Shahajirao Patil Vikas Pratishthan's

# S. B. Patil College of Engineering, Indapur

## Department of Computer Engineering

**SEMESTER-I**

**[AY 2022-2023]**

# LABORATORY MANUAL
# Laboratory Practice III

Name of student-_____

Exam Seat No.-_____

**TEACHING SCHEME:**

Practical: 4Hrs/Week

**EXAMINATION SCHEME:**

**Practical** Assessment: ***50****Marks*

**Term work** Assessment: ***50*** *Marks*

Shahajirao Patil Vikas Pratishthan's

# S. B. Patil College of Engineering, Indapur

S. B. Patil College of Engineering

## CERTIFICATE

This is to certify that, _____**, Seat No.**_____ of first semester of BE in Computer Engineering has completed the term work satisfactorily in **410246:Laboratory Practice III** for academic year 2021-22 as prescribed in the curriculum by S.P. Pune University,Pune.

Place : Indapur
Date : Exam Seat No. : _____

**Subject Teacher** **Head of Department** **Principal**

# List of Experiments

| Sr. no | Group | Title | Page Nos |
|---|---|---|---|
| **410242: Machine Learning** | | | |
| 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.<br>Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset | |
| 2 | | 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.<br>Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv | |
| 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 | |
| 4 | | 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 | |
| | | | |

# 410242:Machine Learning

**Experiment 1**

| | |
|---|---|
| **Experiment No.** | 1A |
| **Title** | Implement Linear regression and randome foreset tree algorithm for Uber |
| **Seat No.** | |
| **Class** | B.E. (C.E.) |
| **Date** | |
| **Subject** | **Laboratory Practice III (ML)** |

**Experiment No:1A**

**Title:**

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.

**Objectives**:

- To learn the concept linear regression and randome forest tree
- To able to implement algorithm for Uber datasets

**Theory:**

**An introduction to linear regression**

In the most simple words, **Linear Regression** is the supervised Machine Learning model in which the **model finds the best fit linear line between the independent and dependent variable** i.e it finds the linear relationship between the dependent and independent variable.

Linear Regression is of two types: **Simple and Multiple**. **Simple Linear Regression** is where only one independent variable is present and the model has to find the linear relationship of it with the dependent variable

Whereas, In **Multiple Linear Regression** there are more than one independent variables for the model to find the relationship.

Equation of Simple Linear Regression, where $b_o$ is the intercept, $b_1$ is coefficient or slope, x is the independent variable and y is the dependent variable.

$$y = b_o + b_1 x$$

Equation of Multiple Linear Regression, where bo is the intercept, $b_1, b_2, b_3, b_4 \ldots, b_n$ are coefficients or slopes of the independent variables $x_1, x_2, x_3, x_4 \ldots, x_n$ and y is the dependent variable.

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

**A Linear Regression model's main aim is to find the best fit linear line and the optimal values of intercept and coefficients such that the error is minimized.**

Error is the difference between the actual value and Predicted value and the goal is to reduce this difference.

Let's understand this with the help of a diagram.

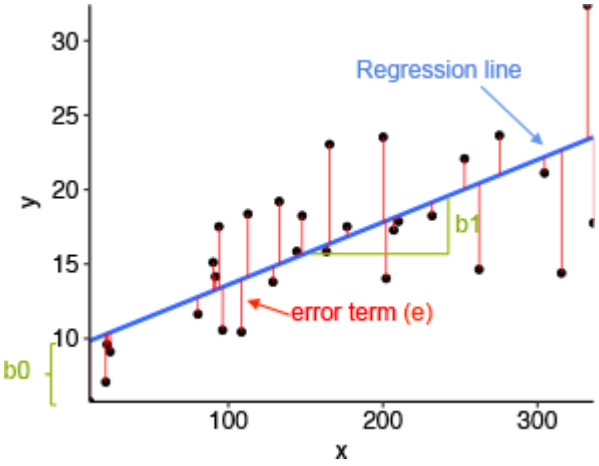



Image Source: Statistical tools for high-throughput data analysis

In the above diagram,

- x is our dependent variable which is plotted on the x-axis and y is the dependent variable which is plotted on the y-axis.
- Black dots are the data points i.e the actual values.
- $b_o$ is the intercept which is 10 and $b_1$ is the slope of the x variable.
- The blue line is the best fit line predicted by the model i.e the predicted values lie on the blue line.

**The vertical distance between the data point and the regression line is known as error or residual.** Each data point has one residual and the sum of all the differences is known as **the Sum of Residuals/Errors.**

**Mathematical Approach:**

Residual/Error = Actual values – Predicted Values

Sum of Residuals/Errors = Sum(Actual- Predicted Values)

Square of Sum of Residuals/Errors = (Sum(Actual- Predicted Values))$^2$

i.e

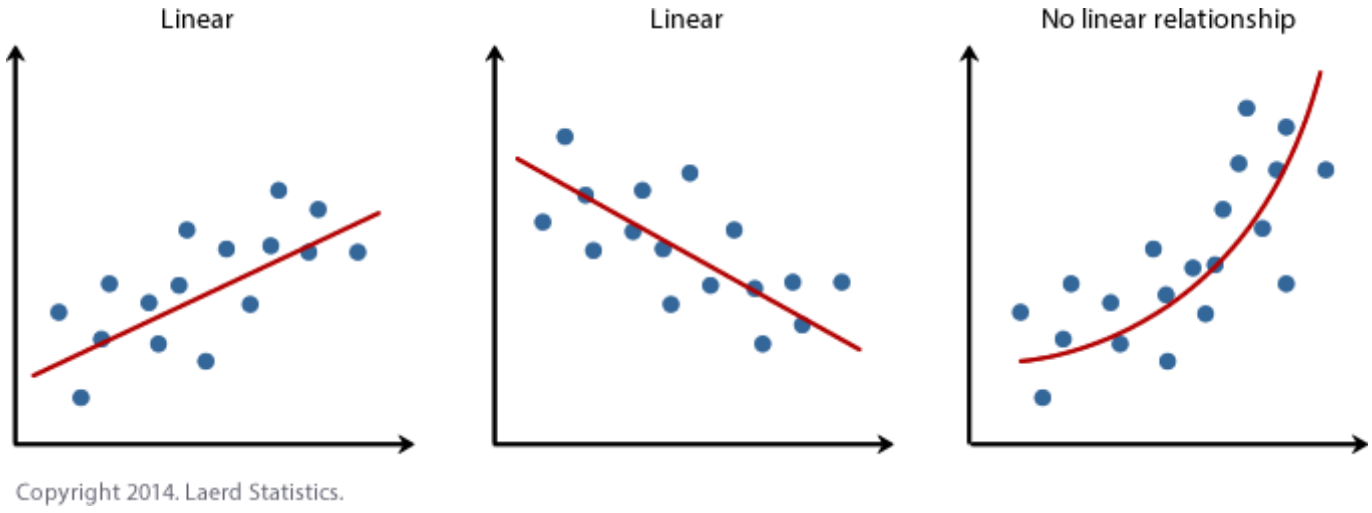
$$\sum e_i^2 = \sum (Y_i - \hat{Y}_i)^2$$

For an in-depth understanding of the Maths behind Linear Regression, please refer to the attached video explanation.

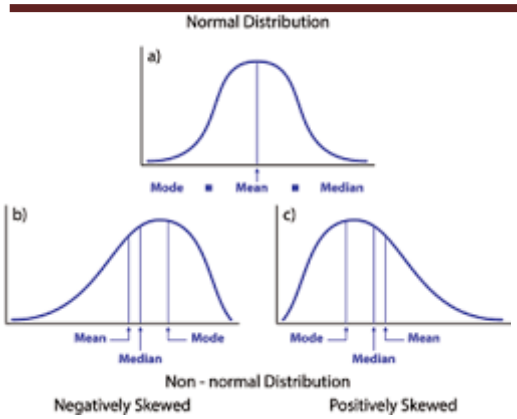
## Assumptions of Linear Regression

The basic assumptions of Linear Regression are as follows:

**1. Linearity**: It states that the dependent variable Y should be linearly related to independent variables. This assumption can be checked by plotting a scatter plot between both variables.

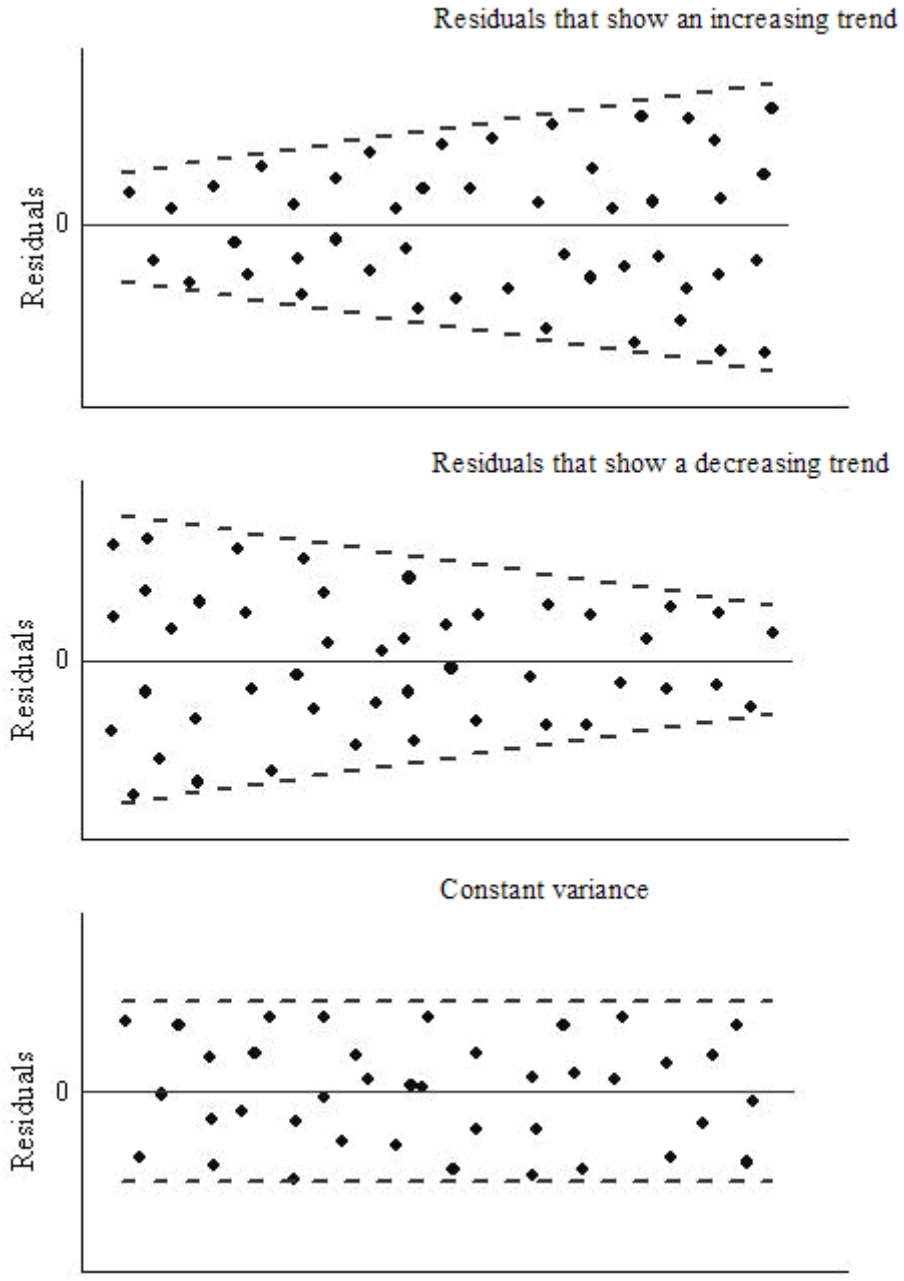



**2. Normality**: The X and Y variables should be normally distributed. Histograms, KDE plots, Q-Q plots can be used to check the Normality assumption.




Source: https://heljves.com/gallery/vol_1_issue_1_2019_8.pdf

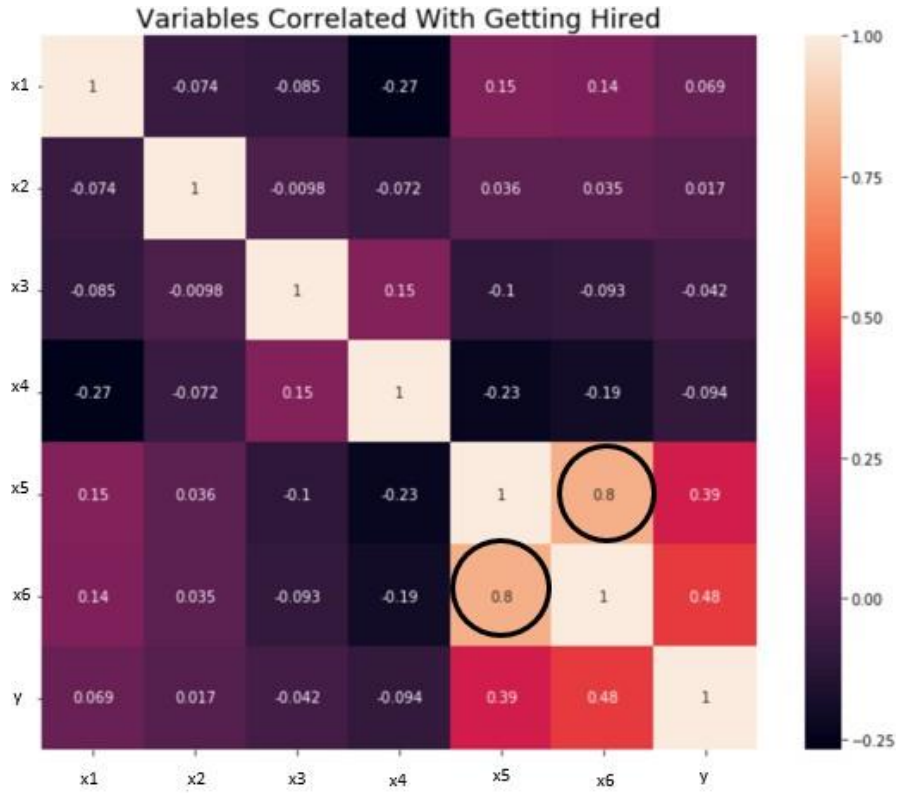
**3. Homoscedasticity**: The variance of the error terms should be constant i.e the spread of residuals should be constant for all values of X. This assumption can be checked by plotting a residual plot. If the assumption is violated then the points will form a funnel shape otherwise they will be constant.




Source: OriginLab

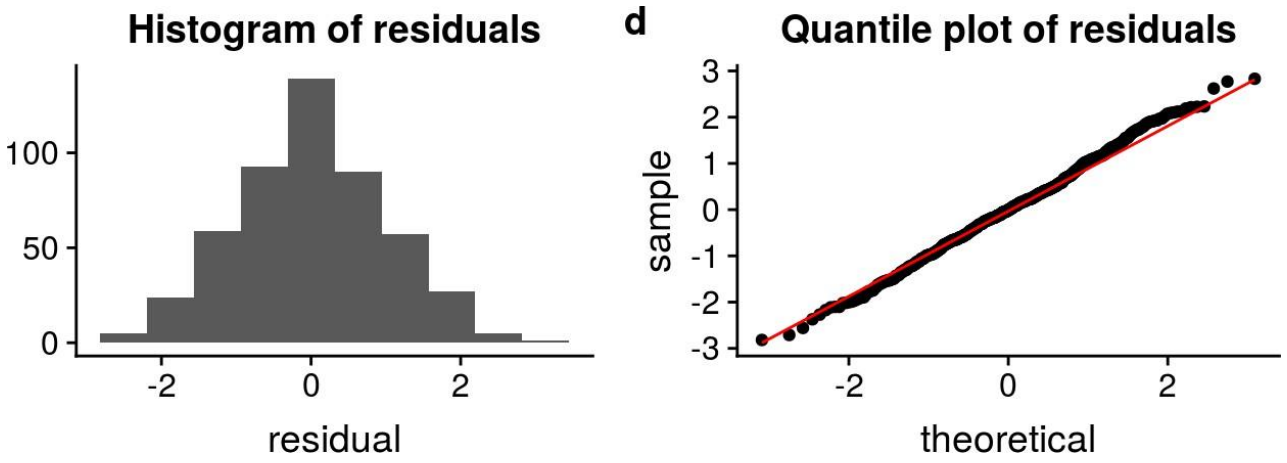
**4. Independence/No Multicollinearity**: The variables should be independent of each other i.e no correlation should be there between the independent variables. To check the assumption, we can use a correlation matrix or VIF score. If the VIF score is greater than 5 then the variables are highly correlated.

In the below image, a high correlation is present between x5 and x6 variables.



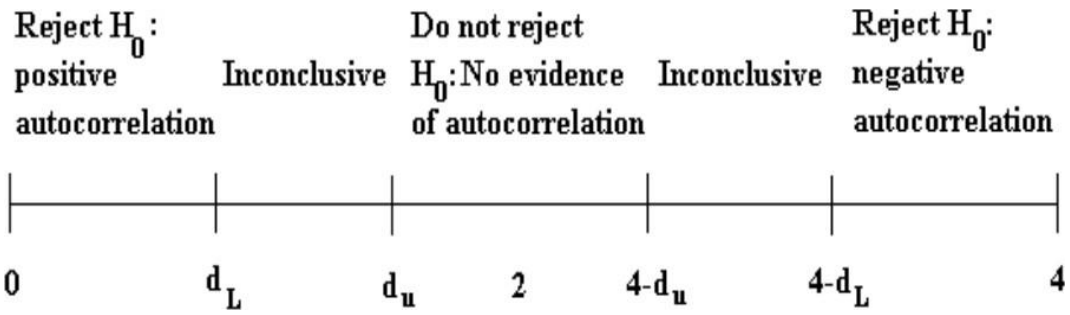

Source: towards data science

**5.** The **error terms should be normally distributed**. Q-Q plots and Histograms can be used to check the distribution of error terms.




Source: http://rstudio-pubs-static.s3.amazonaws.com

**6. No Autocorrelation:** The error terms should be independent of each other. Autocorrelation can be tested using the Durbin Watson test. The null hypothesis assumes that there is no autocorrelation. The value of the test lies between 0 to 4. If the value of the test is 2 then there is no autocorrelation.




Source: itfeature.com

### How to deal with the Violation of any of the Assumption

The Violation of the assumptions leads to a decrease in the accuracy of the model therefore the predictions are not accurate and error is also high.

**For example,** if the Independence assumption is violated then the relationship between the independent and dependent variable can not be determined precisely.

There are various methods are techniques available to deal with the violation of the assumptions. Let's discuss some of them below.

**Violation of Normality assumption of variables or error terms**

To treat this problem, we can transform the variables to the normal distribution using various transformation functions such as log transformation, Reciprocal, or Box-Cox Transformation.

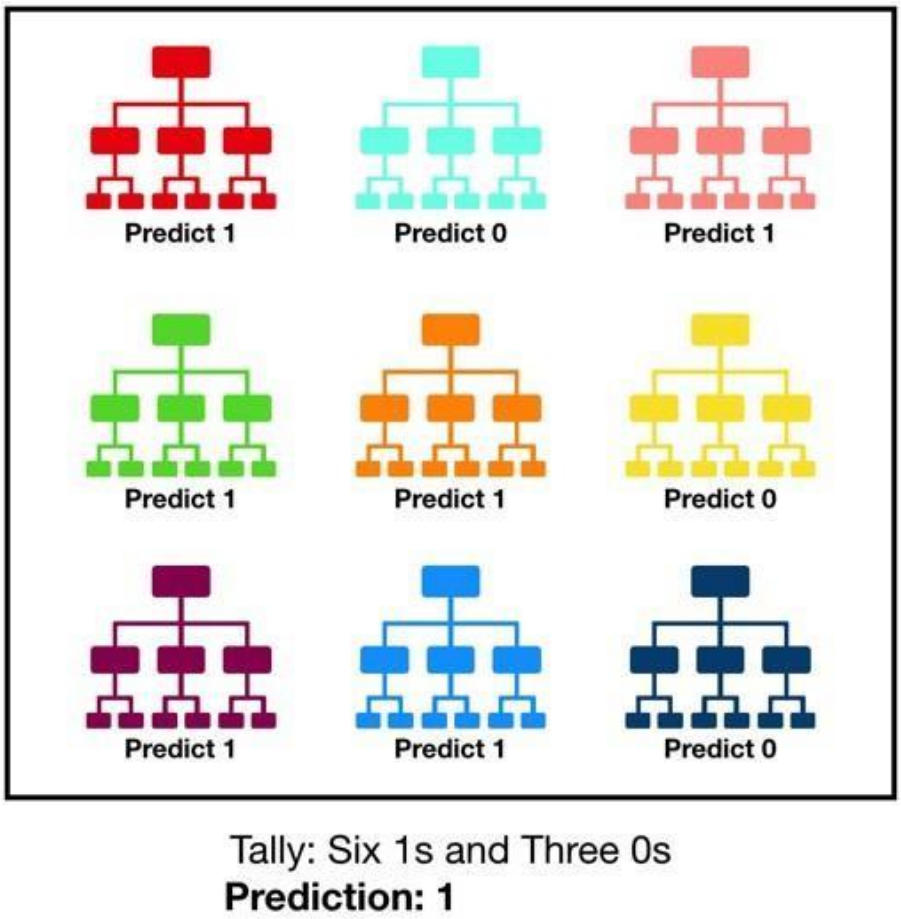
**Violation of MultiCollineraity Assumption**

It can be dealt with by:

- Doing nothing (if there is no major difference in the accuracy)
- Removing some of the highly correlated independent variables.
- Deriving a new feature by linearly combining the independent variables, such as adding them together or performing some mathematical operation.
- Performing an analysis designed for highly correlated variables, such as principal components analysis.

### Random Forest tree Introduction

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below).




Visualization of a Random Forest Model Making a Prediction

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:

***A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.***

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that

is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. **The reason for this wonderful effect is that the trees protect each other from their individual errors** (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

The wonderful effects of having many uncorrelated models is such a critical concept that I want to show you an example to help it really sink in. Imagine that we are playing the following game:

- I use a uniformly distributed random number generator to produce a number.
- If the number I generate is greater than or equal to 40, you win (so you have a 60% chance of victory) and I pay you some money. If it is below 40, I win and you pay me the same amount.
- Now I offer you the the following choices. We can either:

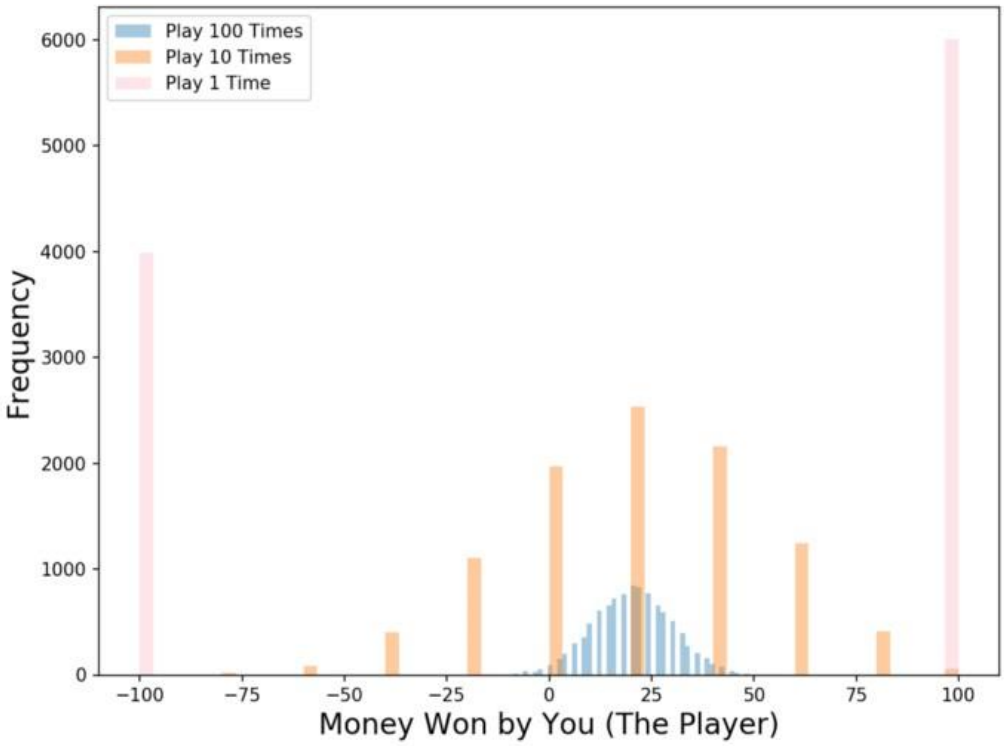
1. **Game 1** — play 100 times, betting $1 each time.
2. **Game 2**— play 10 times, betting $10 each time.
3. **Game 3**— play one time, betting $100.

Which would you pick? The expected value of each game is the same:

*Expected Value Game 1 = (0.60*1 + 0.40*-1)*100 = 20*

*Expected Value Game 2= (0.60*10 + 0.40*-10)*10 = 20*

*Expected Value Game 3= 0.60*100 + 0.40*-100 = 20*

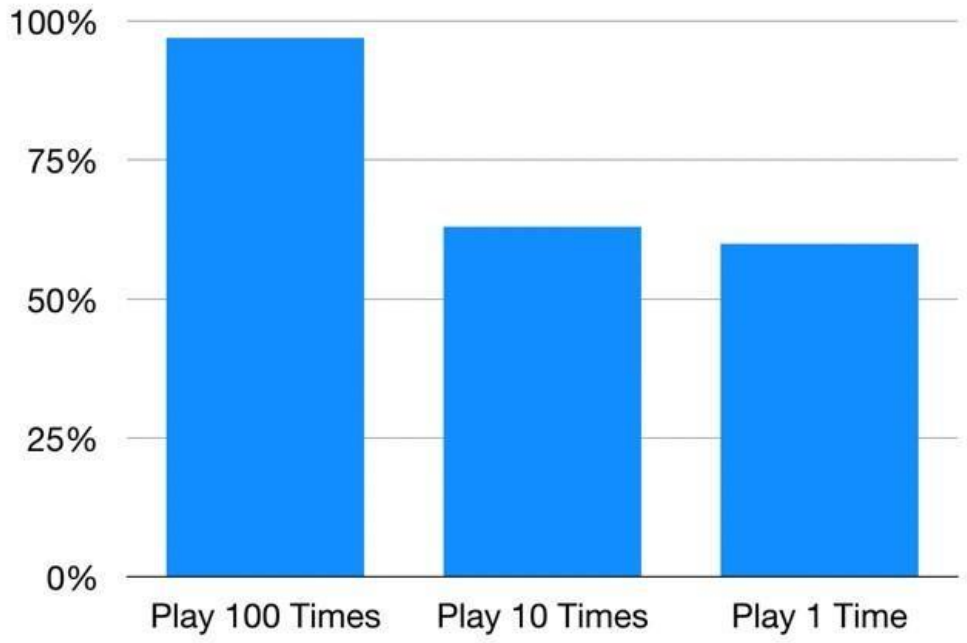



Outcome Distribution of 10,000 Simulations for each Game

What about the distributions? Let's visualize the results with a Monte Carlo simulation (we will run 10,000 simulations of each game type; **for example, we will simulate 10,000 times the 100 plays of Game 1**). Take a look at the chart on the left — now which game would you pick? Even though the expected values are the same, **the outcome distributions are vastly different going from positive and narrow (blue) to binary (pink).**

Game 1 (where we play 100 times) offers up the best chance of making some money — **out of the 10,000 simulations that I ran, you make money in 97% of them!** For Game 2 (where we play 10 times) you make money in 63% of the simulations, a drastic decline (and a drastic increase in your probability

of losing money). And Game 3 that we only play once, you make money in 60% of the simulations, as expected.




Probability of Making Money for Each Game

So even though the games share the same expected value, their outcome distributions are completely different. The more we split up our $100 bet into different plays, the more confident we can be that we will make money. As mentioned previously, this works because each play is independent of the other ones.

Random forest is the same — each tree is like one play in our game earlier. We just saw how our chances of making money increased the more times we played. Similarly, with a random forest model, our chances of making correct predictions increase with the number of uncorrelated trees in our model.

## Evaluation Metrics for Regression Analysis

To understand the performance of the Regression model performing model evaluation is necessary. Some of the Evaluation metrics used for Regression analysis are:

1. **R squared or Coefficient of Determination:** The most commonly used metric for model evaluation in regression analysis is R squared. It can be defined as a Ratio of variation to the Total Variation. The value of R squared lies between 0 to 1, the value closer to 1 the better the model.

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

where SSRES is the Residual Sum of squares and SSTOT is the Total Sum of squares

2. **Adjusted R squared:** It is the improvement to R squared. The problem/drawback with R2 is that as the features increase, the value of R2 also increases which gives the illusion of a good model. So the Adjusted R2 solves the drawback of R2. It only considers the features which are important for the model and shows the real improvement of the model.
Adjusted R2 is always lower than R2.

$$R^2_{adjusted} = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

where
$R^2$ = sample R-square
p = Number of predictors
N = Total sample size.

3. **Mean Squared Error (MSE)**: Another Common metric for evaluation is Mean squared error which is the mean of the squared difference of actual vs predicted values.

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

4. **Root Mean Squared Error (RMSE)**: It is the root of MSE i.e Root of the mean difference of Actual and Predicted values. RMSE penalizes the large errors whereas MSE doesn't.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

**Implemetation:**

Step 1: Reading and Understanding the Data

Step 2: Data Cleansing

Step 3: Data Preparation

Step 4: Model Building

Step 5: Final Analysis

**Dataset-** uber.csv

**Standard Libraries used are**

```
import numpy as np
import pandas as pd
from datetime import datetime
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

```
from sklearn.ensemble import RandomForestRegressor
```

Final results :

Linear Regression

#mean absolute error

```
print(metrics.mean_absolute_error(y_test,y_pred))
```

```
0.2665498584386339
```

#mean squared error

```
print(metrics.mean_squared_error(y_test,y_pred))
```

```
0.2356214239077307
```

#root mean squared error

```
print(np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

```
0.48540851239727006
```

Random Forest Tree

```
prediction = rf.predict(X_test)
mse = metrics.mean_squared_error(y_test, prediction)
rmse = mse**.5
print(mse)
```

```
0.22952269151771001
```

```
print(rmse)
```

```
0.4790852653940737
```

**Conclusion**: we are able to implement linear regression and random forest tree algorithm for Uber ride data set.

| | |
|---|---|
| **Experiment No.** | 2 |
| **Title** | Implement K nearest neigbour and SVM algorithm for spam email or not |
| **Seat No.** | |
| **Class** | B.E. (C.E.) |
| **Date** | |
| **Subject** | **Laboratory Practice III (ML)** |

**Experiment No: 1B**

**Title:** Implement K nearest neigbour and SVM algorithm for spam email or not

**Objectives:**

- To learn the concept of K nearest neigbour and SVM algorithm
- To able resolved problem for sapam email or not.

## Introduction

## K-nearest neigbour

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. Pause! Let us unpack that.

A **supervised machine learning** algorithm (as opposed to an unsupervised machine learning algorithm) is one that relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data.

Imagine a computer is a child, we are its supervisor (e.g. parent, guardian, or teacher), and we want the child (computer) to learn what a pig looks like. We will show the child several different pictures, some of which are pigs and the rest could be pictures of anything (cats, dogs, etc).

When we see a pig, we shout "pig!" When it's not a pig, we shout "no, not pig!" After doing this several times with the child, we show them a picture and ask "pig?" and they will correctly (most of the time) say "pig!" or "no, not pig!" depending on what the picture is. That is supervised machine learning.

Supervised machine learning algorithms are used to solve classification or regression problems.

A **classification problem** has a discrete value as its output. For example, ―likes pineapple on pizza‖ and ―does not like pineapple on pizza‖ are discrete. There is no middle ground. The analogy above of teaching a child to identify a pig is another example of a classification problem.

| Age | Likes Pinapple on Pizza |
|---|---|
| 42 | 1 |
| 65 | 1 |
| 50 | 1 |
| 76 | 1 |
| 96 | 1 |
| 50 | 1 |
| 91 | 0 |
| 58 | 1 |
| 25 | 1 |
| 23 | 1 |
| 75 | 1 |
| 46 | 0 |
| 87 | 0 |
| 96 | 0 |
| 45 | 0 |
| 32 | 1 |
| 63 | 0 |
| 21 | 1 |
| 26 | 1 |
| 93 | 0 |
| 68 | 1 |
| 96 | 0 |

Image showing randomly generated data

This image shows a basic example of what classification data might look like. We have a predictor (or set of predictors) and a label. In the image, we might be trying to predict whether someone likes pineapple (1) on their pizza or not (0) based on their age (the predictor).

It is standard practice to represent the output (label) of a classification algorithm as an integer number such as 1, -1, or 0. In this instance, these numbers are purely representational. Mathematical operations should not be performed on them because doing so would be meaningless. Think for a moment. What is ―likes pineapple‖ + ―does not like pineapple‖ ? Exactly. We cannot add them, so we should not add their numeric representations.

A **regression problem** has a real number (a number with a decimal point) as its output. For example, we could use the data in the table below to estimate someone's weight given their height.

| Height(Inches) | Weight(Pounds) |
|---|---|
| 65.78 | 112.99 |
| 71.52 | 136.49 |
| 69.40 | 153.03 |
| 68.22 | 142.34 |
| 67.79 | 144.30 |
| 68.70 | 123.30 |
| 69.80 | 141.49 |
| 70.01 | 136.46 |
| 67.90 | 112.37 |
| 66.78 | 120.67 |
| 66.49 | 127.45 |
| 67.62 | 114.14 |
| 68.30 | 125.61 |
| 67.12 | 122.46 |
| 68.28 | 116.09 |

Image showing a portion of the SOCR height and weights data set

Data used in a regression analysis will look similar to the data shown in the image above. We have an independent variable (or set of independent variables) and a dependent variable (the thing we are trying to guess given our independent variables). For instance, we could say height is the independent variable and weight is the dependent variable.

Also, each row is typically called an **example, observation, or data point**, while each column (not including the label/dependent variable) is often called a **predictor, dimension, independent variable, or feature.**

An **unsupervised machine learning** algorithm makes use of input data without any labels —in other words, no teacher (label) telling the child (computer) when it is right or when it has made a mistake so that it can self-correct.

Unlike supervised learning that tries to learn a function that will allow us to make predictions given some new unlabeled data, unsupervised learning tries to learn the basic structure of the data to give us more insight into the data.

## KNN Algorithm

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

"Birds of a feather flock together."

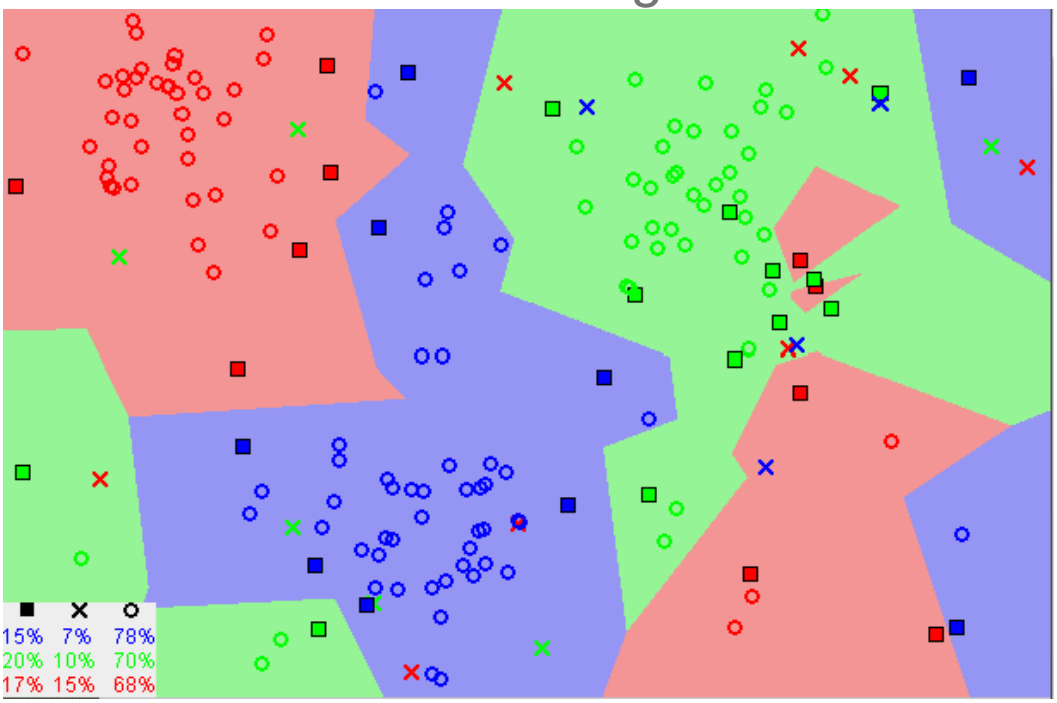



**Image showing how similar data points typically exist close to each other**

Notice in the image above that most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

## The KNN Algorithm

1. Load the data
2. Initialize K to your chosen number of neighbors

3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

5. Pick the first K entries from the sorted collection

6. Get the labels of the selected K entries

7. If regression, return the mean of the K labels

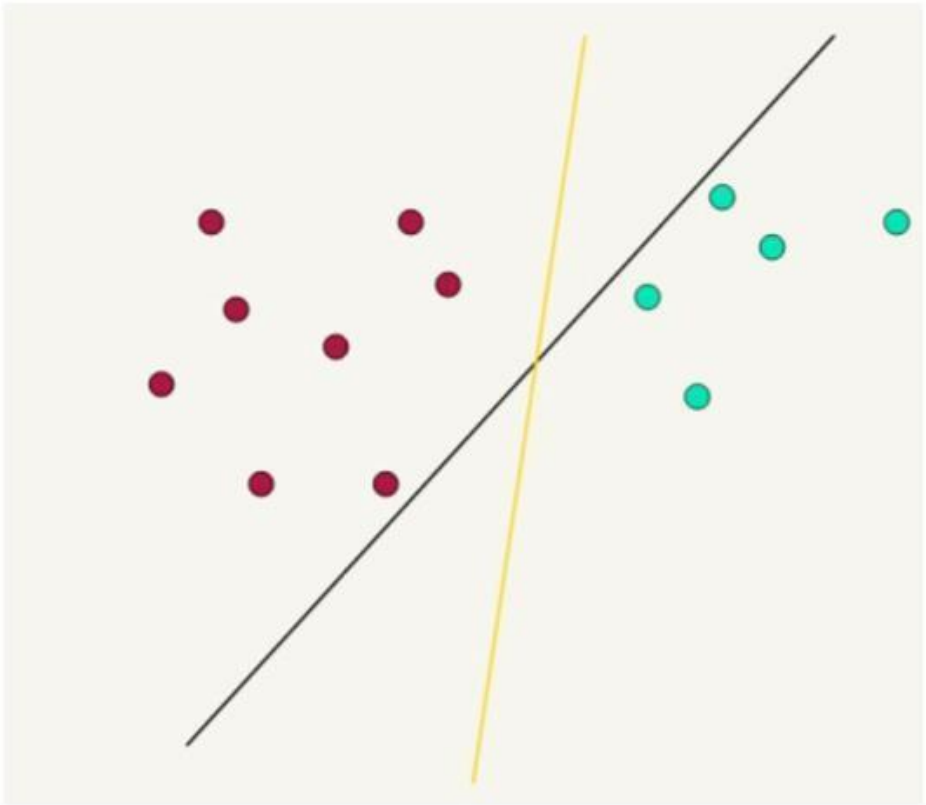
8. If classification, return the mode of the K labels

**Support Vector Machine for classification:**

Support Vector Machine (SVM) is probably one of the most popular ML algorithms used by data scientists. SVM is powerful, easy to explain, and generalizes well in many cases. In this article, I'll explain the rationales behind SVM and show the implementation in Python. For simplicity, I'll focus on binary classification problems in this article. However, SVM supports multi-classification.

## General Ideas Behind SVM

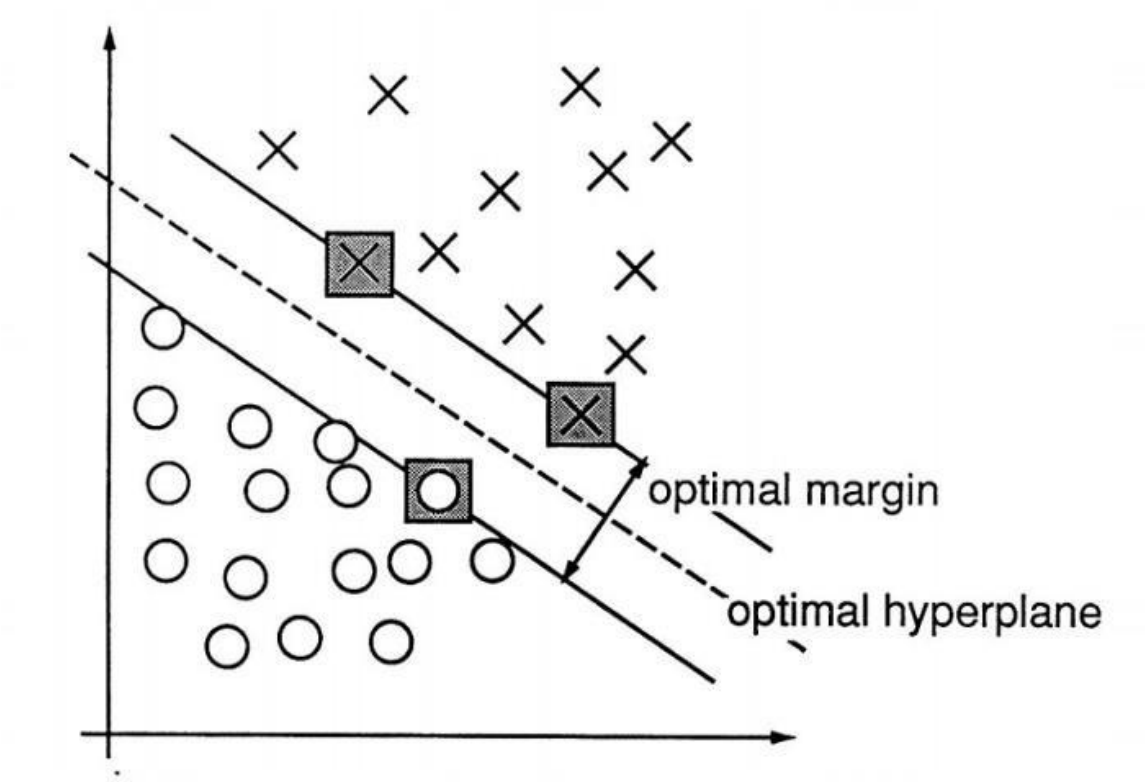
SVM seeks the best decision boundary which separates two classes with the highest generalization ability (why focus on generalization? I'll explain it later in the kernel section). The first question one asks is how does SVM define optimality.



Which decision boundary should we pick?

Unlike logistic regression, which defines optimality by overall probability, SVM wants the **smallest distance between data points and the decision**

**boundary to be as large as possible**. In other words, if you imagine the decision boundary as the central line of a street, SVM prefers an 8-line highway rather than a country road. The width of the street is called the **margin**.




## Math

## Define Margin

Okay, the idea is pretty intuitive. Let's find out the expression for margin. Before we get started, please remember the symbols used in this article:

$$x: N \times K \rightarrow N \text{ data points with } K \text{ features}, y: N \times 1 \rightarrow \text{labels. The data space has } K \text{ dimension}$$

$$W: K \times 1 \rightarrow \text{weights of the decision boundary}, b: \text{intercept of the boundary}$$

If we know the weights and intercepts of the decision boundary, the boundary can be expressed by the following equation:

$$Boundary: W_1X_1 + W_2X_2 + \cdots + W_KX_k + b = 0 \rightarrow X * W + b = 0$$

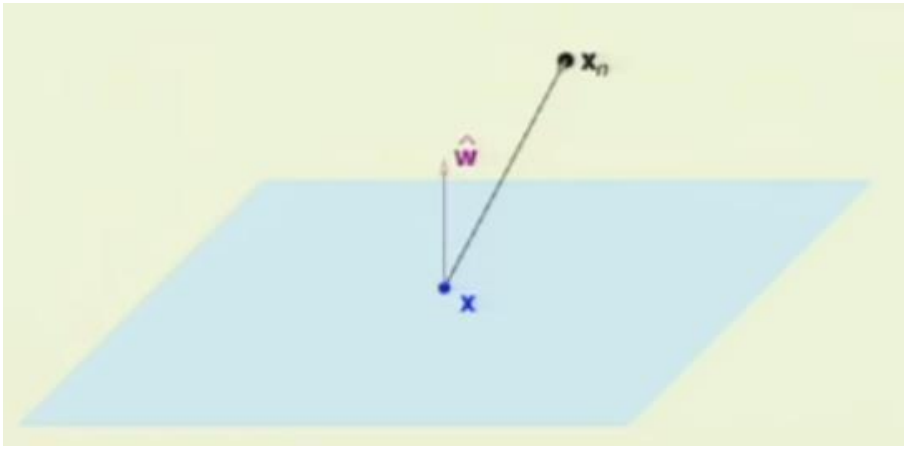
The equation for the broken line in the last figure

Distance from a data point to the boundary is:

$x_p$ *is any point on the boundary,* $x_d$ *is a data point.* $\vec{x}_d$ *is the vector from the orgin to* $x_d$

$$dist_{x_n} = \left|(\vec{x}_n - \vec{x}_p) * \frac{\vec{W}}{\|\vec{W}\|}\right| (projection\ onto\ \hat{W}) = \left|(\vec{x}_n * \vec{W} - \vec{x}_p * \vec{W}) * \frac{1}{\|\vec{W}\|}\right|$$

$$= \left|(\vec{x}_n * \vec{W} + b - \vec{x}_p * \vec{W} - b) * \frac{1}{\|\vec{W}\|}\right| = \left|(\vec{x}_n * \vec{W} + b) * \frac{1}{\|\vec{W}\|}\right| (since\ \vec{x}_p * \vec{W} + b = 0)$$




The margin is the distance from the closet point to the boundary:

$$Margin = \min_{i=1,2,\dots,N} dist_{x_i} = \min_{i=1,2,\dots,N} \left|\frac{(\vec{x}_i * \vec{W} + b)}{\|\vec{W}\|}\right|$$

## Final Objective

Now we know how to calculate the margin, let‘s try to formalize the problem which needs to be optimized:

$$If\ we\ \boldsymbol{define} \min_{i=1,2,\dots,N} |(\vec{x}_i * \vec{W} + b)| = 1, then\ Margin = \frac{1}{\|\vec{W}\|}\ and\ |(\vec{x}_i * \vec{W} + b)| \geq 1 \rightarrow$$

$$y_i(\vec{x}_i * \vec{W} + b) \geq 1\ since\ y \in \{1, -1\}$$

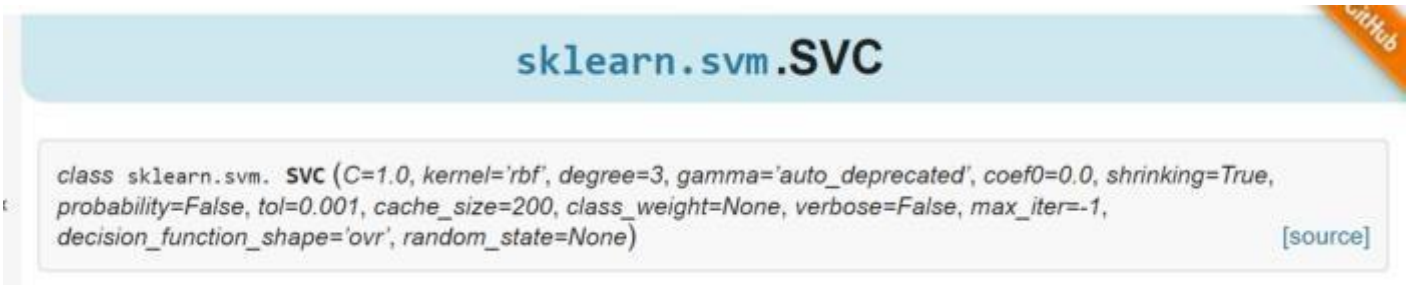
$$We\ want\ to\ \boldsymbol{maximize}\ \frac{1}{\|\vec{W}\|}\ \boldsymbol{subject\ to}: y_i(\vec{x}_i * \vec{W} + b) - 1 \geq 0\ \ \forall i \in x$$

Final Objective

I have to admit this is a weird function. Just remember this: it is like the RMSE for linear regression, cross-entropy for logistic regression, a function that needs to be optimized.

## Python Implementation

Now we understand how SVM works, let's try the svm.SVC function offered by Scikit-Learn




Wait a minute, what are these parameters? There's no C or _kernel' in the objective. Are you sure it is the right function? Unfortunately yes, we just need to dig a little deeper into the math behind SVM.

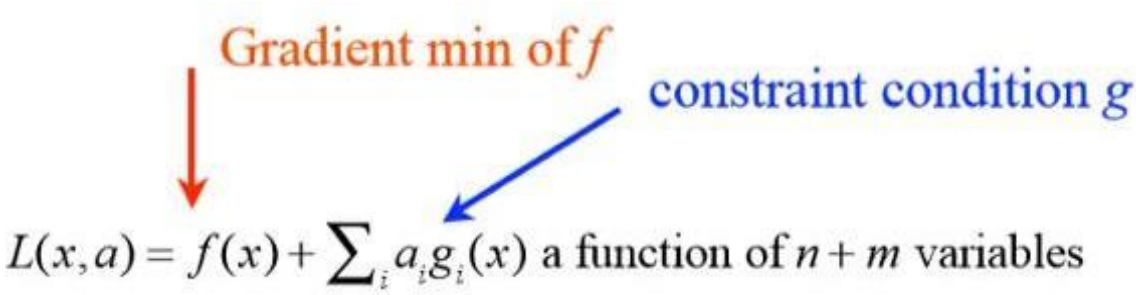
Now we need to talk about how to optimize the objective function. Recall that the function is:

$$\textit{maximize } \frac{1}{\|\vec{w}\|} \textit{ subject to: } y_i\left(\vec{x}_i * \vec{W} + b\right) - 1 \geq 0 \;\; \forall i \in x$$

How on earth do we maximize this equation? We can't just take derivatives and be done with it since the final W and b may not satisfy the constraints. Luckily a guy from high school math class comes to rescue — Lagrange.

**Eliminate the constrains**

*The **method of Lagrange multipliers** is a strategy for finding the local maxima and minima of a function subject to equality constraints (i.e., subject to the condition that one or more equations have to be satisfied exactly by the chosen values of the variables).*
*the derivative test of an unconstrained problem can still be applied — Wikipedia*




With the help of Lagrangian multipliers, we can find the solution for the question above by solving this function:

$$Minimize\ L(W,b,\alpha) = \frac{1}{2}\vec{W}^T\vec{W} - \sum_{i=1}^{N}\alpha_i(y_i(\vec{x}_i\vec{W}+b)-1)$$

$$(\max \frac{1}{\|\vec{w}\|} = \min \frac{1}{2}\vec{W}^T\vec{W}\ ,the\ second\ part\ is\ constraints, \alpha_i = 0\ if\ y_i(\vec{x}_i * \vec{W}+b)-1>0)$$

Unconstrained objective function

## Figure out W and b

Right now, W, b, and alpha are all unknown. Oh God, what do we do? Luckily Vapnik Vladimir figured out this part for us:

Minimize L with respect to W and b first (pretend we know alpha)

$$\frac{\partial L}{\partial W} = \vec{W} - \sum_{i=1}^{N}\alpha_i y_i \vec{x}_i = 0, \vec{W} = \sum_{i=1}^{N}\alpha_i y_i \vec{x}_i\ ; \quad \frac{\partial L}{\partial b} = -\sum_{i=1}^{N}\alpha_i y_i = 0$$

W and b are easy to find as long as we know alpha

Maximize L w.r.t alphas with W and b we just found.

$$Maximize\ \boldsymbol{L(\alpha)} = \sum_{i=1}^{N}\boldsymbol{\alpha_i} - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\boldsymbol{y_i y_j \alpha_i \alpha_j \vec{x}_i \vec{x}_j^T}\ \ subject\ to\ \sum_{i=1}^{N}\boldsymbol{\alpha_i y_i} = \mathbf{0}\ (beacuse\ b=0)$$

$$Which\ can\ be\ rewritten\ as\ \max_{\boldsymbol{\alpha}}\ \boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha^T Q \alpha}\ , Q\ is\ determined\ by\ y, \vec{x}$$

Wait, why all of a sudden does this becomes maximization? This is called the Lagrangian Duality.

The above function has a nice form, which can be solved by Quadratic Programming software. Just pass this function and the constraints to any commercial QP software, it'll return a list of alphas. Now with alpha solved, W and b can be calculated. Done!

**Implemetation:**

Step 1: Reading and Understanding the Data

Step 2: Data Cleansing

Step 3: Data Preparation

Step 4: Model Building

Step 5: Final Analysis

**Dataset-** email.csv

**Standard Libraries used are**

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC
```

**Final results**

**Score K-nearest Neighbour**

```
0.8556701030927835
```

**Score SVM**

```
0.9664948453608248
```

**Conclusion:** We are able to implement K-nearest neigbour and SVM for Email spam

| | |
|---|---|
| **Experiment No.** | 3 |
| **Title** | Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall. |
| **Seat No.** | |
| **Class** | B.E. (C.E.) |
| **Date** | |
| **Subject** | **Laboratory Practice I (HPL)** |

**Experiment No:2**

**Title:** Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall.

**Objective:**

- To learn the concept of K-Nearest Neighbors algorithm
- To able apply it on diabetes.csv.

**Theory:**

K-Nearest Neighbors algorithm-

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. Pause! Let us unpack that.

A **supervised machine learning** algorithm (as opposed to an unsupervised machine learning algorithm) is one that relies on labeled input data to learn a function that produces an appropriate output when given new unlabeled data.

Imagine a computer is a child, we are its supervisor (e.g. parent, guardian, or teacher), and we want the child (computer) to learn what a pig looks like. We will show the child several different pictures, some of which are pigs and the rest could be pictures of anything (cats, dogs, etc).

When we see a pig, we shout "pig!" When it's not a pig, we shout "no, not pig!" After doing this several times with the child, we show them a picture and ask "pig?" and they will correctly (most of the time) say "pig!" or "no, not pig!" depending on what the picture is. That is supervised machine learning.

Supervised machine learning algorithms are used to solve classification or regression problems.

A **classification problem** has a discrete value as its output. For example, ―likes pineapple on pizza‖ and ―does not like pineapple on pizza‖ are discrete. There is no middle ground. The analogy above of teaching a child to identify a pig is another example of a classification problem.

| Age | Likes Pinapple on Pizza |
|---|---|
| 42 | 1 |
| 65 | 1 |
| 50 | 1 |
| 76 | 1 |
| 96 | 1 |
| 50 | 1 |
| 91 | 0 |
| 58 | 1 |
| 25 | 1 |
| 23 | 1 |
| 75 | 1 |
| 46 | 0 |
| 87 | 0 |
| 96 | 0 |
| 45 | 0 |
| 32 | 1 |
| 63 | 0 |
| 21 | 1 |
| 26 | 1 |
| 93 | 0 |
| 68 | 1 |
| 96 | 0 |

Image showing randomly generated data

This image shows a basic example of what classification data might look like. We have a predictor (or set of predictors) and a label. In the image, we might be trying to predict whether someone likes pineapple (1) on their pizza or not (0) based on their age (the predictor).

It is standard practice to represent the output (label) of a classification algorithm as an integer number such as 1, -1, or 0. In this instance, these numbers are purely representational. Mathematical operations should not be performed on them because doing so would be meaningless. Think for a moment. What is ―likes pineapple‖ + ―does not like pineapple‖ ? Exactly. We cannot add them, so we should not add their numeric representations.

A **regression problem** has a real number (a number with a decimal point) as its output. For example, we could use the data in the table below to estimate someone's weight given their height.

| Height(Inches) | Weight(Pounds) |
|---|---|
| 65.78 | 112.99 |
| 71.52 | 136.49 |
| 69.40 | 153.03 |
| 68.22 | 142.34 |
| 67.79 | 144.30 |
| 68.70 | 123.30 |
| 69.80 | 141.49 |
| 70.01 | 136.46 |
| 67.90 | 112.37 |
| 66.78 | 120.67 |
| 66.49 | 127.45 |
| 67.62 | 114.14 |
| 68.30 | 125.61 |
| 67.12 | 122.46 |
| 68.28 | 116.09 |

Image showing a portion of the SOCR height and weights data set

Data used in a regression analysis will look similar to the data shown in the image above. We have an independent variable (or set of independent variables) and a dependent variable (the thing we are trying to guess given our independent variables). For instance, we could say height is the independent variable and weight is the dependent variable.

Also, each row is typically called an **example, observation, or data point**, while each column (not including the label/dependent variable) is often called a **predictor, dimension, independent variable, or feature.**

An **unsupervised machine learning** algorithm makes use of input data without any labels —in other words, no teacher (label) telling the child (computer) when it is right or when it has made a mistake so that it can self-correct.

Unlike supervised learning that tries to learn a function that will allow us to make predictions given some new unlabeled data, unsupervised learning tries to learn the basic structure of the data to give us more insight into the data.

## KNN Algorithm

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

"Birds of a feather flock together."

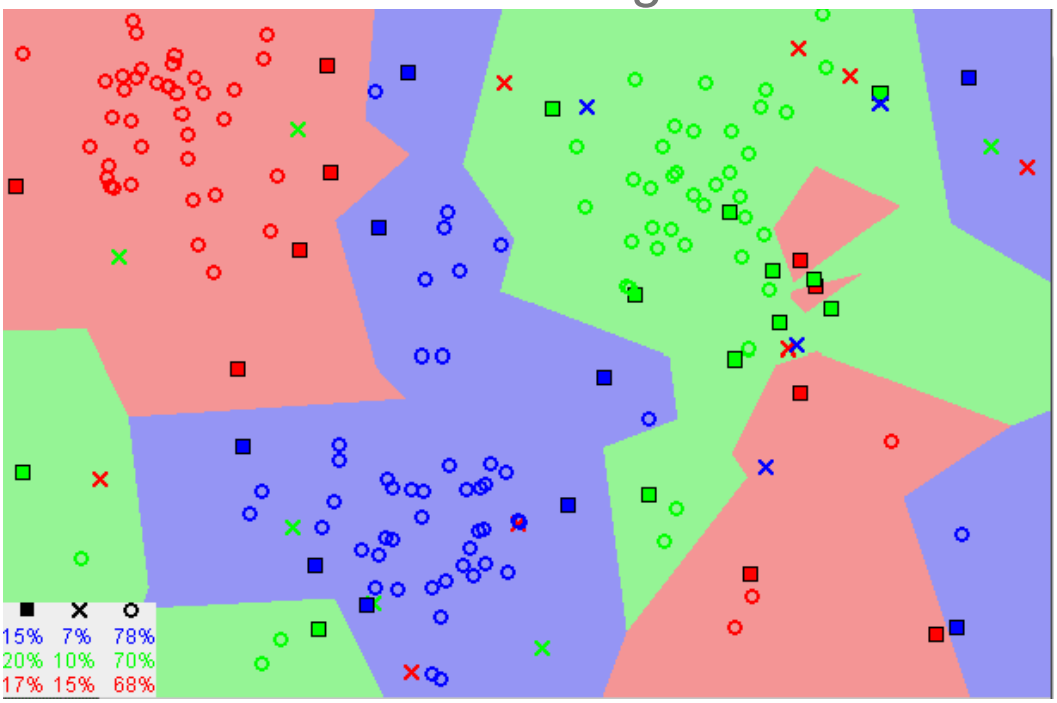



**Image showing how similar data points typically exist close to each other**

Notice in the image above that most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

## The KNN Algorithm

3. Load the data

4. Initialize K to your chosen number of neighbors

3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

5. Pick the first K entries from the sorted collection

6. Get the labels of the selected K entries

7. If regression, return the mean of the K labels

8. If classification, return the mode of the K labels

**Implemetation:**

Step 1: Reading and Understanding the Data

Step 2: Data Cleansing

Step 3: Data Preparation

Step 4: Model Building

Step 5: Final Analysis

**Dataset-** diabetes.csv

**Standard Libraries used are**

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

import warnings

warnings.filterwarnings('ignore')

%matplotlib inline
```

**Final results**

```
0.8193500639171096
```

**Conclusion:** Thus we have studed , K-nearest neibour for diebeties datasets.

| | |
|---|---|
| **Experiment No.** | 4 |
| **Title** | Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method. |
| **Seat No.** | |
| **Class** | B.E. (C.E.) |
| **Date** | |
| **Subject** | **Laboratory Practice III (ML)** |

**Experiment No:4**

**Title:** Implement K-Means clustering/ hierarchical clustering on s a l e s _ d a t a _ s a m p l e .csv dataset. Determine the number of clusters using the elbow method.

**Objectives:**

- To learn the concept of K means clustering
- To able toto apply on sales datasets.

**Theory:**

**K means clustering**

## Clustering:

**Clustering** is one of the most common exploratory data analysis technique used to get an intuition about the structure of the data. It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different. In other words, we try to find homogeneous subgroups within the data such that data points in each cluster are as similar as possible according to a similarity measure such as euclidean-based distance or correlation-based distance. The decision of which similarity measure to use is application-specific.

Clustering analysis can be done on the basis of features where we try to find subgroups of samples based on features or on the basis of samples where we try to find subgroups of features based on samples. We'll cover here clustering based on features. Clustering is used in market segmentation; where we try to find customers that are similar to each other whether in terms of behaviors or attributes, image segmentation/compression; where we try to group similar regions together, document clustering based on topics, etc.

Unlike supervised learning, clustering is considered an unsupervised learning method since we don't have the ground truth to compare the output of the

clustering algorithm to the true labels to evaluate its performance. We only want to try to investigate the structure of the data by grouping the data points into distinct subgroups.

In this post, we will cover only **Kmeans** which is considered as one of the most used clustering algorithms due to its simplicity.

**K means Algorithm:**

**Kmeans** algorithm is an iterative algorithm that tries to partition the dataset into *K*pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way kmeans algorithm works is as follows:

1. Specify number of clusters *K*.
2. Initialize centroids by first shuffling the dataset and then randomly selecting *K* data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.

- Compute the sum of the squared distance between data points and all centroids.
- Assign each data point to the closest cluster (centroid).

- Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

The approach kmeans follows to solve the problem is called **Expectation-Maximization**. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster. Below is a break down of how we can solve it mathematically (feel free to skip it).

The objective function is:

$$J = \sum_{i=1}^{m} \sum_{k=1}^{K} w_{ik} \|x^i - \mu_k\|^2 \qquad (1)$$

where wik=1 for data point xi if it belongs to cluster *k*; otherwise, wik=0. Also, μk is the centroid of xi's cluster.

It's a minimization problem of two parts. We first minimize J w.r.t. wik and treat μk fixed. Then we minimize J w.r.t. μk and treat wik fixed. Technically speaking, we differentiate J w.r.t. wik first and update cluster assignments (*E-step*). Then we differentiate J w.r.t. μk and recompute the centroids after the cluster assignments from previous step (*M-step*). Therefore, E-step is:

$$\frac{\partial J}{\partial w_{ik}} = \sum_{i=1}^{m} \sum_{k=1}^{K} \|x^i - \mu_k\|^2$$

$$\Rightarrow w_{ik} = \begin{cases} 1 & \text{if } k = argmin_j \|x^i - \mu_j\|^2 \\ 0 & \text{otherwise.} \end{cases} \qquad (2)$$

In other words, assign the data point xi to the closest cluster judged by its sum of squared distance from cluster's centroid.

And M-step is:

$$\frac{\partial J}{\partial \mu_k} = 2\sum_{i=1}^{m} w_{ik}(x^i - \mu_k) = 0$$

$$\Rightarrow \mu_k = \frac{\sum_{i=1}^{m} w_{ik}x^i}{\sum_{i=1}^{m} w_{ik}} \quad (3)$$

Which translates to recomputing the centroid of each cluster to reflect the new assignments.

Few things to note here:

- Since clustering algorithms including kmeans use distance-based measurements to determine the similarity between data points, it's recommended to standardize the data to have a mean of zero and a standard deviation of one since almost always the features in any dataset would have different units of measurements such as age vs income.
- Given kmeans iterative nature and the random initialization of centroids at the start of the algorithm, different initializations may lead to different clusters since kmeans algorithm may *stuck in a local optimum and may not converge to global optimum*. Therefore, it's recommended to run the algorithm using different initializations of centroids and pick the results of the run that that yielded the lower sum of squared distance.
- Assignment of examples isn't changing is the same thing as no change in within-cluster variation:

$$\frac{1}{m_k}\sum_{i=1}^{m_k} \|x^i - \mu_{c^k}\|^2 \quad (4)$$

## Implementation:

We'll use simple implementation of kmeans here to just illustrate some concepts. Then we will use `sklearn` implementation that is more efficient take care of many things for us.

## Applications:

kmeans algorithm is very popular and used in a variety of applications such as market segmentation, document clustering, image segmentation and image compression, etc. The goal usually when we undergo a cluster analysis is either:

1. Get a meaningful intuition of the structure of the data we're dealing with.
2. Cluster-then-predict where different models will be built for different subgroups if we believe there is a wide variation in the behaviors of different subgroups. An example of that is clustering patients into different subgroups and build a model for each subgroup to predict the probability of the risk of having heart attack.

In this post, we'll apply clustering on two cases:

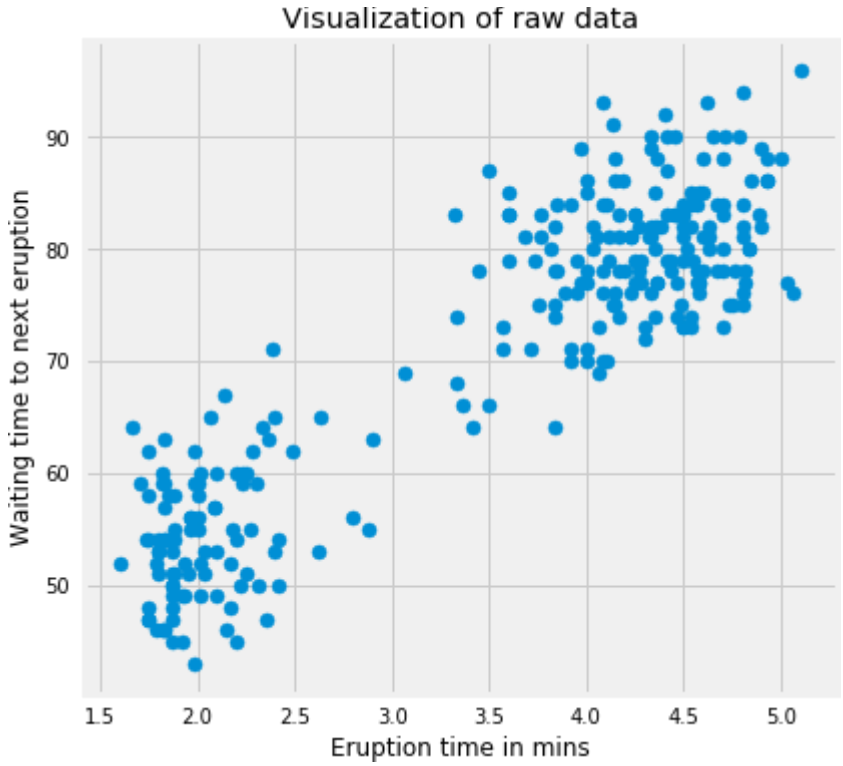
- Geyser eruptions segmentation (2D dataset).
- Image compression.

**Example:**

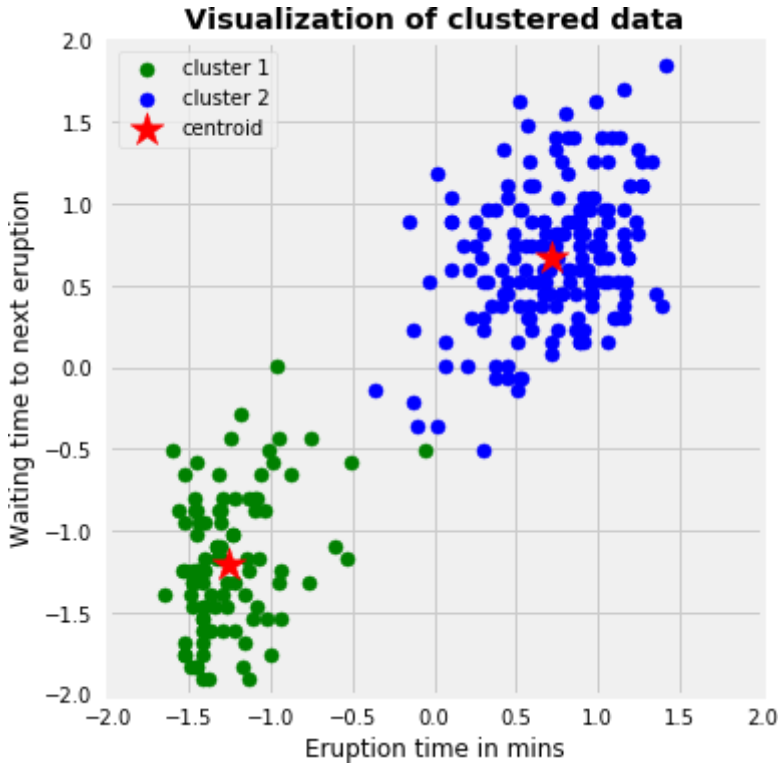
We'll first implement the kmeans algorithm on 2D dataset and see how it works. The dataset has 272 observations and 2 features. The data covers the waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. We will try to find *K* subgroups within the data points and group them accordingly. Below is the description of the features:

- eruptions (float): Eruption time in minutes.
- waiting (int): Waiting time to next eruption.

Let's plot the data first:




We'll use this data because it's easy to plot and visually spot the clusters since its a 2-dimension dataset. It's obvious that we have 2 clusters. Let's standardize the data first and run the kmeans algorithm on the standardized data with K=2.

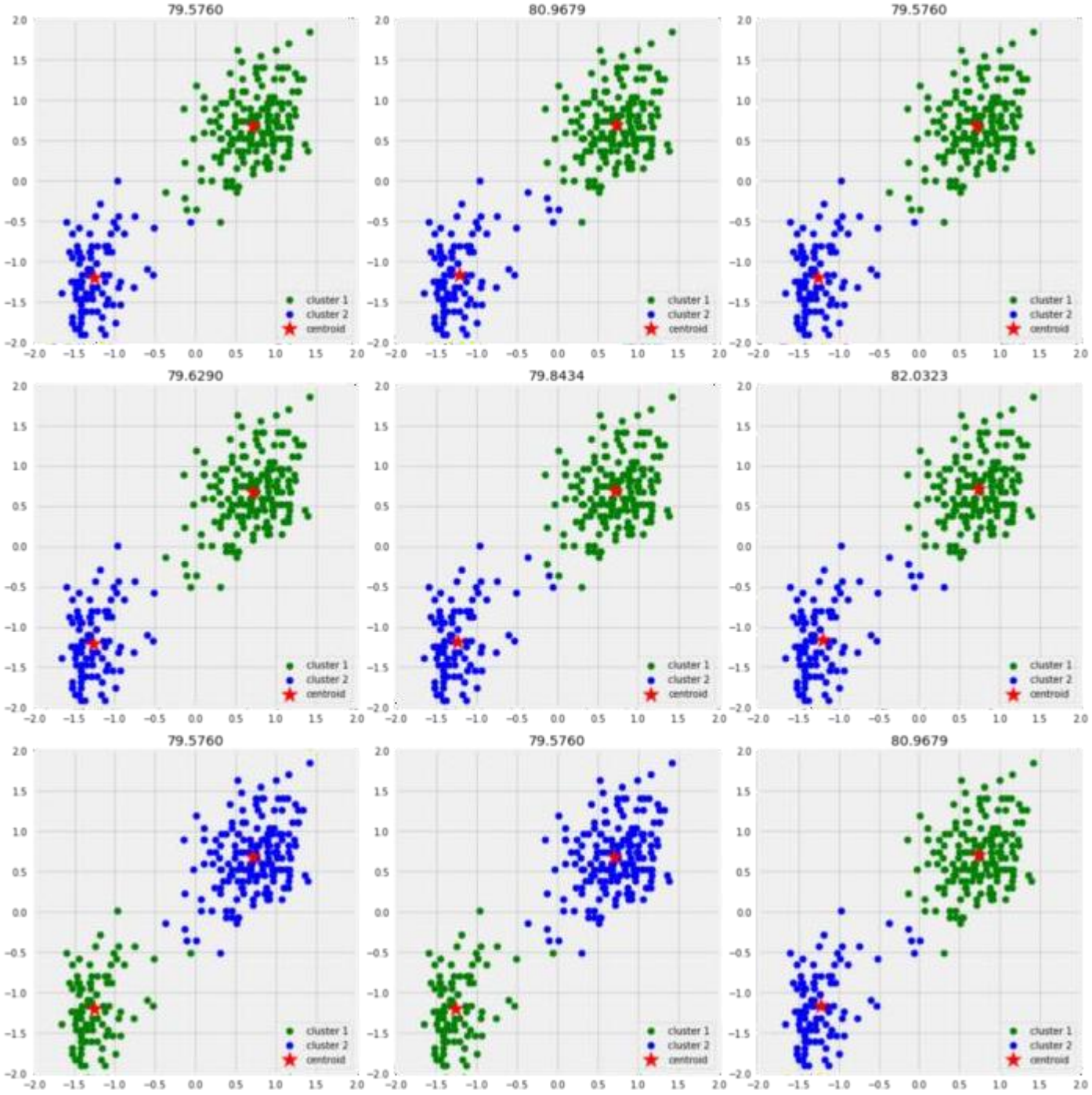



The above graph shows the scatter plot of the data colored by the cluster they belong to. In this example, we chose K=2. The symbol '*' is the centroid of each

cluster. We can think of those 2 clusters as geyser had different kinds of behaviors under different scenarios.

Next, we'll show that different initializations of centroids may yield to different results. I'll use 9 different `random_state` to change the initialization of the centroids and plot the results. The title of each plot will be the sum of squared distance of each initialization.

As a side note, this dataset is considered very easy and converges in less than 10 iterations. Therefore, to see the effect of random initialization on convergence, I am going to go with 3 iterations to illustrate the concept. However, in real world applications, datasets are not at all that clean and nice!




As the graph above shows that we only ended up with two different ways of clusterings based on different initializations. We would pick the one with the lowest sum of squared distance.

## Evaluation Methods:

Contrary to supervised learning where we have the ground truth to evaluate the model‘s performance, clustering analysis doesn‘t have a solid evaluation metric that we can use to evaluate the outcome of different clustering algorithms. Moreover, since kmeans requires *k* as an input and doesn‘t learn it from data,

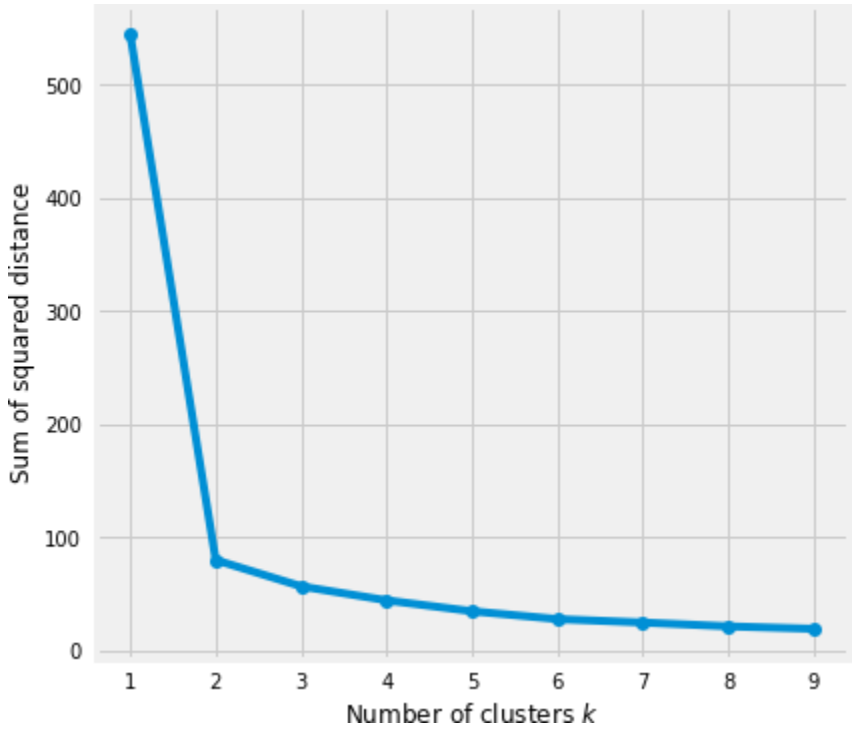
there is no right answer in terms of the number of clusters that we should have in any problem. Sometimes domain knowledge and intuition may help but usually that is not the case. In the cluster-predict methodology, we can evaluate how well the models are performing based on different *K* clusters since clusters are used in the downstream modeling.

In this post we'll cover two metrics that may give us some intuition about *k*:

- Elbow method
- Silhouette analysis

**Elbow Method:**

**Elbow** method gives us an idea on what a good *k* number of clusters would be based on the sum of squared distance (SSE) between data points and their assigned clusters' centroids. We pick *k* at the spot where SSE starts to flatten out and forming an elbow. We'll use the geyser dataset and evaluate SSE for different values of *k* and see where the curve might form an elbow and flatten out.



The graph above shows that k=2 is not a bad choice. Sometimes it's still hard to figure out a good number of clusters to use because the curve is monotonically decreasing and may not show any elbow or has an obvious point where the curve starts flattening out.

**Silhouette analysis** :

**Silhouette analysis** can be used to determine the degree of separation between clusters. For each sample:

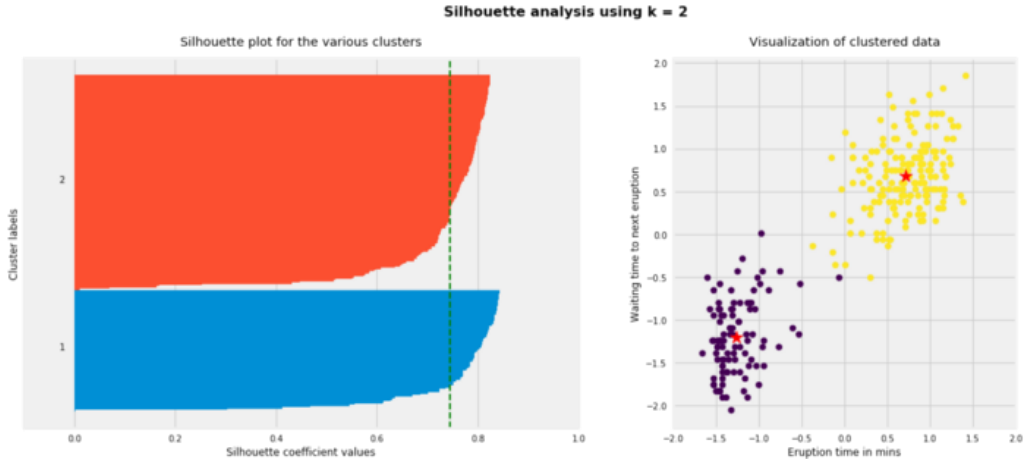
- Compute the average distance from all data points in the same cluster (ai).
- Compute the average distance from all data points in the closest cluster (bi).
- Compute the coefficient:

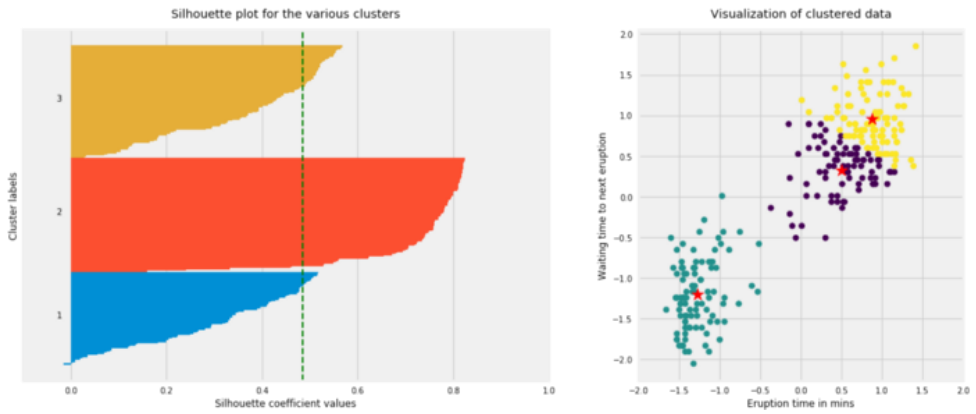
$$\frac{b^i - a^i}{max(a^i, b^i)}$$

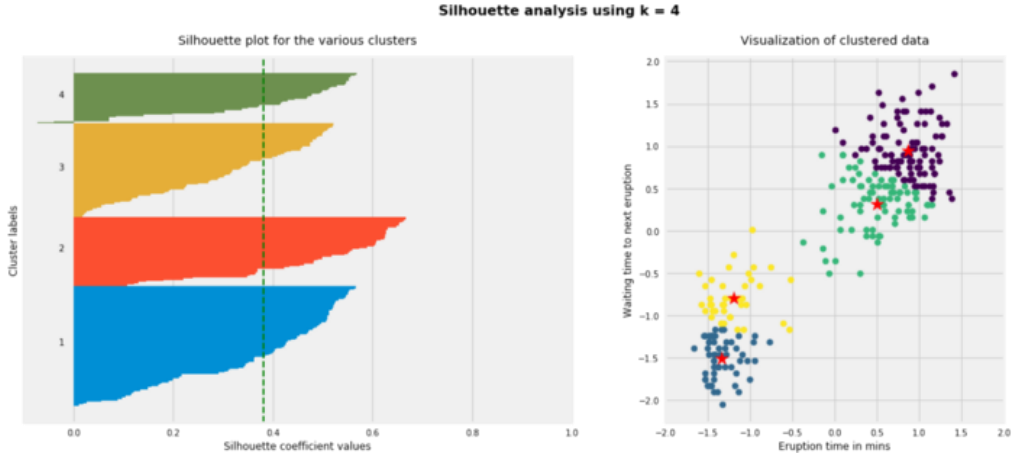
The coefficient can take values in the interval [-1, 1].

- If it is 0 –> the sample is very close to the neighboring clusters.
- It it is 1 –> the sample is far away from the neighboring clusters.
- It it is -1 –> the sample is assigned to the wrong clusters.

Therefore, we want the coefficients to be as big as possible and close to 1 to have a good clusters. We'll use here geyser dataset again because its cheaper to run the silhouette analysis and it is actually obvious that there is most likely only two groups of data points.



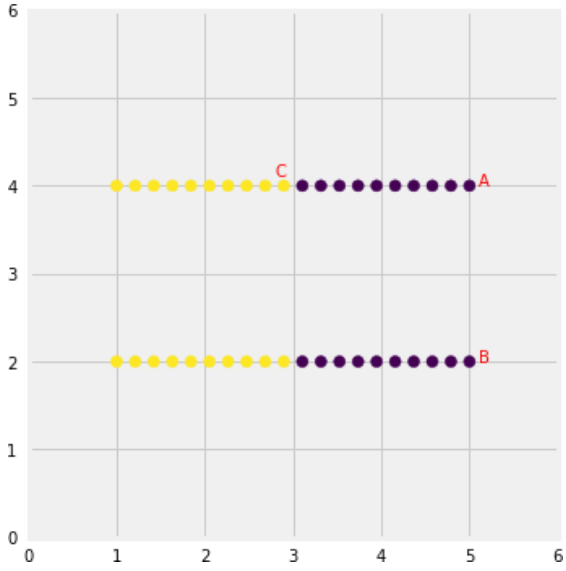




As the above plots show, `n_clusters=2` has the best average silhouette score of around 0.75 and all clusters being above the average shows that it is actually a good choice. Also, the thickness of the silhouette plot gives an indication of how big each cluster is. The plot shows that cluster 1 has almost double the samples than cluster 2. However, as we increased `n_clusters` to 3 and 4, the average silhouette score decreased dramatically to around 0.48 and 0.39 respectively. Moreover, the thickness of silhouette plot started showing wide fluctuations. The bottom line is: Good `n_clusters` will have a well above 0.5 silhouette average score as well as all of the clusters have higher than the average score.

## Drawback

Kmeans algorithm is good in capturing structure of the data if clusters have a spherical-like shape. It always try to construct a nice spherical shape around the centroid. That means, the minute the clusters have a complicated geometric shapes, kmeans does a poor job in clustering the data. We'll illustrate three cases where kmeans will not perform well.

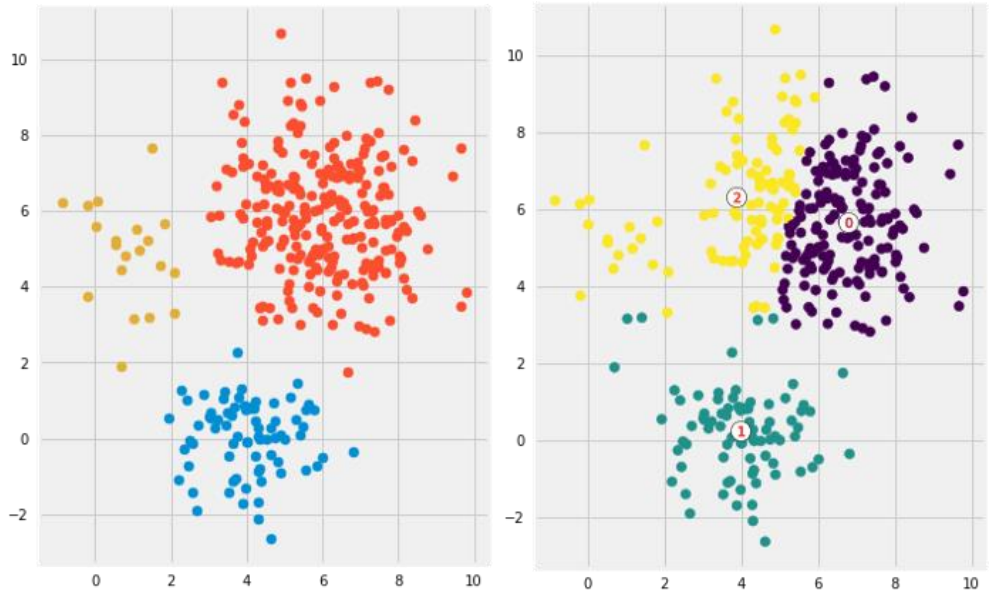
First, kmeans algorithm doesn't let data points that are far-away from each other share the same cluster even though they obviously belong to the same cluster. Below is an example of data points on two different horizontal lines that illustrates how kmeans tries to group half of the data points of each horizontal lines together.




Kmeans considers the point _B' closer to point _A' than point _C' since they have non-spherical shape. Therefore, points _A' and _B' will be in the same cluster but point _C' will be in a different cluster. Note the **Single Linkage** hierarchical clustering method gets this right because it doesn't separate similar points).

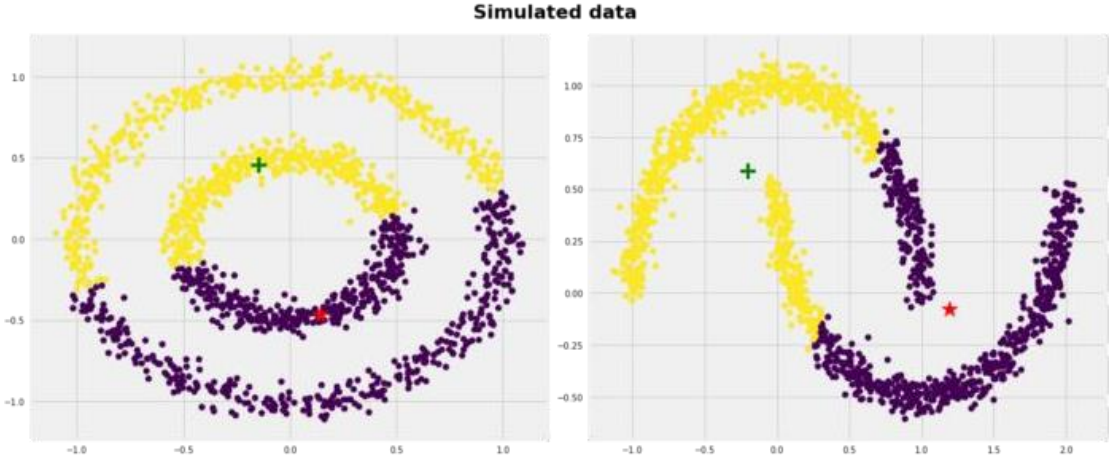
Second, we'll generate data from multivariate normal distributions with different means and standard deviations. So we would have 3 groups of data where each group was generated from different multivariate normal distribution (different mean/standard deviation). One group will have a lot

more data points than the other two combined. Next, we'll run kmeans on the data with K=3 and see if it will be able to cluster the data correctly. To make the comparison easier, I am going to plot first the data colored based on the distribution it came from. Then I will plot the same data but now colored based on the clusters they have been assigned to.

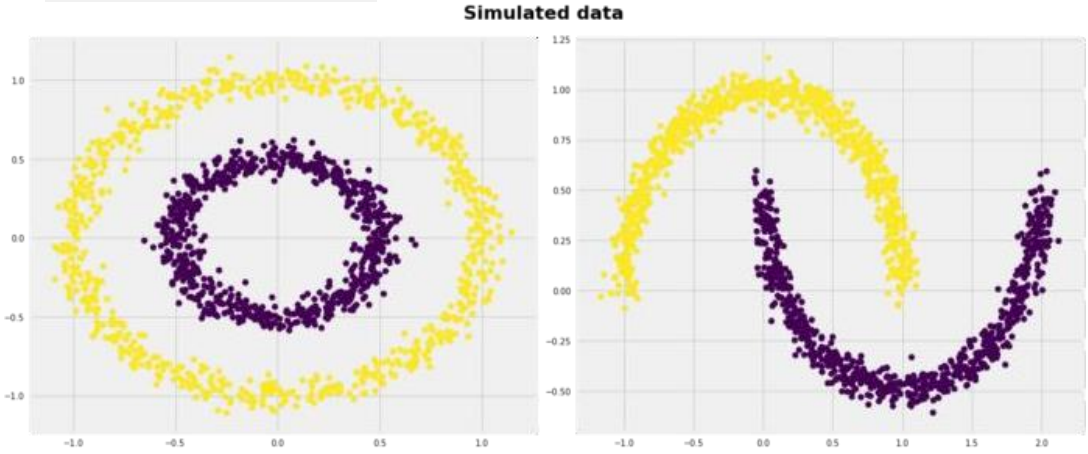



Looks like kmeans couldn't figure out the clusters correctly. Since it tries to minimize the within-cluster variation, it gives more weight to bigger clusters than smaller ones. In other words, data points in smaller clusters may be left away from the centroid in order to focus more on the larger cluster.

Last, we'll generate data that have complicated geometric shapes such as moons and circles within each other and test kmeans on both of the datasets.



As expected, kmeans couldn‘t figure out the correct clusters for both datasets. However, we can help kmeans perfectly cluster these kind of datasets if we use kernel methods. The idea is we transform to higher dimensional representation that make the data linearly separable (the same idea that we use in SVMs). Different kinds of algorithms work very well in such scenarios such as `SpectralClustering`, see below:




**Implemetation:**

Step 1: Reading and Understanding the Data

Step 2: Data Cleansing

Step 3: Data Preparation

Step 4: Model Building

Step 5: Final Analysis

**Dataset-** OnlineRetail.csv

**Standard Libraries used are**

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import datetime as dt
```

```
# import required libraries for clustering

import sklearn

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score

from scipy.cluster.hierarchy import linkage

from scipy.cluster.hierarchy import dendrogram

from scipy.cluster.hierarchy import cut_tree
```

**Final results**

Inference: K-Means Clustering with 3 Cluster Ids

Customers with Cluster Id 1 are the customers with high amount of transactions as compared to other customers. Customers with Cluster Id 1 are frequent buyers. Customers with Cluster Id 2 are not recent buyers and hence least of importance from business point of view.

Hierarchical Clustering with 3 Cluster Labels

Customers with Cluster_Labels 2 are the customers with high amount of transactions as compared to other customers. Customers with Cluster_Labels 2 are frequent buyers. Customers with Cluster_Labels 0 are not recent buyers and hence least of importance from business point of view.

**Conclusion:**We are able to implement K means clustering algorithm.