



Marathwada Mitra Mandal's  
**Marathwada Mitramandal Institute of  
Techology, Lohegaon, Pune.**



## **LABORATORY MANUAL**

### **SUBJECT: Laboratory Practice - III**

**[SUBJECT CODE: 410246]**  
**Group B: Machine Learning**

**CLASS: B.E. Computer Engg.**  
**YEAR: 2022-23**

**PREPARED BY:**

**Mr. Tushar Waykole**  
**SUBJECT TEACHER**

**APPROVED BY:**

**Prof. Subhash G. Rathod**  
**H.O.D. [Comp]**

**Dr. Rupesh Bhortake**  
**PRINCIPAL**

**DEPARTMENT OF COMPUTER ENGINEERING– MMIT (2022-23)**

## List of Experiments

Sr. No.	Title	Page No.
<b>According to SPPU Curriculum</b>		
<b>410242: Machine Learning</b>		
1	<p>Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:</p> <ol style="list-style-type: none"> <li>1. Pre-process the dataset.</li> <li>2. Identify outliers.</li> <li>3. Check the correlation.</li> <li>4. Implement <b>linear regression</b> and <b>random forest regression</b> models.</li> <li>5. Evaluate the models and compare their respective scores like R2, RMSE, etc.</li> </ol> <ul style="list-style-type: none"> <li>• Dataset link: <a href="https://www.kaggle.com/datasets/yasserh/uber-fares-dataset">https://www.kaggle.com/datasets/yasserh/uber-fares-dataset</a></li> </ul>	3
2	<p>Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use <b>K-Nearest Neighbors and Support Vector Machine</b> for classification. Analyze their performance.</p> <ul style="list-style-type: none"> <li>• Dataset link: The emails.csv dataset on the Kaggle <a href="https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv">https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv</a></li> </ul>	12
3	<p>Implement <b>Gradient Descent Algorithm</b> to find the local minima of a function. For example, find the local minima of the function <math>y=(x+3)^2</math> starting from the point <math>x=2</math>.</p>	23
4	<p>Implement <b>K-Nearest Neighbors</b> algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.</p> <ul style="list-style-type: none"> <li>• Dataset link: <a href="https://www.kaggle.com/datasets/abdallamahgoub/diabetes">https://www.kaggle.com/datasets/abdallamahgoub/diabetes</a></li> </ul>	30
5	<p>Implement <b>K-Means clustering/ hierarchical clustering</b> on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method. Dataset link : <a href="https://www.kaggle.com/datasets/kyanyoga/sample-sales-data">https://www.kaggle.com/datasets/kyanyoga/sample-sales-data</a></p>	36

## Experiment No: 1

**Aim** Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.

### Objectives:

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.

### Software Used:

Linux/Unix/Ubuntu Operating Systems, Jupyter Notebook

Programming Language: Python 3.7.8

Dataset link: <https://www.kaggle.com/datasets/yasserh/uber-fares-dataset>

### Theory:

#### 1. Linear Regression

If you want to start machine learning, linear regression is the best place to start. Linear Regression is a regression model, meaning, it'll take features and predict a continuous output, eg : stock price, salary etc. Linear regression as the name says, finds a linear curve solution to every problem.

LR allocates weight parameter, theta for each of the training features. The predicted output ( $h(\theta)$ ) will be a linear function of features and  $\theta$  coefficients.

$$h_{\theta} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

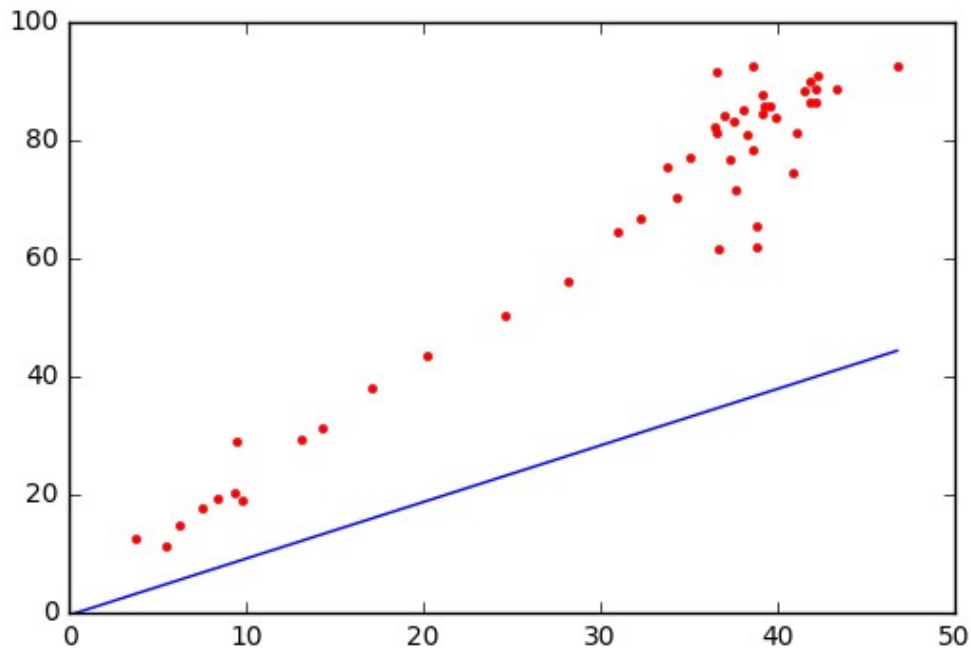
linear regression output. Eqn (1)

During the start of training, each theta is randomly initialized. But during the training, we correct the theta corresponding to each feature such that, the loss (metric of the deviation between expected and predicted output) is minimized. Gradient descend algorithm will be used to align the  $\theta$  values in the right direction.

In the below diagram, each red dots represent the training data and the blue line shows the derived solution.

### Loss function:

In LR, we use mean squared error as the metric of loss. The deviation of expected and actual outputs will be squared and sum up. Derivative of this loss will be used by gradient descend algorithm.

**Advantages:**

- Easy and simple implementation.
- Space complex solution.
- Fast training.
- Value of  $\theta$  coefficients gives an assumption of feature significance.

**Disadvantages:**

- Applicable only if the solution is linear. In many real life scenarios, it may not be the case.
- Algorithm assumes the input residuals (error) to be normal distributed, but may not be satisfied always.
- Algorithm assumes input features to be mutually independent (no co-linearity).

**Assumptions for LR:**

- Linear relationship between the independent and dependent variables.
- Training data to be homoscedastic, meaning the variance of the errors should be somewhat constant.
- Independent variables should not be co-linear.

**Collinearity & Outliers:**

Two features are said to be collinear when one feature can be linearly predicted from the other with somewhat accuracy.

- Collinearity will simply inflate the standard error and causes some significant features to become insignificant during training. Ideally, we should calculate the collinearity prior to training and keep only one feature from highly correlated feature sets.

Outlier is another challenge faced during training. They are data-points that are extreme to normal observations and affects the accuracy of the model.

- Outliers inflate the error functions and affects the curve function and accuracy of the linear regression. Regularization (especially L1) can correct the outliers, by not allowing the  $\theta$  parameters to change violently.
- During Exploratory data analysis phase itself, we should take care of outliers and correct/eliminate them. Box-plot can be used for identifying them.

**Comparison with other models:**

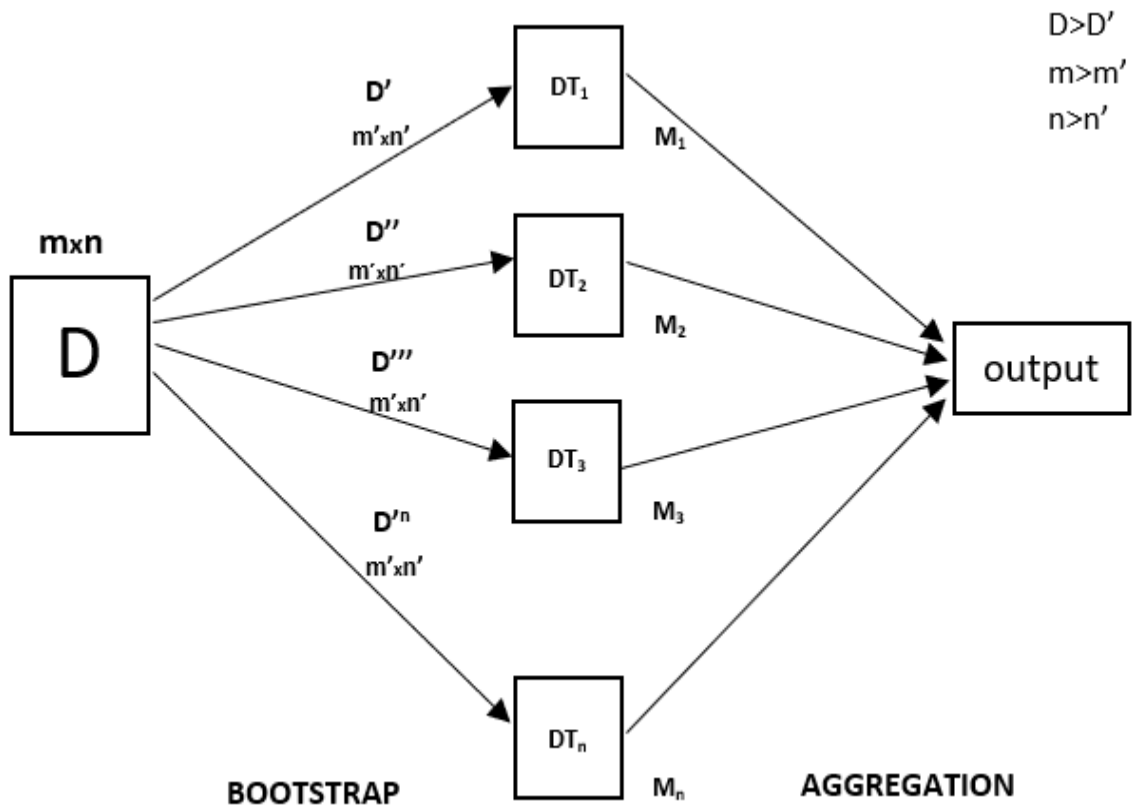
As the linear regression is a regression algorithm, we will compare it with other regression algorithms. One basic difference of linear regression is, LR can only support linear solutions. There are no best models in machine learning that outperforms all others (no free lunch), and efficiency is based on the type of training data distribution.

**Random Forest:**

Every decision tree has high variance, but when we combine all of them together in parallel then the resultant variance is low as each decision tree gets perfectly trained on that particular sample data, and hence the output doesn't depend on one decision tree but on multiple decision trees. In the case of a classification problem, the final output is taken by using the majority voting classifier. In the case of a regression problem, the final output is the mean of all the outputs. This part is called **Aggregation**.

Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as **bagging**. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called **Bootstrap**.



We need to approach the Random Forest regression technique like any other machine learning technique

- Design a specific question or data and get the source to determine the required data.
- Make sure the data is in an accessible format else convert it to the required format.
- Specify all noticeable anomalies and missing data points that may be required to achieve the required data.
- Create a machine learning model
- Set the baseline model that you want to achieve
- Train the data machine learning model.
- Provide an insight into the model with test data
- Now compare the performance metrics of both the test data and the predicted data from the model.
- If it doesn't satisfy your expectations, you can try improving your model accordingly or dating your data, or using another data modeling technique.
- At this stage, you interpret the data you have gained and report accordingly.

Below is a step-by-step sample implementation of Random Forest Regression.

### Implementation:

**Step 1:** Import the required libraries.

**Step 2:** Import and print the dataset

**Step 3:** Select all rows and column 1 from dataset to x and all rows and column 2 as y

# the coding was not shown which is like,

`x= df.iloc[:, :-1]` # " : " means it will select all rows, " : -1 " means that it will ignore last column

`y= df.iloc[:, -1 :]` # " : " means it will select all rows, "-1 : " means that it will ignore all columns except the last one

# The "iloc()" function enables us to select a particular cell of the dataset, that is, it helps us select a value that belongs to a particular row or column from a set of values of a data frame or dataset.

**Step 4:** Fit Random forest regressor to the dataset

**Step 5:** Predicting a new result

**Step 6:** Visualizing the result

### Performance Metrics for Regression Problems

Here, we are going to discuss various performance metrics that can be used to evaluate predictions for regression problems. If one metric is perfect, there is no need for multiple metrics. To understand the benefits and disadvantages of Evaluation metrics because different evaluation metric fits on a different set of a dataset.

#### 1) Mean Absolute Error(MAE)

MAE is a very simple metric which calculates the absolute difference between actual and predicted values. To better understand, let's take an example you have input data and output data and use Linear Regression, which draws a best-fit line.

Now you have to find the MAE of your model which is basically a mistake made by the model known as an error. Now find the difference between the actual value and predicted value that is an absolute error but we have to find the mean absolute of the complete dataset. So, sum all the errors and divide them by a total number of observations and this is MAE. And we aim to get a minimum MAE because this is a loss.

$$\text{MAE} = \frac{1}{N} \sum |Y - \hat{Y}|$$

Annotations in the diagram:

- Divide by total Number of Data Points (points to  $\frac{1}{N}$ )
- Actual Output (points to  $Y$ )
- Predicted Output (points to  $\hat{Y}$ )
- Sum Of (points to  $\sum$ )
- Absolute Value of residual (points to  $|Y - \hat{Y}|$ )

```
from sklearn.metrics import mean_absolute_error
print("MAE",mean_absolute_error(y_test,y_pred))
```

## 2) Mean Squared Error (MSE)

MSE is a most used and very simple metric with a little bit of change in mean absolute error. Mean squared error states that finding the squared difference between actual and predicted value.

So, above we are finding the absolute difference and here we are finding the squared difference.

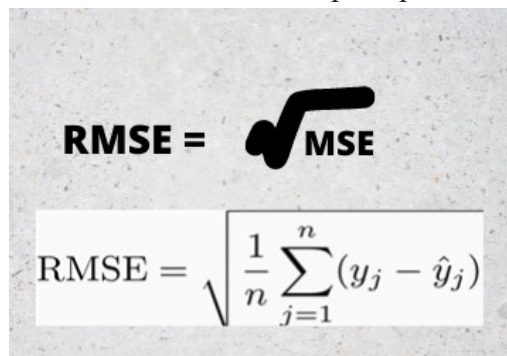
What actually the MSE represents? It represents the squared distance between actual and predicted values. we perform squared to avoid the cancellation of negative terms and it is the benefit of MSE.

$$MSE = \frac{1}{n} \sum \underbrace{\left( y - \hat{y} \right)^2}_{\substack{\text{The square of the difference} \\ \text{between actual and} \\ \text{predicted}}}$$

```
from sklearn.metrics import mean_squared_error
print("MSE",mean_squared_error(y_test,y_pred))
```

## 3) Root Mean Squared Error (RMSE)

As RMSE is clear by the name itself, that it is a simple square root of mean squared error.



The diagram illustrates the relationship between RMSE and MSE. It shows 'RMSE =' followed by a large square root symbol containing 'MSE'. Below this, a box contains the mathematical formula for RMSE:  $RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$ .

for performing RMSE we have to NumPy NumPy square root function over MSE.

```
print("RMSE",np.sqrt(mean_squared_error(y_test,y_pred)))
```

Most of the time people use RMSE as an evaluation metric and mostly when you are working with deep learning techniques the most preferred metric is RMSE.



#### 4) Root Mean Squared Log Error (RMSLE)

Taking the log of the RMSE metric slows down the scale of error. The metric is very helpful when you are developing a model without calling the inputs. In that case, the output will vary on a large scale.

To control this situation of RMSE we take the log of calculated RMSE error and resultant we get as RMSLE.

To perform RMSLE we have to use the NumPy log function over RMSE.

```
print("RMSE",np.log(np.sqrt(mean_squared_error(y_test,y_pred))))
```

It is a very simple metric that is used by most of the datasets hosted for Machine Learning competitions.

#### 5) R Squared (R2)

R2 score is a metric that tells the performance of your model, not the loss in an absolute sense that how many wells did your model perform. In contrast, MAE and MSE depend on the context as we have seen whereas the R2 score is independent of context.

So, with help of R squared we have a baseline model to compare a model which none of the other metrics provides. The same we have in classification problems which we call a threshold which is fixed at 0.5. So basically R2 squared calculates how must regression line is better than a mean line. Hence, R2 squared is also known as Coefficient of Determination or sometimes also known as Goodness of fit.

$$\text{R2 Squared} = 1 - \frac{\text{SSr}}{\text{SSm}}$$

SSr = Squared sum error of regression line

SSm = Squared sum error of mean line

Now, how will you interpret the R2 score? Suppose If the R2 score is zero then the above regression line by mean line is equal means 1 so 1-1 is zero. So, in this case, both lines are overlapping means model performance is worst, it is not capable to take advantage of the output column.

Now the second case is when the R2 score is 1, it means when the division term is zero and it will happen when the regression line does not make any mistake, it is perfect. In the real world, it is not possible.

So we can conclude that as our regression line moves towards perfection, R2 score move towards one. And the model performance improves.

```
from sklearn.metrics import r2_score
```

```
r2 = r2_score(y_test,y_pred)
```

```
print(r2)
```

### 6) Adjusted R Squared

The disadvantage of the R<sup>2</sup> score is while adding new features in data the R<sup>2</sup> score starts increasing or remains constant but it never decreases because it assumes that while adding more data variance of data increases.

But the problem is when we add an irrelevant feature in the dataset then at that time R<sup>2</sup> sometimes starts increasing which is incorrect.

Hence, to control this situation Adjusted R Squared came into existence.

$$R_a^2 = 1 - \left[ \left( \frac{n-1}{n-k-1} \right) \times (1 - R^2) \right]$$

where:

n = number of observations

k = number of independent variables

R<sub>a</sub><sup>2</sup> = adjusted R<sup>2</sup>

Now as K increases by adding some features so the denominator will decrease, n-1 will remain constant. R<sup>2</sup> score will remain constant or will increase slightly so the complete answer will increase and when we subtract this from one then the resultant score will decrease. So this is the case when we add an irrelevant feature in the dataset.

And if we add a relevant feature then the R<sup>2</sup> score will increase and 1-R<sup>2</sup> will decrease heavily and the denominator will also decrease so the complete term decreases, and on subtracting from one the score increases.

n=40

k=2

adj\_r2\_score = 1 - ((1-r2)\*(n-1)/(n-k-1))

print(adj\_r2\_score)

Hence, this metric becomes one of the most important metrics to use during the evaluation of the model.

### Conclusion

We have implemented and compared performance of linear regression and random forest algorithms for predicting the price of the Uber ride from a given pickup point to the agreed drop-off location.

## Lab Assignment No. 01

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Date of Performance: \_\_\_\_\_ Date of Completion \_\_\_\_\_

## Experiment No: 2

**Aim:** Classify the email using the binary classification method. Use **K-Nearest Neighbors and Support Vector Machine** for classification.

**Objectives:**

- Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam.
- Analyse their performance.

**Software Used:**

Linux/Unix/Ubuntu Operating Systems, Jupyter Notebook

Programming Language: Python 3.7.8

Dataset link: The emails.csv dataset on the Kaggle

<https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv>

**Theory:**

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well –

Lazy learning algorithm – KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.

Non-parametric learning algorithm – KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

**Working of KNN Algorithm**

K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new data points which further means that the new data point will be assigned a value based on how closely it matches the points in the training set.

We can understand its working with the help of following steps –

Step 1 – For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 – Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 – For each point in the test data do the following –

3.1 – Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.

3.2 – Now, based on the distance value, sort them in ascending order.

3.3 – Next, it will choose the top K rows from the sorted array.

3.4 – Now, it will assign a class to the test point based on most frequent class of these rows.

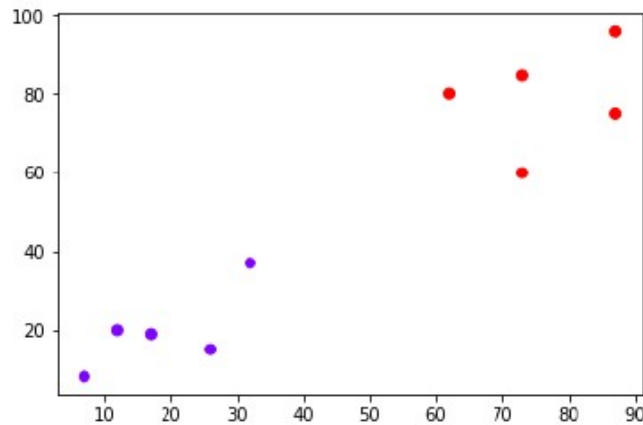
Step 4 – End

### Example

The following is an example to understand the concept of K and working of KNN algorithm. Suppose we have a dataset which can be plotted as follows –

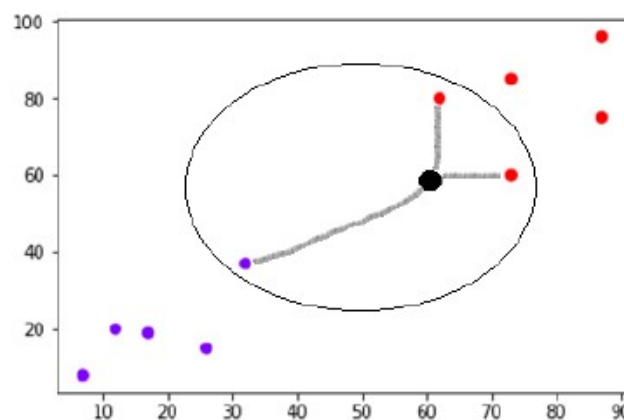
### Concept of K

Now, we need to classify new data point with black dot (at point 60, 60) into blue or red class. We are assuming  $K = 3$  i.e. it would find three nearest data points. It is shown in the next diagram –



### KNN Algorithm

We can see in the above diagram the three nearest neighbors of the data point with black dot. Among those three, two of them lie in Red class, hence the black dot will also be assigned in red class.



## Implementation in Python

As we know K-nearest neighbors (KNN) algorithm can be used for both classification as well as regression. The following are the recipes in Python to use KNN as classifier as well as regressor –

### KNN as Classifier

1. Start with importing necessary python packages.
2. Download the dataset from its weblink
3. Assign column names to the dataset
4. Read dataset to pandas dataframe
5. Data Preprocessing will be done.
6. Divide the data into train and test split. Following code will split the dataset into 60% training data and 40% of testing data
7. Data scaling will be done.
8. Train the model with the help of KNeighborsClassifier class of sklearn
9. At last we need to make prediction
10. Print the results using confusion matrix, Classification report and accuracy.

### Pros and Cons of KNN

#### Pros

- It is very simple algorithm to understand and interpret.
- It is very useful for nonlinear data because there is no assumption about data in this algorithm.
- It is a versatile algorithm as we can use it for classification as well as regression.
- It has relatively high accuracy but there are much better supervised learning models than KNN.

#### Cons

- It is computationally a bit expensive algorithm because it stores all the training data.
- High memory storage required as compared to other supervised learning algorithms.
- Prediction is slow in case of big N.
- It is very sensitive to the scale of data as well as irrelevant features.

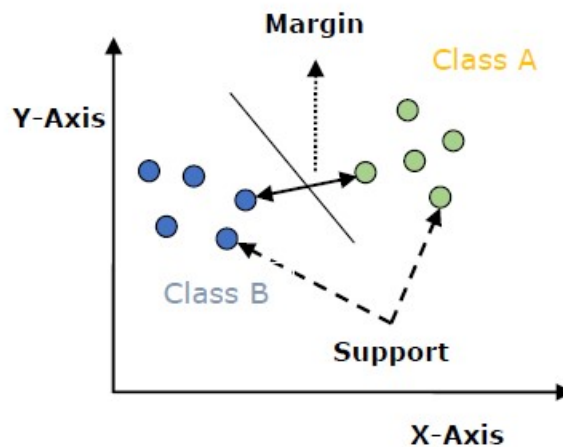
### Introduction to SVM

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

### Working of SVM

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized.

The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).



The followings are important concepts in SVM –

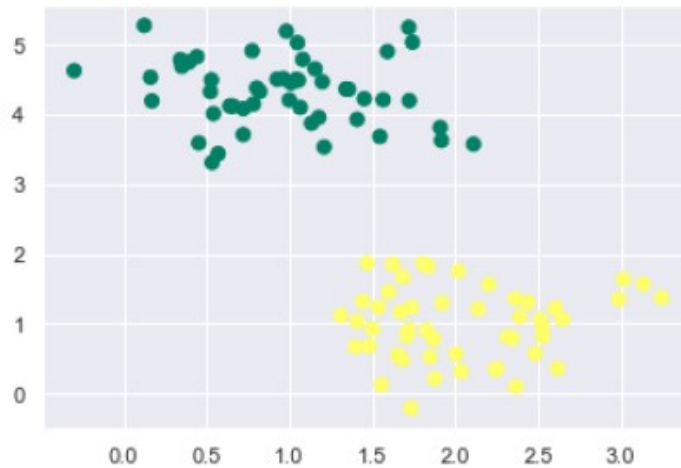
**Support Vectors** – Data points that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.

**Hyperplane** – As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.

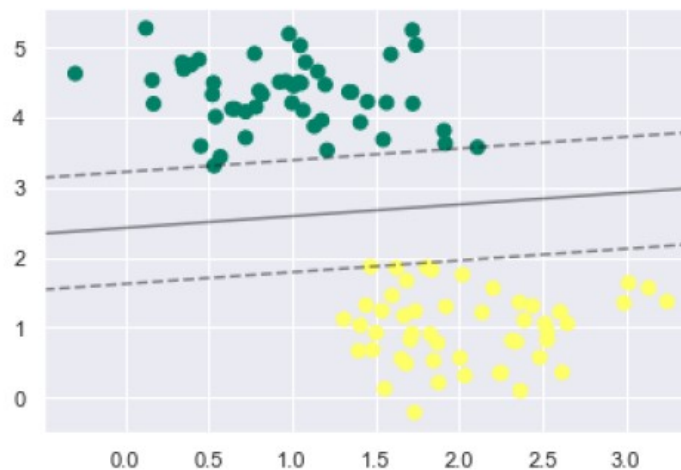
**Margin** – It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

The main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) and it can be done in the following two steps –

First, SVM will generate hyperplanes iteratively that segregates the classes in best way.



Then, it will choose the hyperplane that separates the classes correctly.



### SVM Kernels

In practice, SVM algorithm is implemented with kernel that transforms an input data space into the required form. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts non-separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate.

The following are some of the types of kernels used by SVM –

#### Linear Kernel

It can be used as a dot product between any two observations. The formula of linear kernel is as below –

$$K(x, x_i) = \sum(x * x_i)$$

From the above formula, we can see that the product between two vectors say  $x$  &  $x_i$  is the sum of the multiplication of each pair of input values.



### Polynomial Kernel

It is more generalized form of linear kernel and distinguish curved or nonlinear input space. Following is the formula for polynomial kernel –

$$K(x, x_i) = 1 + \sum (x * x_i)^d$$

Here  $d$  is the degree of polynomial, which we need to specify manually in the learning algorithm.

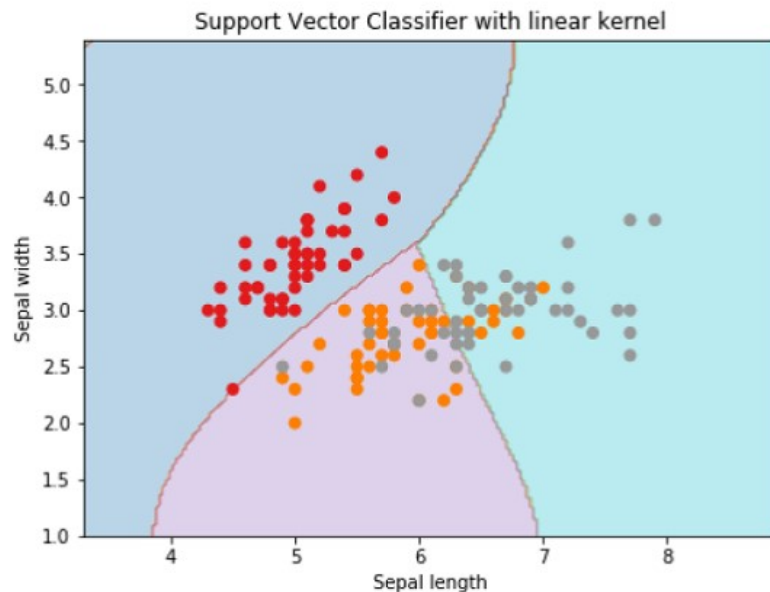
### Radial Basis Function (RBF) Kernel

RBF kernel, mostly used in SVM classification, maps input space in indefinite dimensional space. Following formula explains it mathematically –

$$K(x, x_i) = \exp(-\gamma * \sum ((x - x_i)^2))$$

Here,  $\gamma$  ranges from 0 to 1. We need to manually specify it in the learning algorithm. A good default value of  $\gamma$  is 0.1.

As we implemented SVM for linearly separable data, we can implement it in Python for the data that is not linearly separable. It can be done by using kernels.



### Implementing SVM in Python

1. For implementing SVM in Python we will start with the standard libraries import.
2. Next, we are download a spam dataset from kaggle, having linearly separable data.
3. We know that SVM supports discriminative classification. It divides the classes from each other by simply finding a line in case of two dimensions or manifold in case of multiple dimensions. It is implemented on the given dataset.
4. As discussed, the main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) hence rather than drawing a zero line between classes we can draw around each line a margin of some width up to the nearest point

5. Next, we will use Scikit-Learn's support vector classifier to train an SVM model on this data.
6. Now, for a better understanding, plot the decision functions for 2D SVC
7. For evaluating model, we need to create grid.
8. Next, we need to plot decision boundaries and margins.
9. Now, similarly plot the support vectors.
10. Now, use SVM function to fit our models.

### Pros and Cons of SVM Classifiers

#### Pros of SVM classifiers

SVM classifiers offers great accuracy and work well with high dimensional space. SVM classifiers basically use a subset of training points hence in result uses very less memory.

#### Cons of SVM classifiers

They have high training time hence in practice not suitable for large datasets. Another disadvantage is that SVM classifiers do not work well with overlapping classes.

### Performance Metrics for Classification Problems

Here, we are going to discuss various performance metrics that can be used to evaluate predictions for classification problems.

#### Confusion Matrix

Confusion Matrix is a performance measurement for the machine learning classification problems where the output can be two or more classes. It is a table with combinations of predicted and actual values.

A confusion matrix is defined as the table that is often used to describe the performance of a classification model on a set of the test data for which the true values are known.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

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

**True Positive:** We predicted positive and it's true.

**True Negative:** We predicted negative and it's true.

**False Positive (Type 1 Error) -** We predicted positive and it's false.

**False Negative (Type 2 Error) -** We predicted negative and it's false.

We can use `confusion_matrix` function of `sklearn.metrics` to compute Confusion Matrix of our classification model.

### Accuracy

Accuracy simply measures how often the classifier correctly predicts. We can define accuracy as the ratio of the number of correct predictions and the total number of predictions.

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

We can use `accuracy_score` function of `sklearn.metrics` to compute accuracy of our classification model.

### Classification Report

This report consists of the scores of Precisions, Recall, F1 and Support. They are explained as follows –

**1. Precision**— precision explains how many of the correctly predicted cases actually turned out to be positive. Precision is useful in the cases where False Positive is a higher concern than False Negatives. The importance of Precision is in music or video recommendation systems, e-commerce websites, etc. where wrong results could lead to customer churn and this could be harmful to the business.

**Precision for a label is defined as the number of true positives divided by the number of predicted positives.**

$$\text{Precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}}$$

**2. Recall (Sensitivity)**— Recall explains how many of the actual positive cases we were able to predict correctly with our model. It is a useful metric in cases where False Negative is of higher concern than False Positive. It is important in medical cases where it doesn't matter whether we raise a false alarm but the actual positive cases should not go undetected!

**Recall for a label is defined as the number of true positives divided by the total number of actual positives.**

$$\text{Recall} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalseNegative}}$$

**3. F1 Score**— it gives a combined idea about Precision and Recall metrics. It is maximum when Precision is equal to Recall.

**F1 Score is the harmonic mean of precision and recall.**

$$F1 = 2. \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

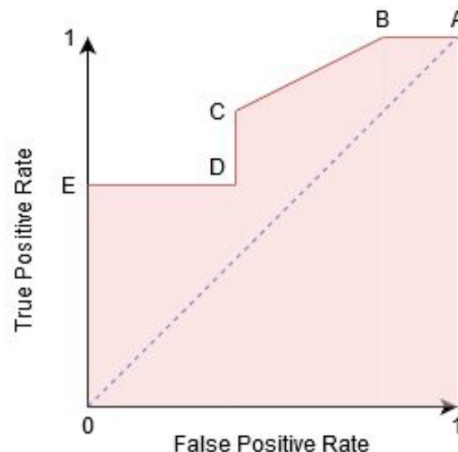
The F1 score punishes extreme values more. F1 Score could be an effective evaluation metric in the following cases:

- When FP and FN are equally costly.
- Adding more data doesn't effectively change the outcome
- True Negative is high

**4. AUC-ROC**—The Receiver Operator Characteristic (ROC) is a probability curve that plots the TPR (True Positive Rate) against the FPR (False Positive Rate) at various threshold values and separates the 'signal' from the 'noise'.

The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes. From the graph, we simply say the area of the curve ABDE and the X and Y-axis.

From the graph shown below, the greater the AUC, the better is the performance of the model at different threshold points between positive and negative classes. This simply means that When AUC is equal to 1, the classifier is able to perfectly distinguish between all Positive and Negative class points. When AUC is equal to 0, the classifier would be predicting all Negatives as Positives and vice versa. When AUC is 0.5, the classifier is not able to distinguish between the Positive and Negative classes.



Working of AUC—In a ROC curve, the X-axis value shows False Positive Rate (FPR), and Y-axis shows True Positive Rate (TPR). Higher the value of X means higher the number of False Positives (FP) than True Negatives (TN), while a higher Y-axis value indicates a higher number of TP than FN. So, the choice of the threshold depends on the ability to balance between FP and FN.

**5. Log Loss**—Log loss (Logistic loss) or Cross-Entropy Loss is one of the major metrics to assess the performance of a classification problem.

For a single sample with true label  $y \in \{0, 1\}$  and a probability estimate  $p = \Pr(y=1)$ , the log loss is:

$$\text{logloss}_{(N=1)} = y \log(p) + (1 - y) \log(1 - p)$$

Understanding how well a machine learning model will perform on unseen data is the main purpose behind working with these evaluation metrics. Metrics like accuracy, precision, recall are good ways to evaluate classification models for balanced datasets, but if the data is imbalanced then other methods like ROC/AUC perform better in evaluating the model performance.

ROC curve isn't just a single number but it's a whole curve that provides nuanced details about the behavior of the classifier. It is also hard to quickly compare many ROC curves to each other.

### **Conclusion:**

Hence, we have used two classification algorithms KNN and SVM for spam email detection problems and analyzed their performance using evaluation metrics.

**Lab Assignment No. 02**

<b>Write-up</b>	<b>Correctness of Program</b>	<b>Documentation of Program</b>	<b>Viva</b>	<b>Timely Completion</b>	<b>Total</b>	<b>Dated Sign of Subject Teacher</b>
<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>20</b>	

Date of Performance: \_\_\_\_\_ Date of Completion \_\_\_\_\_

## Experiment No: 03

**Aim:** Implement **Gradient Descent Algorithm** to find the local minima of a function. For example, find the local minima of the function  $y=(x+3)^2$  starting from the point  $x=2$ .

### Objectives:

- To find the local minima of a function.

### Software Used:

Linux/Unix/Ubuntu Operating Systems, Jupyter Notebook  
Programming Language: Python 3.7.8

### Theory:

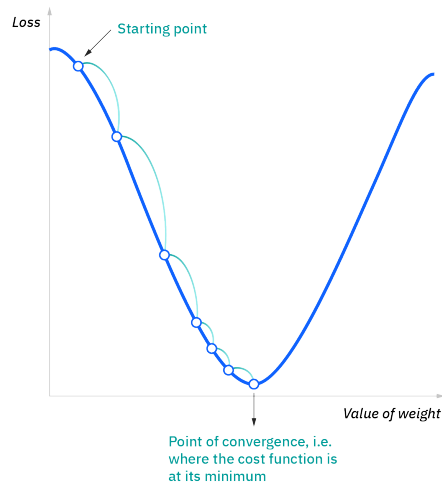
#### What is gradient descent?

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error. Once machine learning models are optimized for accuracy, they can be powerful tools for artificial intelligence (AI) and computer science applications.

#### How does gradient descent work?

Before we dive into gradient descent, it may help to review some concepts from linear regression. You may recall the following formula for the slope of a line, which is  $y = mx + b$ , where  $m$  represents the slope and  $b$  is the intercept on the y-axis.

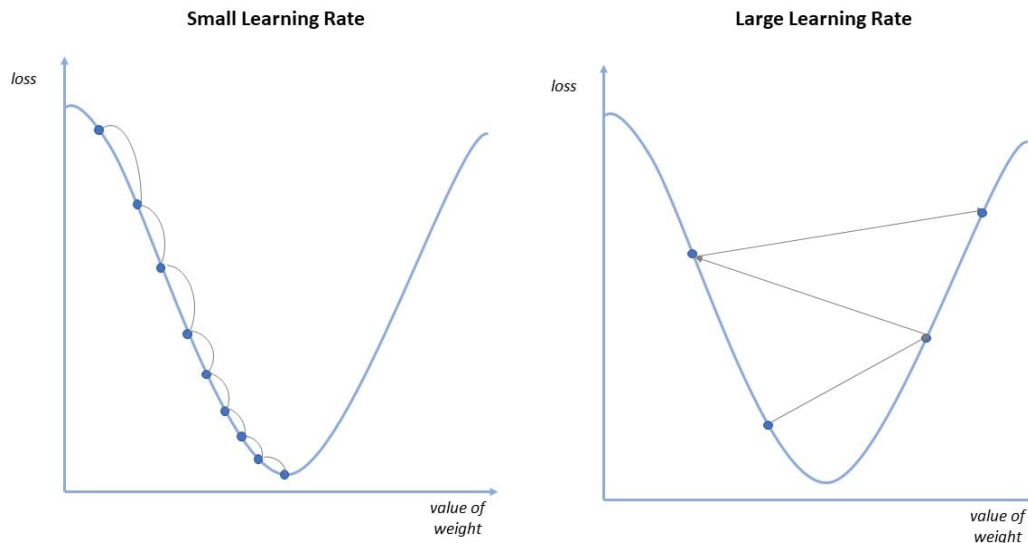
You may also recall plotting a scatterplot in statistics and finding the line of best fit, which required calculating the error between the actual output and the predicted output ( $\hat{y}$ ) using the mean squared error formula. The gradient descent algorithm behaves similarly, but it is based on a convex function, such as the one below:



The starting point is just an arbitrary point for us to evaluate the performance. From that starting point, we will find the derivative (or slope), and from there, we can use a tangent line to observe the steepness of the slope. The slope will inform the updates to the parameters—i.e. the weights and bias. The slope at the starting point will be steeper, but as new parameters are generated, the steepness should gradually reduce until it reaches the lowest point on the curve, known as the point of convergence.

Similar to finding the line of best fit in linear regression, the goal of gradient descent is to minimize the cost function, or the error between predicted and actual  $y$ . In order to do this, it requires two data points—a direction and a learning rate. These factors determine the partial derivative calculations of future iterations, allowing it to gradually arrive at the local or global minimum (i.e. point of convergence). More detail on these components can be found below:

- **Learning rate** (also referred to as step size or the alpha) is the size of the steps that are taken to reach the minimum. This is typically a small value, and it is evaluated and updated based on the behavior of the cost function. High learning rates result in larger steps but risks overshooting the minimum. Conversely, a low learning rate has small step sizes. While it has the advantage of more precision, the number of iterations compromises overall efficiency as this takes more time and computations to reach the minimum.



- **The cost (or loss) function** measures the difference, or error, between actual  $y$  and predicted  $y$  at its current position. This improves the machine learning model's efficacy by providing feedback to the model so that it can adjust the parameters to minimize the error and find the local or global minimum. It continuously iterates, moving along the direction of steepest descent (or the negative gradient) until the



cost function is close to or at zero. At this point, the model will stop learning. Additionally, while the terms, cost function and loss function, are considered synonymous, there is a slight difference between them. It's worth noting that a loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.

### Challenges with gradient descent

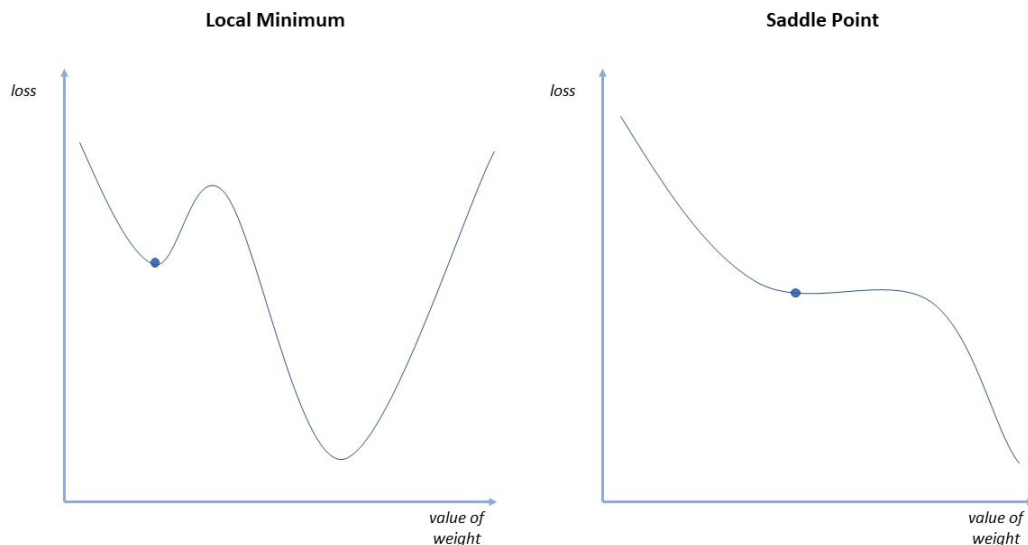
While gradient descent is the most common approach for optimization problems, it does come with its own set of challenges. Some of them include:

#### Local minima and saddle points

For convex problems, gradient descent can find the global minimum with ease, but as nonconvex problems emerge, gradient descent can struggle to find the global minimum, where the model achieves the best results.

Recall that when the slope of the cost function is at or close to zero, the model stops learning. A few scenarios beyond the global minimum can also yield this slope, which are local minima and saddle points. Local minima mimic the shape of a global minimum, where the slope of the cost function increases on either side of the current point. However, with saddle points, the negative gradient only exists on one side of the point, reaching a local maximum on one side and a local minimum on the other. Its name inspired by that of a horse's saddle.

Noisy gradients can help the gradient escape local minimums and saddle points.



To implement a gradient descent algorithm, we require a cost function that needs to be minimized, the number of iterations, a learning rate to determine the step size at each iteration while moving towards the minimum, partial derivate for weight & bias to update the parameters at each iteration, and a prediction function.

Till now we have seen the parameters required for gradient descent. Now let us map the parameters with the gradient descent algorithm and work on an example to better understand gradient descent. Let us consider a parabolic equation  $y=(x+3)^2$ . Now let us see the algorithm for gradient descent and how we can obtain the local minima by applying gradient descent:

### Algorithm for Gradient Descent

Steps should be made in proportion to the negative of the function gradient (move away from the gradient) at the current point to find local minima. Gradient Ascent is the procedure for approaching a local maximum of a function by taking steps proportional to the positive of the gradient (moving towards the gradient).

repeat until convergence

```
{
w = w - (learning_rate * (dJ/dw))
b = b - (learning_rate * (dJ/db))
}
```

**Step 1:** Initializing all the necessary parameters and deriving the gradient function for the parabolic equation  $(x+3)^2$ . The derivative of parabolic equation  $(x+3)^2$  will be  $2x+6$ .

$x_0 = 2$  (random initialization of  $x$ )

$learning\_rate = 0.01$  (to determine the step size while moving towards local minima)

**Step 2:** Let us perform 3 iterations of gradient descent:

For each iteration keep on updating the value of  $x$  based on the gradient descent formula.

Iteration 1:

```
x1 = x0 - (learning_rate * gradient)
x1 = 2 - (0.01 * (2 * 2 + 6))
x1 = 2 - 0.1
x1 = 1.9
```

Iteration 2:

```
x2 = x1 - (learning_rate * gradient)
x2 = 1.9 - (0.01 * (2 * 1.9 + 6))
x2 = 1.9 - 0.098
x2 = 1.802
```

Iteration 3:

```
x3 = x2 - (learning_rate * gradient)
x3 = 1.802 - (0.01 * (2 * 1.802 + 6))
x3 = 1.802 - 0.09604
x3 = 1.70596
```

From the above three iterations of gradient descent, we can notice that the value of  $x$  is decreasing iteration by iteration and will slowly converge to 0 (local minima) by running the gradient descent for more iterations. Now you might have a question, for how many iterations we should run gradient descent?

**We can set a stopping threshold** i.e. when the difference between the previous and the present value of  $x$  becomes less than the stopping threshold we stop the iterations. When it comes to the implementation of gradient descent for machine learning algorithms and deep learning algorithms we try to minimize the cost function in the algorithms using gradient

descent. Now that we are clear with the gradient descent's internal working, let us look into the python implementation of gradient descent where we will be minimizing the cost function of the linear regression algorithm and finding the best fit line. In our case the parameters are below mentioned:

### Prediction Function

The prediction function for the linear regression algorithm is a linear equation given by  $y=wx+b$ .

$\text{prediction\_function}(y) = (w * x) + b$

Here, x is the independent variable

y is the dependent variable

w is the weight associated with input variable

b is the bias

### Cost Function

The cost function is used to calculate the loss based on the predictions made. In linear regression, we use mean squared error to calculate the loss. Mean Squared Error is the sum of the squared differences between the actual and predicted values.

$$\text{Cost Function } (J) = \left(\frac{1}{n}\right) \sum_{i=1}^n (y_i - (wx_i + b))^2$$

Here, n is the number of samples

### Partial Derivatives (Gradients)

Calculating the partial derivatives for weight and bias using the cost function. We get:

$$\frac{dJ}{dw} = \left(\frac{-2}{n}\right) \sum_{i=1}^n x_i * (y_i - (wx_i + b))$$

$$\frac{dJ}{db} = \left(\frac{-2}{n}\right) \sum_{i=1}^n (y_i - (wx_i + b))$$

### Parameter Updation

Updating the weight and bias by subtracting the multiplication of learning rates and their respective gradients.

$w = w - (\text{learning\_rate} * (dJ/dw))$

$b = b - (\text{learning\_rate} * (dJ/db))$

### Python Implementation for Gradient Descent

In the implementation part, we will be writing two functions, one will be the cost functions that take the actual output and the predicted output as input and returns the loss, the second will be the actual gradient descent function which takes the independent variable, target variable as input and finds the best fit line using gradient descent algorithm. The iterations, learning\_rate, and stopping threshold are the tuning parameters

for the gradient descent algorithm and can be tuned by the user. In the main function, we will be initializing linearly related random data and applying the gradient descent algorithm on the data to find the best fit line. The optimal weight and bias found by using the gradient descent algorithm are later used to plot the best fit line in the main function. The iterations specify the number of times the update of parameters must be done, the stopping threshold is the minimum change of loss between two successive iterations to stop the gradient descent algorithm.

**Conclusion:**

Hence, we have studied local minima problem in gradient descent algorithm and implemented GD for function  $y = (x+3)^2$

**Lab Assignment No. 03**

<b>Write-up</b>	<b>Correctness of Program</b>	<b>Documentation of Program</b>	<b>Viva</b>	<b>Timely Completion</b>	<b>Total</b>	<b>Dated Sign of Subject Teacher</b>
<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>20</b>	

Date of Performance: \_\_\_\_\_ Date of Completion \_\_\_\_\_

## Experiment No: 04

**Aim:** Implement **K-Nearest Neighbors** algorithm on diabetes.csv dataset.

**Objectives:**

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

**Software Used:**

Linux/Unix/Ubuntu Operating Systems, Jupyter Notebook

Programming Language: Python 3.7.8

Dataset link: <https://www.kaggle.com/datasets/abdallamahgoub/diabetes>

**Theory:**

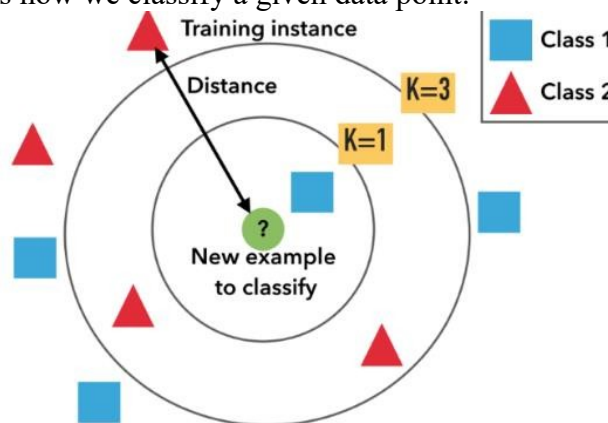
**K-Nearest Neighbors (kNN) Algorithm-**

KNN is a non-parametric lazy learning algorithm. That is a pretty concise statement. When you say a technique is non parametric, it means that it does not make any assumptions on the underlying data distribution. This is pretty useful, as in the real world, most of the practical data does not obey the typical theoretical assumptions made. Non parametric algorithms like KNN come to the rescue here.

It is also a lazy algorithm. What this means is that it does not use the training data points to do any generalization. In other words, there is no explicit training phase or it is very minimal. This means the training phase is pretty fast. Lack of generalization means that KNN keeps all the training data. More exactly, all the training data is needed during the testing phase. This is in contrast to other techniques like SVM where you can discard all non-support vectors without any problem. Most of the lazy algorithms – especially KNN – makes decision based on the entire training data set (in the best case a subset of them).

The dichotomy is pretty obvious here – There is a non-existent or minimal training phase but a costly testing phase. The cost is in terms of both time and memory. More time might be needed as in the worst case, all data points might take part in decision. More memory is needed as we need to store all training data.

KNN Algorithm is based on feature similarity: How closely out-of-sample features resemble our training set determines how we classify a given data point:



Example of k-NN classification. The test sample (inside circle) should be classified either to the first class of blue squares or to the second class of red triangles. If  $k = 3$  (outside circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If, for example  $k = 5$  it is assigned to the first class (3 squares vs. 2 triangles outside the outer circle).

KNN can be used for classification—the output is a class membership (predicts a class—a discrete value). An object is classified by a majority vote of its neighbours, with the object being assigned to the class most common among its  $k$  nearest neighbours. It can also be used for regression—output is the value for the object (predicts continuous values). This value is the average (or median) of the values of its  $k$  nearest neighbours.

### Assumptions in KNN

Before using KNN, let us revisit some of the assumptions in KNN.

KNN assumes that the data is in a feature space. More exactly, the data points are in a metric space. The data can be scalars or possibly even multidimensional vectors. Since the points are in feature space, they have a notion of distance – This need not necessarily be Euclidean distance although it is the one commonly used.

Each of the training data consists of a set of vectors and class label associated with each vector. In the simplest case, it will be either + or – (for positive or negative classes). But KNN, can work equally well with arbitrary number of classes.

We are also given a single number " $k$ ". This number decides how many neighbours (where neighbours is defined based on the distance metric) influence the classification. This is usually an odd number if the number of classes is 2. If  $k=1$ , then the algorithm is simply called the nearest neighbour algorithm.

### KNN for Classification

Let's see how to use KNN for classification. In this case, we are given some data points for training and also a new unlabelled data for testing. Our aim is to find the class label for the new point. The algorithm has different behaviour based on  $k$ .

#### Case 1: $k = 1$ or Nearest Neighbour Rule

This is the simplest scenario. Let  $x$  be the point to be labelled. Find the point closest to  $x$ . Let it be  $y$ . Now nearest neighbour rule asks to assign the label of  $y$  to  $x$ . This seems too simplistic and sometimes even counter intuitive. If you feel that this procedure will result a huge error, you are right – but there is a catch. This reasoning holds only when the number of data points is not very large.

If the number of data points is very large, then there is a very high chance that label of  $x$  and  $y$  are same. An example might help – Let's say you have a (potentially) biased coin. You toss it for 1 million time and you have got head 900,000 times. Then most likely your next call will be head. We can use a similar argument here.

Let me try an informal argument here - Assume all points are in a  $D$  dimensional plane. The number of points is reasonably large. This means that the density of the plane at any point is fairly high. In other words, within any subspace there is adequate number of points. Consider a point  $x$  in the subspace which also has a lot of neighbours. Now let  $y$  be the nearest neighbour. If  $x$  and  $y$  are sufficiently close, then we can assume that probability that  $x$  and  $y$  belong to same class is fairly same – Then by decision theory,  $x$  and  $y$  have the same class.

### **Case 2: $k = K$ or $k$ -Nearest Neighbour Rule**

This is a straightforward extension of 1NN. Basically what we do is that we try to find the  $k$  nearest neighbour and do a majority voting. Typically  $k$  is odd when the number of classes is 2. Let's say  $k = 5$  and there are 3 instances of  $C1$  and 2 instances of  $C2$ . In this case, KNN says that new point has to label as  $C1$  as it forms the majority. We follow a similar argument when there are multiple classes.

One of the straight forward extension is not to give 1 vote to all the neighbours. A very common thing to do is weighted kNN where each point has a weight which is typically calculated using its distance. For eg. under inverse distance weighting, each point has a weight equal to the inverse of its distance to the point to be classified.

This means that neighbouring points have a higher vote than the farther points.

It is quite obvious that the accuracy \*might\* increase when you increase  $k$  but the computation cost also increases.

### **Some Basic Observations**

1. If we assume that the points are  $d$ -dimensional, then the straight forward implementation of finding  $k$  Nearest Neighbour takes  $O(dn)$  time.
2. We can think of KNN in two ways – One way is that KNN tries to estimate the posterior probability of the point to be labelled (and apply Bayesian decision theory based on the posterior probability). An alternate way is that KNN calculates the decision surface (either implicitly or explicitly) and then uses it to decide on the class of the new points.
3. There are many possible ways to apply weights for KNN – One popular example is the Shephard's method.
4. Even though the naive method takes  $O(dn)$  time, it is very hard to do better unless we make some other assumptions. There are some efficient data structures like KD- Tree which can reduce the time complexity but they do it at the cost of increased training time and complexity.
5. In KNN,  $k$  is usually chosen as an odd number if the number of classes is 2.
6. Choice of  $k$  is very critical – A small value of  $k$  means that noise will have a higher influence on the result. A large value make it computationally expensive and kind of defeats the basic philosophy behind KNN (that points that are near might have similar densities or classes) .A simple approach to select  $k$  is set
7. There are some interesting data structures and algorithms when you apply KNN on graphs – See Euclidean minimum spanning tree and nearest neighbour graph.



8. There are also some nice techniques like condensing, search tree and partial distance that try to reduce the time taken to find the k nearest neighbour.

### **Applications of KNN**

KNN is a versatile algorithm and is used in a huge number of fields. Let us take a look at few uncommon and non-trivial applications.

#### **1. Nearest Neighbour based Content Retrieval**

This is one the fascinating applications of KNN – Basically we can use it in Computer Vision for many cases – You can consider handwriting detection as a rudimentary nearest neighbour problem. The problem becomes more fascinating if the content is a video – given a video find the video closest to the query from the database – Although this looks abstract, it has lot of practical applications – Eg : Consider ASL (American Sign Language) . Here the communication is done using hand gestures.

So let's say if we want to prepare a dictionary for ASL so that user can query it doing a gesture. Now the problem reduces to find the (possibly k) closest gesture(s) stored in the database and show to user. In its heart it is nothing but a KNN problem.

#### **2. Protein-Protein interaction and 3D structure prediction**

Graph based KNN is used in protein interaction prediction. Similarly KNN is used in structure prediction.

3. Credit ratings—collecting financial characteristics vs. comparing people with similar financial features to a database. By the very nature of a credit rating, people who have similar financial details would be given similar credit ratings. Therefore, they would like to be able to use this existing database to predict a new customer's credit rating, without having to perform all the calculations.

4. Should the bank give a loan to an individual? Would an individual default on his or her loan? Is that person closer in characteristics to people who defaulted or did not default on their loans?

5. In political science—classing a potential voter to a “will vote” or “will not vote”, or to “vote Democrat” or “vote Republican”.

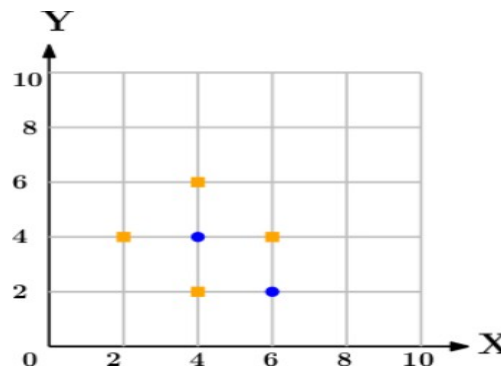
6. More advance examples could include handwriting detection (like OCR), image recognition and even video recognition.

#### **Pros:**

- No assumptions about data—useful, for example, for nonlinear data
- Simple algorithm—to explain and understand/interpret
- High accuracy (relatively)—it is pretty high but not competitive in comparison to better supervised learning models
- Versatile—useful for classification or regression

#### **Cons:**

- Computationally expensive—because the algorithm stores all of the training data
- High memory requirement
- Stores all (or almost all) of the training data
- Prediction stage might be slow (with big N)
- Sensitive to irrelevant features and the scale of the data



Given Diagram Represent Positive and Negative Point with Colour.

In the following diagram let blue circles indicate positive examples and orange squares indicate negative examples. We want to use k-NN algorithm for classifying the points. If  $k=3$ , find-the class of the point (6, 6). Extend the same example for Distance-Weighted k-NN and locally weighted Averaging

### Algorithm

1. Import the Required Packages
2. Read Given Dataset
3. Import KNeighborshood Classifier and create object of it.
4. Predict the class for the point (6, 6) w.r.t to General KNN.
5. Predict the class for the point (6, 6) w.r.t to Distance Weighted KNN.

### Conclusion:

In this way we learn KNN Classification to predict the General and Distance Weighted KNN for given dataset of diabetes.csv for Positive or Negative.

**Lab Assignment No. 04**

<b>Write-up</b>	<b>Correctness of Program</b>	<b>Documentation of Program</b>	<b>Viva</b>	<b>Timely Completion</b>	<b>Total</b>	<b>Dated Sign of Subject Teacher</b>
<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>20</b>	

Date of Performance: \_\_\_\_\_ Date of Completion \_\_\_\_\_

## Experiment No: 05

**Aim:** Implement **K-Means clustering/ hierarchical clustering** on sales\_data\_sample.csv dataset.

**Objectives:**

Determine the number of clusters using the elbow method.

**Software Used:**

Linux/Unix/Ubuntu Operating Systems, Jupyter Notebook

Programming Language: Python 3.7.8

Dataset link: <https://www.kaggle.com/datasets/kyanyoga/sample-sales-data>

**Theory:**

**What is Clustering?**

Clustering is dividing data points into homogeneous classes or clusters: Points in the same group are as similar as possible

Points in different group are as dissimilar as possible

When a collection of objects is given, we put objects into group based on similarity.

**Application of Clustering:**

Clustering is used in almost all the fields. You can infer some ideas from Example 1 to come up with lot of clustering applications that you would have come across.

Listed here are few more applications, which would add to what you have learnt.

- ☐ Clustering helps marketers improve their customer base and work on the target areas. It helps group people (according to different criteria's such as willingness, purchasing power etc.) based on their similarity in many ways related to the product under consideration.
- ☐ Clustering helps in identification of groups of houses on the basis of their value, type and geographical locations.
- ☐ Clustering is used to study earth-quake. Based on the areas hit by an earthquake in a region, clustering can help analyse the next probable location where earthquake can occur.

**Clustering Algorithm:**

A Clustering Algorithm tries to analyse natural groups of data on the basis of some similarity. It locates the centroid of the group of data points. To carry out effective clustering, the algorithm evaluates the distance between each point from the centroid of the cluster.

The goal of clustering is to determine the intrinsic grouping in a set of unlabelled data.

**What is K-means Clustering?**

K-means (Macqueen, 1967) is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining.



### K-means Clustering – Example 1:

A pizza chain wants to open its delivery centres across a city. What do you think would be the possible challenges?

- They need to analyse the areas from where the pizza is being ordered frequently.
- They need to understand as to how many pizza stores has to be opened to cover delivery in the area.
- They need to figure out the locations for the pizza stores within all these areas in order to keep the distance between the store and delivery points minimum.

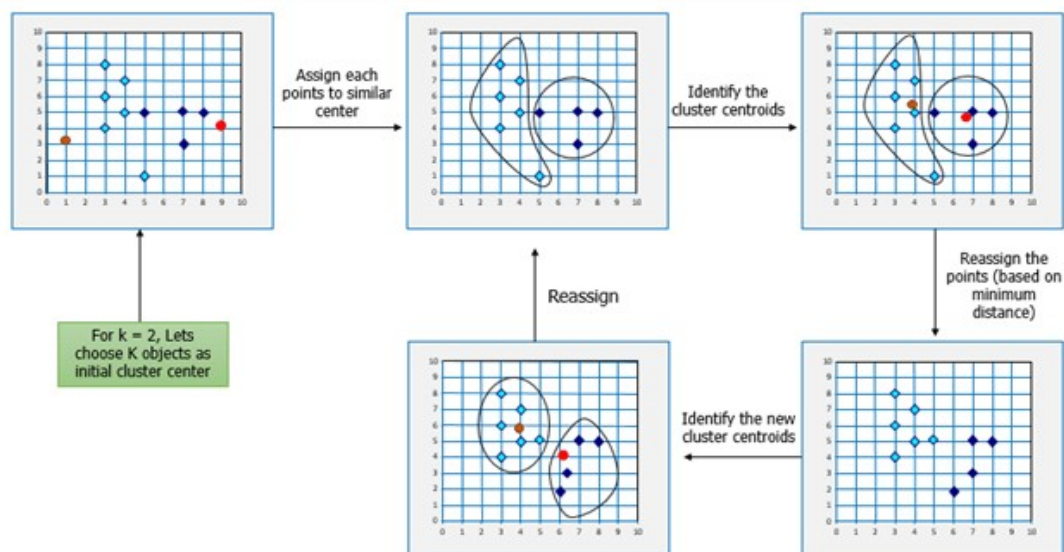
Resolving these challenges includes a lot of analysis and mathematics. We would now learn about how clustering can provide a meaningful and easy method of sorting out such real life challenges. Before that let's see what clustering is.

### K-means Clustering Method:

If  $k$  is given, the K-means algorithm can be executed in the following steps:

- Partition of objects into  $k$  non-empty subsets
- Identifying the cluster centroids (mean point) of the current partition.
- Assigning each point to a specific cluster
- Compute the distances from each point and allot points to the cluster where the distance from the centroid is minimum.
- After re-allotting the points, find the centroid of the new cluster formed.

**The step by step process:**



**Mathematical Formulation for K-means Algorithm:**

K-Means clustering intends to partition  $n$  objects into  $k$  clusters in which each object nearest mean. This method produces exactly  $k$  different clusters of greatest possible  $d$  clusters  $k$  leading to the greatest separation (distance) is not known as a priori and mu objective of K-Means clustering is to minimize total intra-cluster variance, or, the square

The diagram shows the objective function  $J = \sum_{j=1}^k \sum_{i=1}^n \|x_i^{(j)} - c_j\|^2$  with several annotations:
 

- An arrow points from the text "number of clusters" to the variable  $k$  in the first summation.
- An arrow points from the text "number of cases" to the variable  $n$  in the second summation.
- An arrow points from the text "case  $i$ " to the term  $x_i^{(j)}$ .
- An arrow points from the text "centroid for cluster  $j$ " to the term  $c_j$ .
- A bracket under the term  $\|x_i^{(j)} - c_j\|^2$  is labeled "Distance function".
- An arrow points from the text "objective function" to the variable  $J$ .

**Algorithm**

1. Import the Required Packages
2. Create dataset using DataFrame
3. Find centroid points
4. Plot the given points
5. For  $i$  in centroids ():
6. Plot given elements with centroid elements
7. Import KMeans class and create object of it
8. Using labels find population around centroid
9. Find new centroids

**Conclusion:**

In this way we learn K-mean Clustering Algorithm and used implementation to determine sales data in different regions.

**Lab Assignment No. 05**

<b>Write-up</b>	<b>Correctness of Program</b>	<b>Documentation of Program</b>	<b>Viva</b>	<b>Timely Completion</b>	<b>Total</b>	<b>Dated Sign of Subject Teacher</b>
<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>20</b>	

Date of Performance: \_\_\_\_\_ Date of Completion \_\_\_\_\_