion Matrix:

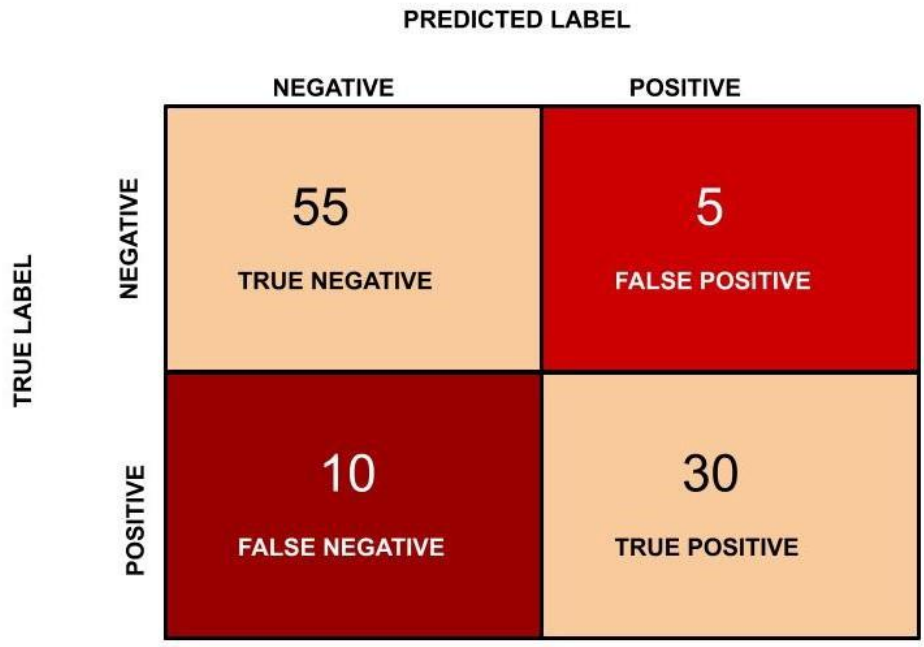
Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier. Below is the code for it:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

What is Confusion Matrix and why you need it?

Confusion Matrix is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values. In this case, TN = 55, FP = 5, FN = 10, TP = 30. The confusion matrix is as follows.



It is extremely useful for measuring Recall, Precision, Specificity, Accuracy, and most importantly AUC-ROC curves.

Let's understand TP, FP, FN, TN

True Positive:
Interpretation: You predicted positive and it's true.
You predicted that a Man is terrorist, and he actually is.

True Negative:
Interpretation: You predicted negative and it's true.
You predicted that a man is not terrorist, and he actually is not.

False Positive: (Type 1 Error)
Interpretation: You predicted positive and it's false.
You predicted that a man is terrorist, but he actually is not.

False Negative: (Type 2 Error)
Interpretation: You predicted negative and it's false.
You predicted that a man is not terrorist, but he actually is.

$$TPR = \frac{TP}{Actual\ Positive} = \frac{TP}{TP + FN}$$

$$FNR = \frac{FN}{Actual\ Positive} = \frac{FN}{TP + FN}$$

$$TNR = \frac{TN}{Actual\ Negative} = \frac{TN}{TN + FP}$$

$$FPR = \frac{FP}{Actual\ Negative} = \frac{FP}{TN + FP}$$

Accuracy is the given as

What is the **accuracy** of the machine learning model for this classification task?

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN}$$

Accuracy represents the number of correctly classified data instances over the total number of data instances.
In this example, Accuracy = (55 + 30)/(55 + 5 + 30 + 10) = 0.85 and in percentage the accuracy will be 85%.

Is accuracy the best measure?

Accuracy may not be a good measure if the dataset is not balanced (both negative and positive classes have different number of data instances).
Even if data is imbalanced, we can figure out that our model is working well or not. For that, the values of TPR and TNR should be high, and FPR and FNR should be as low as possible.
With the help of TP, TN, FN, and FP, other performance metrics can be calculated.

Precision, Recall

Both precision and recall are crucial for information retrieval, where positive class mattered the most as compared to negative. **Why?**
While searching something on the web, the model does not care about something **irrelevant** and **not retrieved** (this is the true negative case). Therefore, only TP, FP, FN are used in Precision and Recall.

Precision

What does precision mean?

Precision should ideally be 1 (high) for a good classifier. *Precision* becomes 1 only when the numerator and denominator are equal i.e *TP* = *TP* +*FP*, this also means *FP* is zero. As *FP* increases the value of denominator becomes greater than the numerator and *precision* value decreases (which we don't want).
So in the TERRORIST example,
precision = 30/(30+ 5) = 0.857

$$Precision = \frac{TP}{TP + FP}$$

$$precision = \frac{true\ positives}{true\ positives + false\ positives} = \frac{terrorists\ correctly\ identified}{terrorists\ correctly\ identified + individuals\ incorrectly\ labeled\ as\ terroists}$$

Recall

Recall is also known as sensitivity or true positive rate and is defined as follows:

$$Recall = \frac{TP}{TP + FN}$$

$$recall = \frac{true\ positives}{true\ positives\ + false\ negatives} = \frac{terrorists\ correctly\ identified}{terrorists\ correctly\ identified + terrorists\ incorrectly\ labeled\ as\ not\ terrorists}$$

Recall should ideally be 1 (high) for a good classifier. Recall becomes 1 only when the numerator and denominator are equal i.e TP = TP +FN, this also means FN is zero. As FN increases the value of denominator becomes greater than the numerator and recall value decreases (which we don't want).
So in the TERRORIST example let us see what will be the recall.

Recall = 30/(30+ 10) = 0.75

Conclusion: Thus we Implemented K-Nearest Neighbors algorithm on given data set and to evaluate the performance we Computed confusion matrix, accuracy, error rate, precision and recall on the given dataset.

ASSIGNMENT NO.4

TITLE: BUILD A NEURAL NETWORK-BASED CLASSIFIER FOR CUSTOMER CHURN.

Problem Statement: Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.

Dataset Description: The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as Customer Id, Credit Score, Geography, Gender, Age, Tenure, Balance, etc. Link to the Kaggle project: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling Perform following steps: 1. Read the dataset. 2. Distinguish the feature and target set and divide the data set into training and test sets. 3. Normalize the train and test data. 4. Initialize and build the model. Identify the points of improvement and implement the same. 5. Print the accuracy score and confusion matrix (5 points).

Objective:

1 To build a neural network-based classifier
2. Compute confusion matrix, accuracy score on the given dataset.

Theory:

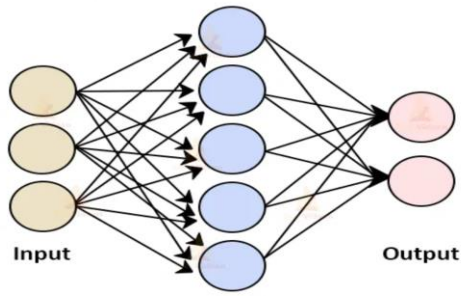
What is Neural Network?

Neural Network is a series of algorithms that are trying to mimic the human brain and find the relationship between the sets of data. It is being used in various use-cases like in regression, classification, Image Recognition and many more.

As we have talked above that neural networks tries to mimic the human brain then there might be the difference as well as the similarity between them. Let us talk in brief about it.

Some major differences between them are biological neural network does parallel processing whereas the Artificial neural network does series processing also in the former one processing is slower (in millisecond) while in the latter one processing is faster (in a nanosecond).

Architecture Of ANN

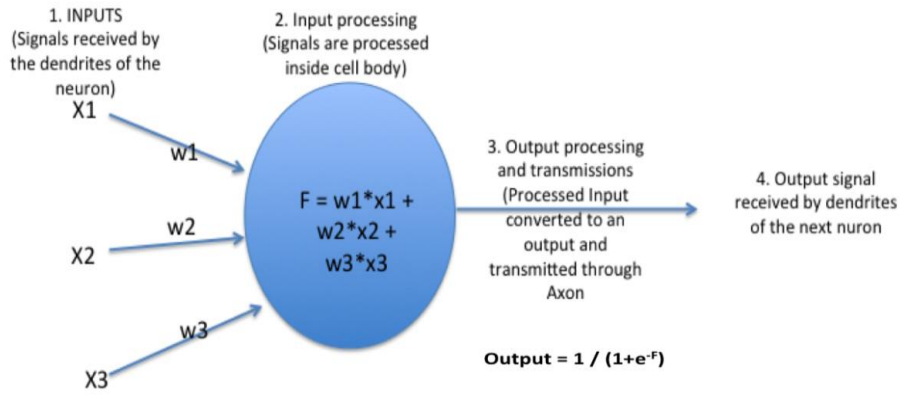


A neural network has many layers and each layer performs a specific function, and as the complexity of the model increases, the number of layers also increases that why it is known as the multi-layer perceptron.

The purest form of a neural network has three layers input layer, the hidden layer, and the output layer. The input layer picks up the input signals and transfers them to the next layer and finally, the output layer gives the final prediction and these neural networks have to be trained with some training data as well like machine learning algorithms before providing a particular problem. Now, let's understand more about perceptron.

Perceptron

As discussed above multi-layered perceptron these are basically the hidden or the dense layers. They are made up of many neurons and neurons are the primary unit that works together to form perceptron. In simple words, as you can see in the above picture each circle represents neurons and a vertical combination of neurons represents perceptrons which is basically a dense layer.



Now in the above picture, you can see each neuron's detailed view. Here, each neurons have some weights (in above picture w1, w2, w3) and biases and based on this computations are done as, combination = bias + weights * input (F = w1*x1 + w2*x2 + w3*x3) and finally activation function is applied output = activation(combination) in above picture activation is sigmoid represented by 1/(1 + e-F). There are some other activation functions as well like ReLU, Leaky ReLU, tanh, and many more.

Working Of ANN

At First, information is feed into the input layer which then transfers it to the hidden layers, and interconnection between these two layers assign weights to each input randomly at the initial point. and then bias is added to each input neuron and after this, the weighted sum which is a combination of weights and bias is passed through the activation function. Activation Function has the responsibility of which node to fire for feature extraction and finally output is calculated. This whole process is known as Forward Propagation. After getting the output model to compare it with the original output and the error is known and finally, weights are updated in backward propagation to reduce the error and this process

continues for a certain number of epochs (iteration). Finally, model weights get updated, and prediction is done.

Advantages

1. ANN has the ability to learn and model non-linear and complex relationships as many relationships between input and output are non-linear.
2. After training, ANN can infer unseen relationships from unseen data, and hence it is generalized.
3. Unlike many machines learning models, ANN does not have restrictions on datasets like data should be Gaussian distributed or nay other distribution.

Applications

There are many applications of ANN. Some of them are:

1. Image Preprocessing and Character Recognition.
2. Forecasting.
3. Credit rating.
4. Fraud Detection.
5. Portfolio Management

Python code for ANN

1. Understanding and Loading the datasets

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
```

2. Defining the Keras Model

```
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Models in Keras are defined as a sequence of layers in which each layer is added one after another. The input should contain input features and is specified when creating the first layer with the **input_dims** argument. Here **inputs_dims will be 8.**

A fully connected network with a three-layer is used which is defined using the Dense Class. The first argument takes the number of neurons in that layer and, and the activation argument takes the activation function as an input. Here ReLU is used as an activation function in the first two layers and sigmoid in the last layer as it is a binary classification problem.

3. Compile Keras Model

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

While compiling we must specify the loss function to calculate the errors, the optimizer for updating the weights and any metrics.
In this case, we will use "binary_crossentropy" as the loss argument as it is a binary classification problem.
Here we will take optimizer as "adam"as it automatically tunes itself and gives good results in a wide range of problems and finally we will collect and report the classification accuracy
through metrics argument.

4. Fitting the Keras Model.

```
model.fit(X, y, epochs=150, batch_size=10)
```

Now we will fit our model on the loaded data by calling the fit() function on the model.
The training process will run for a fixed number of iterations through the dataset which is specified using the epochs argument. The number of dataset rows should be and are updated within each epoch, and set using the batch_size argument.
Here, We will run for 150 epochs and a batch size of 10.

5. Evaluate Keras Model

```
_, accuracy = model.evaluate(X, y)
print('Accuracy: %.2f' % (accuracy*100))
```

The evaluation of the model on the dataset can be done using the evaluate() function. It takes two arguments i.e, input and output. It will generate a prediction for each input and output pair and collect scores, including the average loss and any metrics such as accuracy.
The evaluate() function will return a list with two values first one is the loss of the model and the second will be the accuracy of the model on the dataset. We are only interested in reporting the accuracy and hence we ignored the loss value.

6. Make Predictions

```
predictions = model.predict(X)
rounded = [round(x[0]) for x in predictions]
```

Prediction can be done by calling the predict () function on the model. Here sigmoid activation function is used on the output layer, so the predictions will be a probability in the range between 0 and 1.

Performance evaluation is done using Confusion metrics which is covered in previous assignments.

Conclusion: we have learned how to build a neural network-based classifier that can determine whether bank customer will leave or not in the next 6 months.

ASSIGNMENT NO.5

TITLE: CLASSIFY THE EMAIL USING THE BINARY CLASSIFICATION METHOD

Problem Statement: Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance. Dataset link: The emails.csv dataset on the Kaggle ttps://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

Objective:

1. To read the dataset and classify Email is Spam or Not Spam using K-NN
2. To read the dataset and classify Email is Spam or Not Spam using SVM
3. Compute confusion matrix, accuracy score on the given dataset.

Theory:

What is SVM?

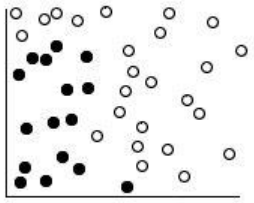
Support Vector Machine (SVM) is a robust classification and regression technique that maximizes the predictive accuracy of a model without overfitting the training data. SVM is particularly suited to analyzing data with very large numbers (for example, thousands) of predictor fields.

SVM has applications in many disciplines, including customer relationship management (CRM), facial and other image recognition, bioinformatics, text mining concept extraction, intrusion detection, protein structure prediction, and voice and speech recognition.

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data are transformed in such a way that the separator could be drawn as a hyperplane. Following this, characteristics of new data can be used to predict the group to which a new record should belong.

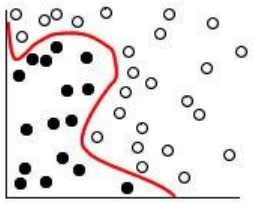
For example, consider the following figure, in which the data points fall into two different categories.

Figure 1. Original dataset



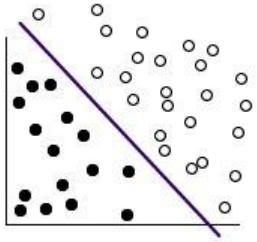
The two categories can be separated with a curve, as shown in the following figure.

Figure 2. Data with separator added



After the transformation, the boundary between the two categories can be defined by a hyperplane, as shown in the following figure.

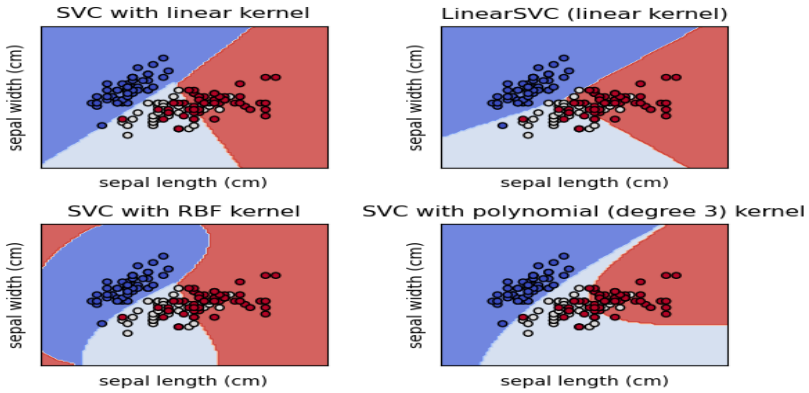
Figure 3. Transformed data



The mathematical function used for the transformation is known as the kernel function. SVM in Modeler supports the following kernel types:

- Linear
- Polynomial
- Radial basis function (RBF)
- Sigmoid

A linear kernel function is recommended when linear separation of the data is straightforward. In other cases, one of the other functions should be used. You will need to experiment with the different functions to obtain the best model in each case, as they each use different algorithms and parameters. Below figure shows how kernel functions on data



Tuning Hyperparameters

- **Kernel**: The main function of the kernel is to transform the given dataset input data into the required form. There are various types of functions such as linear, polynomial, and radial basis function (RBF). Polynomial and RBF are useful for non-linear hyperplane. Polynomial and RBF kernels compute the separation line in the higher dimension. In some of the applications, it is suggested to use a more complex kernel to separate the classes that are curved or nonlinear. This transformation can lead to more accurate classifiers.

- **Regularization**: Regularization parameter in python's Scikit-learn C parameter used to maintain regularization. Here C is the penalty parameter, which represents misclassification or error term. The misclassification or error term tells the SVM optimization how much error is bearable. This is how you can control the trade-off between decision boundary and misclassification term. A smaller value of C creates a small-margin hyperplane and a larger value of C creates a larger-margin hyperplane.

- **Gamma**: A lower value of Gamma will loosely fit the training dataset, whereas a higher value of gamma will exactly fit the training dataset, which causes over-fitting. In other words, you can say a low value of gamma considers only nearby points in calculating the separation line, while the a value of gamma considers all the data points in the calculation of the separation line.

Advantages of SVM:

- Effective in high dimensional cases
- Its memory efficient as it uses a subset of training points in the decision function called support vectors
- Different kernel functions can be specified for the decision functions and its possible to specify custom kernels
- SVM Classifiers offer good accuracy and perform faster prediction compared to Naïve Bayes algorithm.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.

- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

- SVM is not suitable for large datasets because of its high training time and it also takes more time in training compared to Naïve Bayes. It works poorly with overlapping classes and is also sensitive to the type of kernel used.

SVM Kernel:
The SVM kernel is a function that takes low dimensional input space and transforms it into higher-dimensional space, ie it converts not separable problem to separable problem. It is mostly useful in non-linear separation problems. Simply put the kernel, it does some extremely complex data transformations then finds out the process to separate the data based on the labels or outputs defined.

SVM implementation in python:

Objective: Predict if cancer is beningn or malignant.

Using historical data about patients diagnosed with cancer, enable the doctors to differentiate malignant cases and benign given the independent attributes.

```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
# Importing Data file
data = pd.read_csv('bc2.csv')
dataset = pd.DataFrame(data)
dataset.columns
dataset = dataset.replace('?', np.nan)
dataset = dataset.apply(lambda x: x.fillna(x.median()),axis=0)

# converting the hp column from object 'Bare Nuclei'/ string type to float
dataset['Bare Nuclei'] = dataset['Bare Nuclei'].astype('float64')
dataset.isnull().sum()
from sklearn.model_selection import train_test_split

# To calculate the accuracy score of the model
from sklearn.metrics import accuracy_score, confusion_matrix

target = dataset["Class"]
features = dataset.drop(["ID","Class"], axis=1)
X_train, X_test, y_train, y_test = train_test_split(features,target, test_size = 0.2, random_state = 10)
```

```
from sklearn.svm import SVC

# Building a Support Vector Machine on train data
svc_model = SVC(C= .1, kernel='linear', gamma= 1)
svc_model.fit(X_train, y_train)
prediction = svc_model .predict(X_test)
# check the accuracy on the training set
print(svc_model.score(X_train, y_train))
print(svc_model.score(X_test, y_test))
```

Output:

0.9749552772808586

0.9642857142857143

```
print("Confusion Matrix:\n",confusion_matrix(prediction,y_test))
```

Confusion Matrix:

[[95 2]

[3 40]]

```
# Building a Support Vector Machine on train data
svc_model = SVC(kernel='rbf')
svc_model.fit(X_train, y_train)
print(svc_model.score(X_train, y_train))
print(svc_model.score(X_test, y_test))
```

Output:

0.998211091234347

0.9571428571428572

```
#Building a Support Vector Machine on train data (changing the kernel)
svc_model  = SVC(kernel='poly')
svc_model.fit(X_train, y_train)

prediction = svc_model.predict(X_test)

print(svc_model.score(X_train, y_train))
print(svc_model.score(X_test, y_test))
```

Output:

1.0

0.9357142857142857

```
svc_model = SVC(kernel='sigmoid')
svc_model.fit(X_train, y_train)
```

```
prediction = svc_model.predict(X_test)

print(svc_model.score(X_train, y_train))
print(svc_model.score(X_test, y_test))
```

Output:
0.3434704830053667

0.32857142857142857

K-NN code for Email Classification

```
df=pd.read_csv('/content/emails.csv',error_bad_lines=False)
df.dropna(inplace = True)
df.drop(['Email No.'],axis=1,inplace=True)
X = df.drop(['Prediction'],axis = 1)
y = df['Prediction']
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Prediction",y_pred)
print("KNN accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

KNN accuracy = 0.8009020618556701

```
print("Confusion matrix",metrics.confusion_matrix(y_test,y_pred))
```

Confusion matrix
[[804 293]
[16 439]]

SVM code Email Classification

```
# cost C = 1
model = SVC(C = 1)

# fit
model.fit(X_train, y_train)

# predict
y_pred = model.predict(X_test)
print("SVM accuracy = ",metrics.accuracy_score(y_test,y_pred))
```

SVM accuracy = 0.9381443298969072

```
metrics.confusion_matrix(y_true=y_test, y_pred=y_pred)
array(

[[1091, 6], '

[ 90, 365]])
```

Conclusion: We understood what SVM is, different kernels used for hyper parameter tuning.
We also implemented K-NN and SVM for Email classification and found that SVM gives good result compared to K-NN for the given data set.