Lecture Notes

Analytics using PySpark

You earlier learnt about various machine learning algorithms and also got an understanding of their implementation using stand-alone Python on your local machine. However, while handling a huge data set, you had to switch to Spark for training the algorithms. In this module, you first performed basic EDA using Spark MLlib library and then learnt about some basic machine learning algorithms and got an understanding of their implementation using PySpark.

Linear Regression

In the first session of this module, you learnt about linear regression, a supervised learning algorithm. You were introduced to some basic model-building techniques. You also got an understanding of the implementation of the linear regression algorithm.

Understanding machine learning algorithms

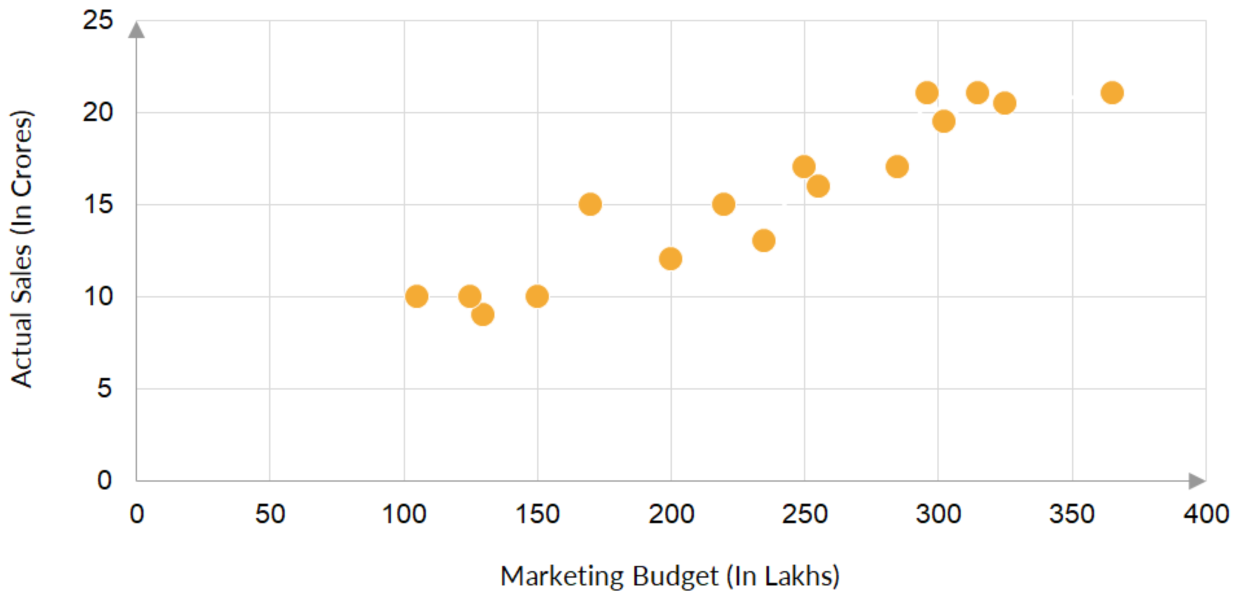
Machine learning algorithms are classified into the following two categories:

1. **Supervised learning**: Linear regression and logistic regression
2. **Unsupervised learning**: Clustering

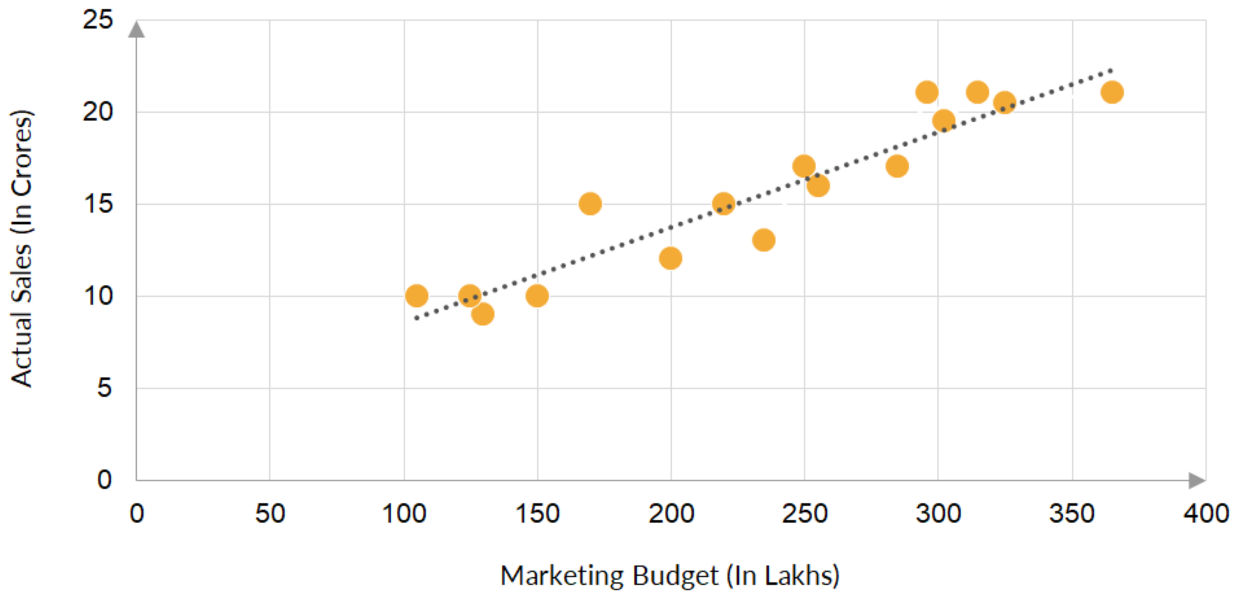
The linear regression model attempts to explain the relationship between a dependent variable and an independent variable using a straight line. In the example regarding the sales prediction of a company based on the marketing budget:

- Sales prediction was a dependent variable, and
- The marketing budget was an independent variable.

Now, the first step of linear regression involves visualising the historic data points present in the given data set and drawing a scatter plot, as shown below.



The next step involves fitting a linear or straight line through these data points, as shown below, so that the model can learn the behaviour from the data points and predict the value of the actual sales from the given marketing budget.



The equation of any straight line is expressed as follows:

$$Y = \beta_0 + \beta_1 X$$

Where, β_0= Intercept
β_1= Slope

You then learnt about residuals, which is the difference between the actual value and the predicted value. The residual is also interpreted as an error term between the actual values and the predicted values.

$e_i = y_i - y_{pred}$

The sum of squared errors is expressed as follows:

$e_1^2 + e_2^2 + \_\_\_ + e_n^2$ (Residual Sum of Squares)
$RSS = (Y_1 - \beta_0 - \beta_1 X_1)^2 + (Y_2 - \beta_0 - \beta_1 X_2)^2 \_\_\_\_ + (Y_n - \beta_0 - \beta_1 X_n)^2$
$RSS = \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_i)^2$

Now in order to find the line of best fit, the value of the residual sum of squares (RSS) should be minimised to obtain the optimal value of β_0 and β_1. You can minimise the RSS using the following methods:

- Gradient descent
- Differentiation

One of the drawbacks of the RSS is that it considers the absolute number. To mitigate this limitation, the strength of the linear regression model can be explained using R^2 as follows:

$R^2 = 1 - (TSS/RSS)$

Where, RSS = Residual sum of squares
TSS = Total sum of squares

TSS is a measure of how a data set varies around a central number (for example, the mean).
When a model has a high R^2 score, it performs well over the data set.

Logistic Regression

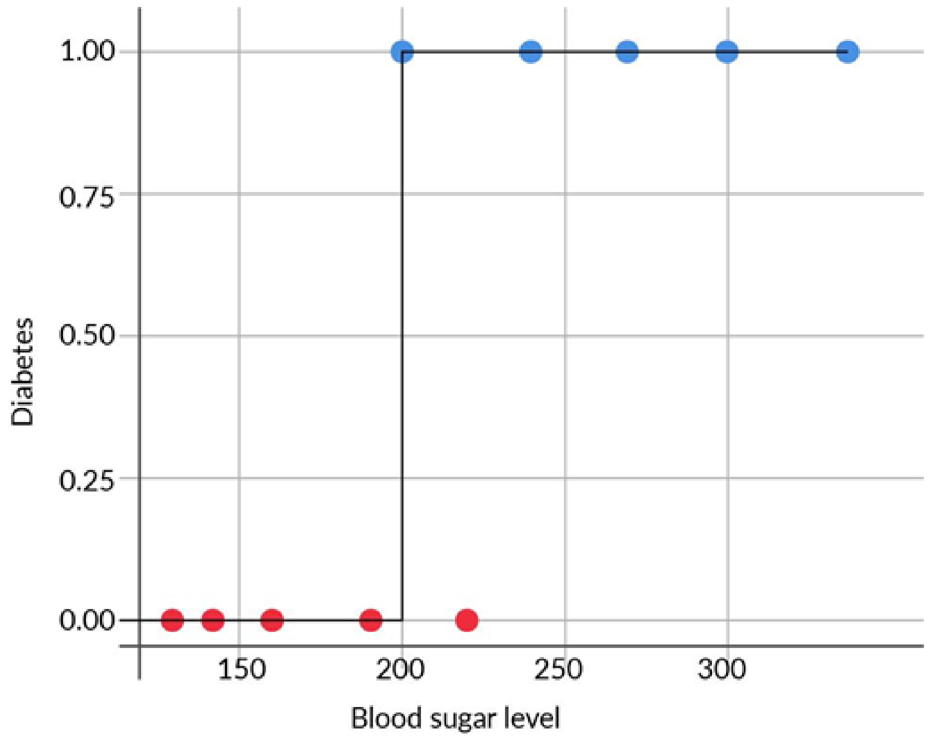
In the third session of the module, you learnt about logistic regression, a classification algorithm. You also got an understanding of the implementation of the logistic regression algorithm. Finally, you built a logistic regression model using the Spark ML library on the Click Through Rate (CTR) prediction data set.

Understanding Logistic Regression

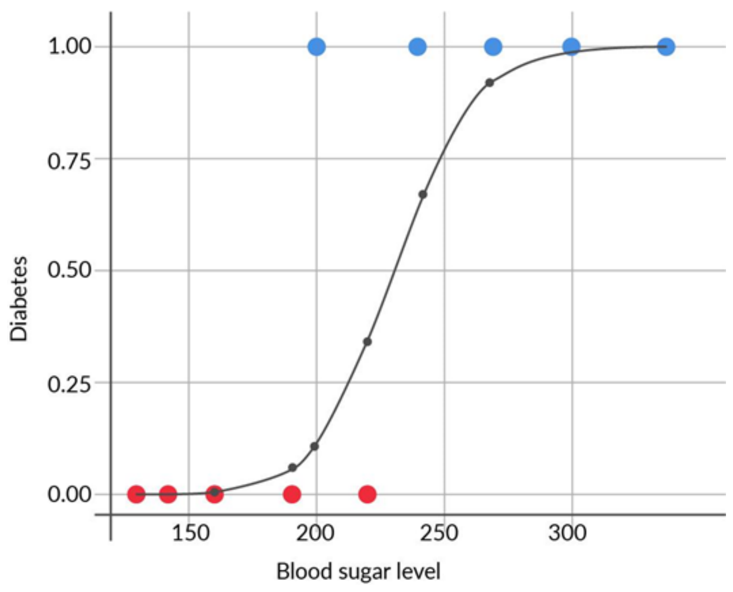
Even though the name implies '**Regression**', logistic regression is not used to solve problems pertaining to continuous variables; it is used to solve classification problems.
Now, classification problems can be of the following two types:

- **Binary:** In this problem, target variables have only two possible values.
- **Multiclass:** In this problem, target variables have more than two possible values.

For the diabetes classification problem, the hard decision boundary given below can be used to classify people based on whether they have diabetes or not.



However, this hard decision boundary would give erroneous results for borderline cases. So, to mitigate this issue, the smooth sigmoid curve can be used, as shown below. This curve gives the probability of a person having diabetes at any given blood sugar level.



The equation for the curve given above can be expressed as follows:

$$y\ (\text{Probability of Diabetes}) = \frac{1}{1+e^{-(\beta_0 + \beta_1 X)}}$$

Where,

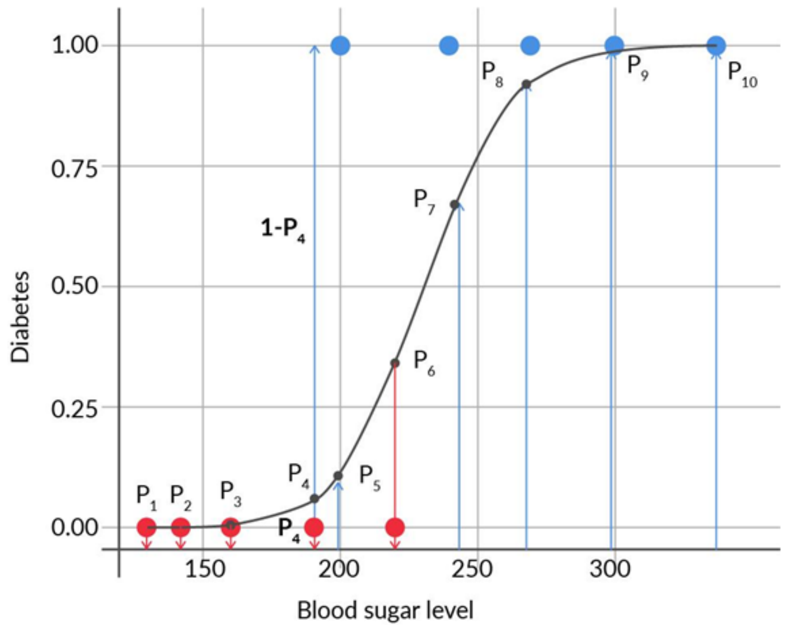
y = The probability a person having diabetes

x = Blood sugar level

Now, you can create a different sigmoid curve by changing the values of β_1 and β_0. In order to find the optimal combination of the values of β_1 and β_0 that would maximise the likelihood, you need to maximise the likelihood expression. The likelihood expression for the diabetes curve can be written as follows:

$$\text{Likelihood} = (1-P_1)(1-P_2)(1-P_3)(1-P_4)(P_5)(1-P_6)(P_7)(P_8)(P_9)(P_{10})$$

The likelihood expression can be derived from the graph given below.



Maximum Likelihood Estimation (MLE) is a method used to maximise the likelihood expression in order to find the optimal values of β_1 and β_0.

CTR Prediction

You got an understanding of the implementation of the logistic regression model using the CTR prediction case study. The CTR prediction is an important metric used to evaluate
, i.e., whether or not the user is clicking the advertisement shown on the website/app. CTR can be explained as follows:

CTR = Clicks/Impressions

Essentially, CTR is the ratio of users who click on the advertisement link with respect to the total number of times the advertisement is displayed.
The features of the data set that you worked on are as follows:

1. **id:** Ad identifier
2. **click:** Zero for no-click and one for click
3. **hour:** Format is in YYMMDDHH. So, if the user clicked on the link on 14091123, it means 23:00 on 11 September 2014 Coordinated Universal Time (UTC).
4. **C1:** Anonymised categorical variable
5. **banner_pos:** Describes the position of the Ad on the website
6. **site_id:** Unique identifier of different websites
7. **Site_domain:** Domain of the given website (For example: News, Sports, Web services, etc.)
8. **Site_category:** Category of the website
9. **app_id:** Unique identifier of different apps
10. **app_domain:** Domain of the given app
11. **app_category:** Category of the app
12. **device_id:** Unique identifier of the device on which the Ad is displayed
13. **device_ip:** IP address of the device
14. **device_model:** Model of the device being used
15. **device_type:** Type of device
16. **device_conn_type:** Source of the connection for the device
17. **C14-C21:** Anonymised categorical variables

Model Implementation

In the model implementation part, you covered the model building steps from scratch.

First, you selected the integer columns for model building because we could not draw any inference from the string columns. The following code was used for this:

```
df_selected = df.select(['click', 'C1', 'banner_pos', 'device_type',
'device_conn_type', 'C14', 'C15', 'C16', 'C17', 'C18', 'C19', 'C20', 'C21'])
```

Next, you selected the columns with fewer distinct values. You then one-hot encoded the columns and used the vector assembler. Next, you dumped the above steps into a pipeline to form the final DataFrame for training the model.

```
OHE = OneHotEncoderEstimator(inputCols=['C1', 'banner_pos', 'device_type',
'device_conn_type','C15', 'C16', 'C18'],outputCols=['C1_encoded',
'banner_pos_encoded', 'device_type_encoded','device_conn_type_encoded',
'C15_encoded','C16_encoded', 'C18_encoded'])

vec_assembler = VectorAssembler(inputCols=['C1_encoded',
'banner_pos_encoded', 'device_type_encoded',
'device_conn_type_encoded','C15_encoded', 'C16_encoded', 'C18_encoded'],
outputCol="features")

final_pipe = Pipeline(stages=[OHE, vec_assembler])
piped_data = final_pipe.fit(df_filtered).transform(df_filtered)
```

Next, you split the data into training data and test data. You then built the model using the following code:

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol='features', labelCol='label')
model = lr.fit(training)
```

After building the model, you evaluated its performance through the ROC and Precision-Recall curves. You evaluated the accuracy of the model using the following code:

```
summary = model.evaluate(test)
summary.accuracy
```

K-Means Clustering

In the previous session of this module, you learnt about K-means, a clustering algorithm. You also got an understanding of the implementation of the K-Means Clustering algorithm. Finally, you built a clustering model using the Spark ML library on the music industry data set.

Understanding Clustering

Clustering is the process of organising objects into groups whose members are similar in some way or have some underlying pattern to them. Unlike supervised learning, there is no target variable in unsupervised learning. Here, the aim is to organise objects into multiple segments based on their properties.

An ideal Clustering model should have the following properties:

1. A cluster with high intra-class similarity, i.e., all the objects in the same cluster are similar to each other

2. A set of clusters with low inter-class similarity, i.e., each cluster or group is clearly separated from the others, thereby reducing similarity

K-Means Clustering

The K-Means algorithm computes the centroid of a set of points in each iteration to create K clusters. In simple terms, the centroid of n points in the x-y plane is the point that is often referred to as the geometric centre of the n points.

Consider three points with the following coordinates: (x1, y1),(x2, y2) and (x3, y3). The centroid of these three points is the average of the x and y coordinates of the three points, i.e., the point with the coordinates $(\frac{x1+x2+x3}{3}, \frac{y1+y2+y3}{3})$.

Similarly, if you have n points with the coordinates (x1,y1),…,(xn,yn), then the formula (coordinates) of the centroid will be $(\frac{x1+x2+....+xn}{3}, \frac{y1+y2+....+yn}{3})$.

The K-Means algorithm uses the Euclidean distance as a measure to compute the distance between different points. The Euclidean distance between two points in a two-dimensional plane with coordinates the (p1, p2) and (q1, q2) is defined as $\sqrt{(p1 - q1)^2 + (p2 - q2)^2}$.

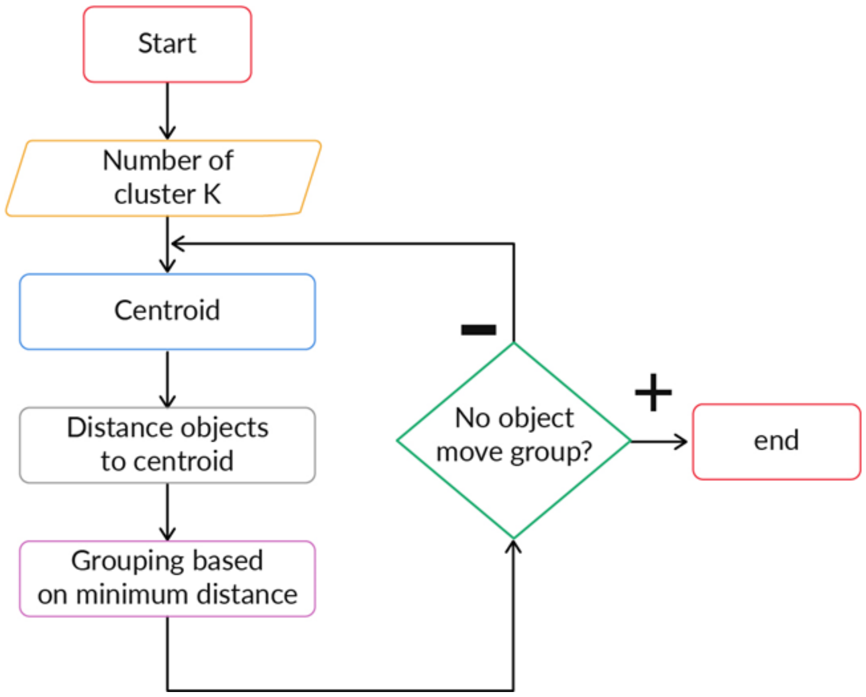
Similarly, if the points lie in the n-dimensional Euclidean space with the coordinates $(p1, ..., p_n)$ and $(q1,........qn)$, then the Euclidean distance between these two points will be as follows:

$$\sqrt{(p1 - q1)^2 + (p2 - q2)^2 +........ + (pn - qn)^2}$$

Now that you have understood the maths behind the K-Means algorithm, let's take a look at the steps involved in implementing the algorithm, which are as follows:

1. Select K, the number of clusters you want. Let's select K = 4.
2. Initialise K random points as the centroids of the initial clusters.
3. Measure the Euclidean distance between each data point and each centroid, and assign each data point to its closest centroid and corresponding cluster.
4. Recalculate the midpoint (centroid) of each cluster.
5. Repeat steps 3 and 4 to reassign data points to clusters based on the new centroid locations.
6. Stop the process when the centroids have been stabilised. After computing the centroid of a cluster, no data points are reassigned.

The flow chart of the steps mentioned above can be visualised as follows:



Start
Number of cluster K
Centroid
Distance objects to centroid
Grouping based on minimum distance
No object move group?
−
+
end

After the clusters are formed, the **loss function** is used to determine how good the clusters are. The loss function is expressed as follows:

$$J = \sum^{n}_{i=1} ||X_i - \mu_{k(i)}||^2$$

Where,

- **J** = The loss function
- $\mathbf{X_i}$ = The ith data point
- **In** $\boldsymbol{\mu_{k(i)}}$ = k(i) gives the cluster number corresponding to the ith data point, and $\mu_{k(i)}$ is the centroid associated with that data point

As evident from the equation, a **loss function** is the **sum of squared distances between each point and the associated cluster centre**. In order to form the best possible cluster, the loss function should be minimum.

Elbow Method

Another important parameter in the K-Means algorithm is the value of K, which indicates the number of clusters to be formed. The elbow curve is used to optimise the value of K. The elbow curve is plotted between K and the sum of squared distances, as shown below.

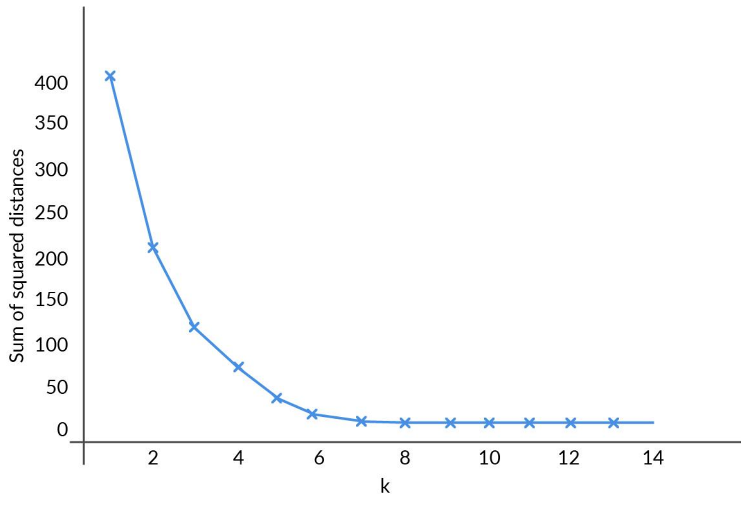


Fig3:Elbow Curve

As you can observe in the image given above, the cost function decreases rapidly with an increasing the value k. However, after reaching a certain value of k, the decrease in the cost function is not that prominent. So, the graph looks like an elbow, and hence, it is called the 'elbow curve'. In this example, k=6 is the optimal value of k, as the sum of squared distances decreases rapidly with an increasing k till k=6. However, after this point, the decrease is not that prominent. Therefore, in this case, the optimal value of k is 6.

Silhouette Coefficient Score

Another important parameter that is used to test the accuracy of the model is the **Silhouette Coefficient Score**. This score ranges from **-1 to 1**, where

- **k=1** indicates that clusters are well-separated and clearly distinguished.
- **k=0** implies the distance between clusters is not significant.
- **k=-1** means that clusters are assigned incorrectly.

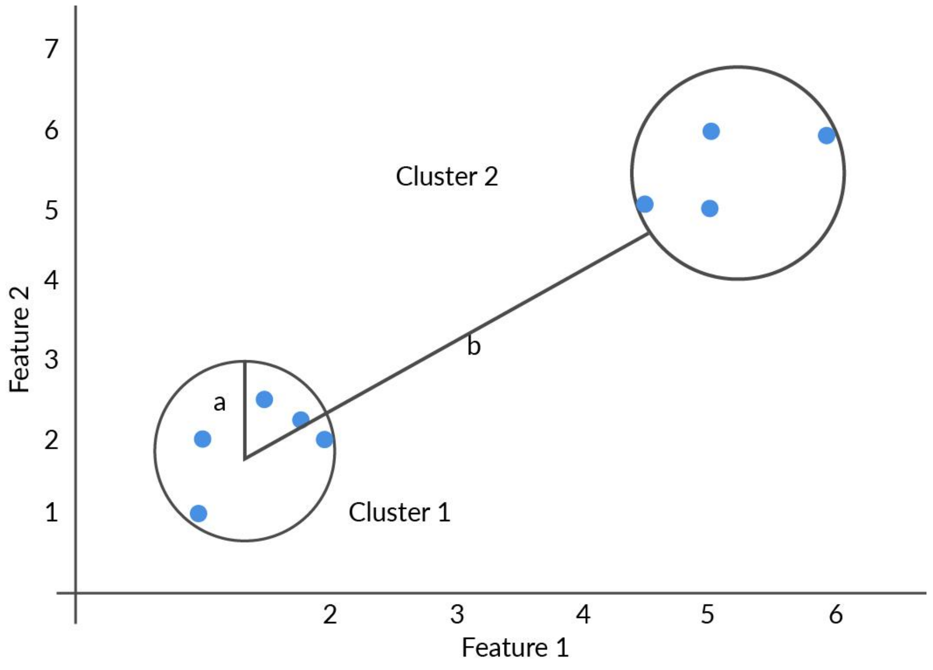
The Silhouette Coefficient Score can be calculated using the following formula:

$$\text{Silhouette Score} = (b - a)/\max(a, b)$$

Where,

- **a** = The average intra-cluster distance, i.e., the average distance between points within a cluster.
- **b** = The average inter-cluster distance, i.e., the average distance between all clusters**.**

The values of a and b can be visualised using the graph given below.



Model Building

For building a K-Means model, you first imported the K-Means model using the following code:

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
```

After importing the model, you first need to initialise the value of K. For this, you used the following code:

```
kmeans = KMeans(k=10, seed=1)
```

Next, you built the model using the following code:

```
model = kmeans.fit(model_data.select('features'))
```

Now after building the model, you appended the prediction column to the dataset containing the cluster ID of each data.
The next step is to evaluate the model. You calculated the Silhouette Score using the following code:

```
evaluator = ClusteringEvaluator()
silhouette_score = evaluator.evaluate(output)
print("silhouette_score = " + str(silhouette_score))
```

Next, you calculated the cost function for various K values using the following code:

```
ks = [3,7,10]
```

```
costfunction = []

for k_num in ks:
    # build kmeans model with k as no of cluster
    print("K :  ",k_num)
    model_k = KMeans(k=k_num , seed=1)

    # train the model
    model = model_k.fit(model_data.select('features'))

    # Append costfunction to list of costfunction
    costfunction.append(model.computeCost(model_data))
```

You then plotted the elbow curve using the following code:

```
import matplotlib.pyplot as plt
%matplotlib inline

# Plot k vs cost_function
plt.plot(ks, costfunction, '-o')
plt.xlabel('number of clusters, k')
plt.ylabel('cost function')
plt.xticks(ks)
plt.show()
```

Using this elbow curve, you determined the optimal K value and the Silhouette Coefficient Score, according to which the model performed well.