3. APPLIED MACHINE LEARNING USING PYTHON – CLASSIFICATION & CLUSTERING

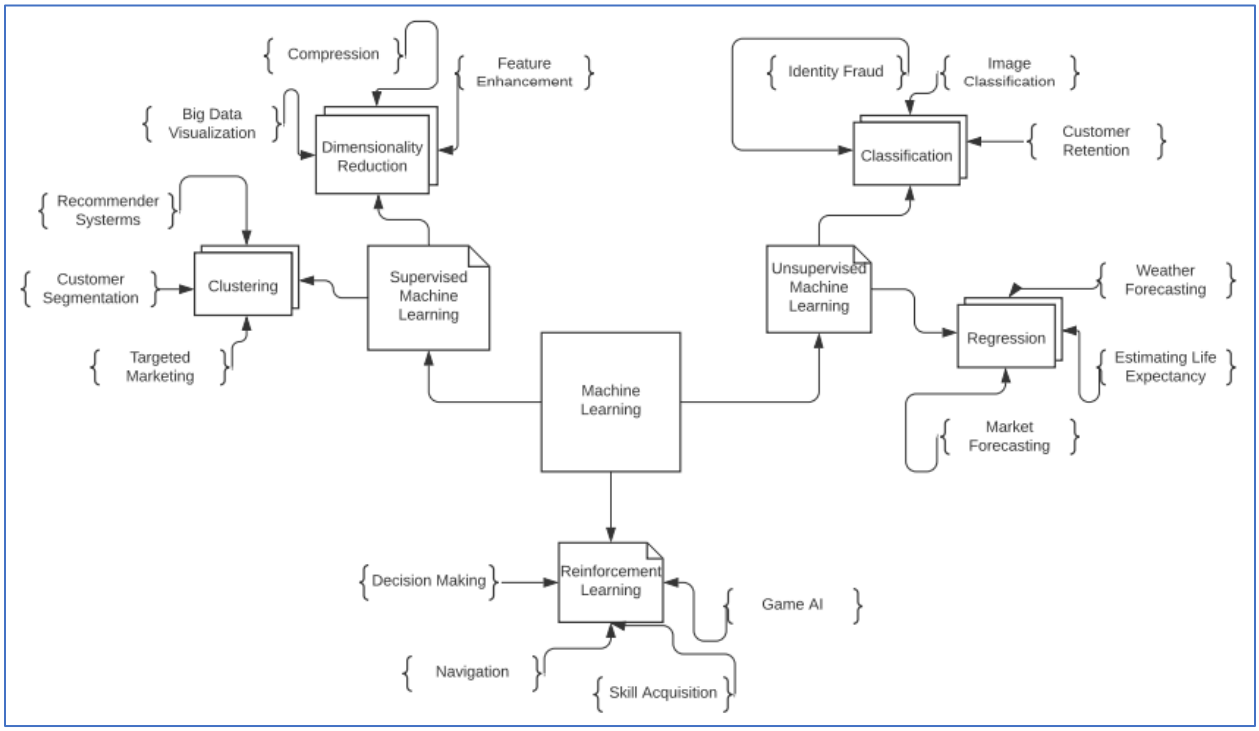
In the previous chapters we have discussed about programming basics in Python and the role of Mathematics in indulging Artificial Intelligence to software development. Both concepts are interrelated and form the core of building inner AI theories like Machine Learning and Data Science. Learning is defined as the acquisition of knowledge and skills through practice, learning or experience. Based on the idea of learning, machine learning implements this through studying given input and understanding the data.

Machine Learning, a subbranch of the broad Artificial Intelligence bracket, is a computer science field that generates the ability to learn within software systems. The aim of this learning is to improve existing data quality from the experience of learning, without having to be explicitly told to generate a specific output. The emphasis of machine learning is to help computer systems learn from the provided input without human interference.

Now, to implement this form of learning like the human brain, computer systems also need information. To provide this data to computers we go through a phase that is called Data Observation or Data Preparation. During this time, the computer system is exposed to tremendous amounts of data that aid its learning process. Experience with this input data gives the machine an understanding of what patterns to study and the end goal of the provided information. This in-turn helps in generating the required analysis and forecasting as output.

Machine Learning and its Varied Forms

Algorithms in Machine Learning are segregated based on the type of input they receive and the computations they do to learn from it. We will soon cover the mathematics and detailed implementation of these algorithms, but for now let us look at the broad classification of these algorithms.



Machine Learning is broadly classified into three categories that is Supervised, Unsupervised and Reinforcement Learning. The above image shows differentiation between these algorithms and their use cases.

Supervised Machine Learning

Algorithms that work on labelled data are called supervised learning algorithms. These are the most used mechanisms for training and working on data. Labelled data means data that has questions and answers both present. The algorithm looks at all the various input and expected output and learns what kinds of input map to what form of output. Based on this learning, when new data is provided to this algorithm, it can predict the output.

To get the understanding of how supervised learning functions work, presume we have an input variable x and an output variable y. A function is needed to map the input and the output.

The function **Y=f(x)** is used to learn the mapping between the input and output

Now, the central objective is to estimate this plotting function in a manner that, whenever new input data (x) is presented to the expression, the algorithms predict the output (Y) from this input.

Primarily supervised leaning problems can be divided into the following two kinds of problems:

- **Classification:** A categorical variable solution is called classification. Categorical variables include outputs like color, gender, etc. that can have a specified set of values and the algorithm needs to choose from this set. We will be covering Classification in detail in this chapter.
- **Regression:** A regression is a problem that has a definite solution to predict. For instance, the weight of a car, or the height of a tree. These outputs are definite values. Predicting these is a form of regression.

Some algorithms that are generally used in supervised learning tasks are Decision Trees, Random Forests, Nearest Neighbor Solutions (KNN), Logistic Regression.

Unsupervised Machine Learning

The key difference between supervised and unsupervised algorithms is that unsupervised algorithms are not provided with labelled data. That means, the expected output of the logic is not known to the algorithm during its training process. Therefore, unsupervised learning algorithms are what people generally associate with Artificial Intelligence. It can be understood as follows: These algorithms do not necessarily have a correct or incorrect answer. They only help in understanding theories, patterns of information and help discover interesting forms in data.

Unsupervised Machine Learning algorithms can be apportioned into the following:

- **Clustering:** Problems associated with clustering help in discovering groupings within data. These groupings of data are generally inherently present in the input data. For example, grouping soccer players based on their attributes of scoring goals. An example of this algorithm is K-Means Clustering.
- **Association:** A solution is called association when the algorithm tries to discover the rules that connect portions of the input data. For instance, finding common traits in players who score goals as well as give assists. An example is Apriori algorithm.

Reinforcement Machine Learning

The use of reinforcement learning algorithms is comparatively less to the above two (supervised and unsupervised). These procedures train the learning algorithms to make decisions. The basis of this decision making is the exposure this algorithm gets to environment. It is expected to learn from these surroundings and understand the behavior of objects in this situation using a trial-and-error mechanism. Since the algorithm is based on learning from its own

experience, decisions tend to be more accurate if the algorithm is built properly and go drastically wrong otherwise. This extreme nature of output possibility makes the use of this algorithm less popular. Markov Decision Process is an illustration of a reinforcement learning algorithm.

Classification and Clustering of Data

Classification is a type of supervised learning and Clustering is used in unsupervised learning algorithms. Both algorithms work on the logic of grouping similar data points together. The difference lies in the fact that classification happens when data is labelled, and output is known whereas clustering occurs on unlabeled data. Classification stipulates the class onto which data elements belong and predicts a class for a finite input variable. Clustering on the other hand, can work on infinite set of data points as well.

There are 2 types of Classification:

- **Binomial:** The output is aimed to be grouped and classified into one of two possible categories.
- **Multi-Class:** The idea is to group an outcome into one of multiple groups.

Use cases of Classification:

1. Finding out whether a received email is spam based on the history of spam messages received.
2. Identify segments of shopping customers based on their purchase patterns.
3. To understand if a customer will repay his loan based on his credit history.
4. Predicting whether a kid will pass a given examination based on the analysis of questions and the student's academic background.

Clustering or Cluster Analysis

Cluster Analysis is an unsupervised learning mechanism that helps in determining interesting patterns in data. There is generally no correct algorithm to use and it is always considered a good practice to explore a range of algorithms with varied combinations to conclude. A cluster is often an area of concentrated density in the given input space.

Use cases of Clustering:

1. Identifying fake news and sensationalized or click-bait articles.
2. Personalized advertisement targeting for customers based on their search patters.
3. Understanding the broad area of network traffic on a page.
4. Analysis of fantasy football and recommendation of players for next week's team.

Understanding Classifiers in Python

To build classifiers in Python, we will use the scikit-learn library that is used for Machine Learning. Here are some important functions and methods we need to understand before moving onto building a classifier.

Scikit-Learn: The scikit learn library is probably the most comprehensive toolkit for machine learning in Python. It contains methods for training learning models, performing statistical analysis, and visualizing data. We will be using scikit learn throughout this course for implementing various machine learning and artificial intelligence algorithms.

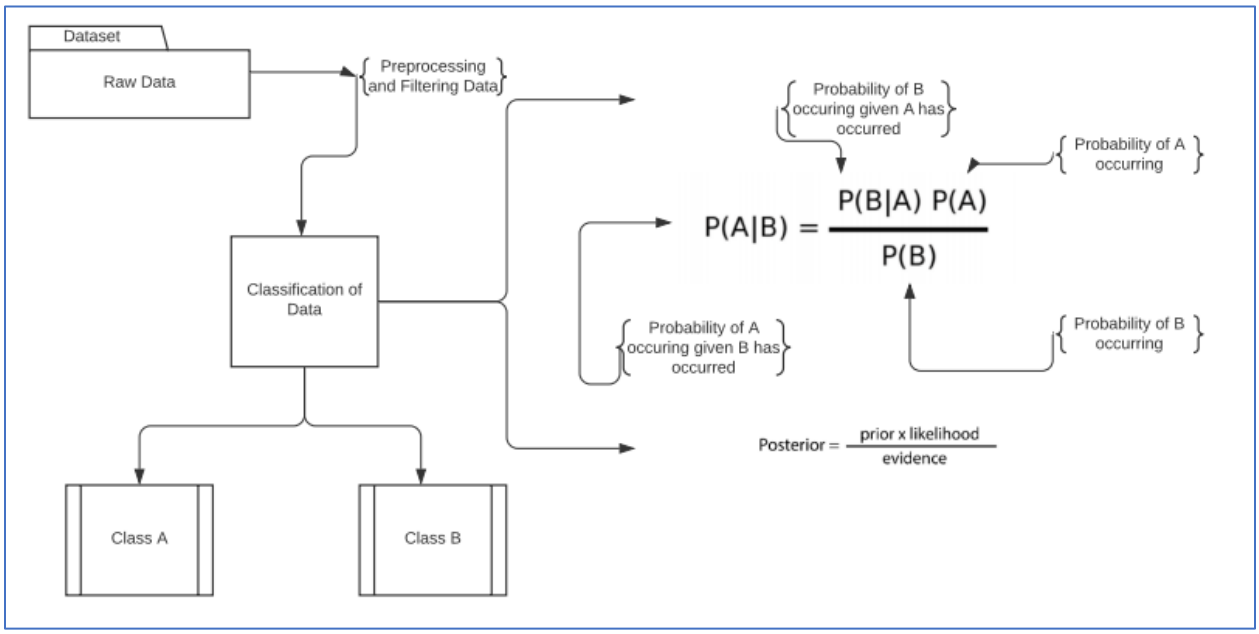
Importing Datasets from Scikit-Learn: The scikit learn library has various inbuilt datasets meant for practicing machine learning problems. These datasets are open-sourced and available for using in locally running algorithms. We will be using import statements to fetch data from some of these datasets.

Train-Test-Split: This method is commonly used to split arrays or matrices into arbitrary training and testing groups. The splitting can be explicitly mentioned as an argument of the function while calling it. This split makes sure that all the present data is not used during training, to avoid situations of **overfitting and underfitting** the model.

```
# The Train Test and Split Method Definition sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)
# test_size: Value should be between 0.0 and 1.0 and signifies the percent of data we are putting under test data
# train_size: Like test size for training data divison

# Implementing it using a numpy array
import numpy as np
from sklearn.model_selection import train_test_split
X, y = np.arange(10).reshape((5, 2)), range(5)
X
>>> array([[2,3], [0,1], [4, 5], [8,9], [6,7]])
list(y)
>>> [0, 1, 2, 3, 4]

# Doing a train test and split on the data with a test size of 43% of input data
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.43)
# Check all the 4 individual arrays to see that the input has been split
X_train
>>> array([[0, 1], [6, 7], [4, 5]])
y_train
>>> [0, 3, 2]
X_test
>>> array([[8,9], [2,3]])
y_test
>>> [1, 4]
# This process helps in avoiding underfitting and overfitting problems with data
```

We will be using concepts revolving around sci-kit learn, NumPy, pandas, matplotlib and seaborn in the remainder of the course to get hold of data preparation, cleaning, visualization, and training. These libraries combined make Python one of the strongest languages for solving data science problems. Let us get started with our first machine learning example in the following section.

The Naïve-Bayes Algorithm



Representation of how Naïve Bayes algorithms work. Generally, all other classification and clustering models work on the same principles of grouping data. Only their inline mathematical operations are different.

```
# Importing the Sci-kit learn library and importing Wisconsin breast cancer dataset
import sklearn
from sklearn.datasets import load_breast_cancer
data_set = load_breast_cancer()

# Differentiating features and labels on the imported data set
label_names = data_set['target_names']
labels = data_set['target']
feature_names = data_set['feature_names']
features = data_set['data']

# We observe the that are two categories of cancer that need to be labelled (Output Parameters)
print(label_names)
>>> ['malignant' 'benign']

# Printing the raw data from the data-set and using .DESCR function to retrieve the descriptions of columns present in the data
print(data_set.data)
>>>
```

```
 [1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
.... Output Truncated
```

```
print(data_set.DESCR)
```

>>>

```
Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------
**Data Set Characteristics:**
   :Number of Instances: 569
   :Number of Attributes: 30 numeric, predictive attributes and the class
   :Attribute Information:
      - radius (mean of distances from center to points on the perimeter)
      - texture (standard deviation of gray-scale values)
      - perimeter
      - area
      - smoothness (local variation in radius lengths)
      - compactness (perimeter^2 / area - 1.0)
      - concavity (severity of concave portions of the contour)
      - concave points (number of concave portions of the contour)
      - symmetry
      - fractal dimension ("coastline approximation" - 1)
....
.....Output Truncated
```

```
# Import pandas to use data frames for exploration

# Read the DataFrame, using the feature names from the data set

import pandas as pd

df = pd.DataFrame(data_set.data, columns=data_set.feature_names)

# Add a column to generate the ouput

df['target'] = data_set.target


# Use the head() method from Pandas to see the top of the data set

df.head()
```

```
>>>
```

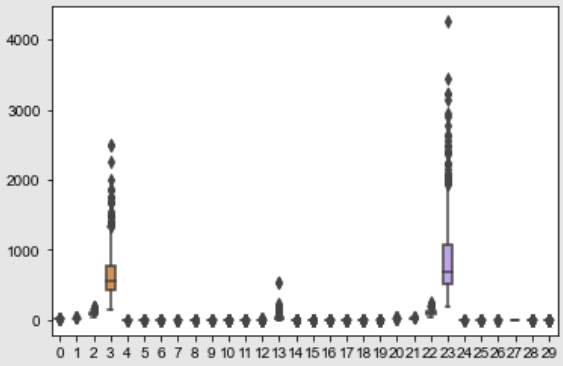
	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	target
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	0

1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...
	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	0	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...
	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	0	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...
	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	0	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...
	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	0	

5 rows × 31 columns

```
# Import the seaborn library to visualize the input data
# We will form a box plot to understand the spread of the data set in comparison to the target column
import seaborn as sns
box_plot = data_set.data
box_target = data_set.target
sns.boxplot(data = box_plot,width=0.6,fliersize=7)
sns.set(rc={'figure.figsize':(2,15)})
```

>>>




```
# We will now move towards applying Machine Learning to this data
# Currently we know the target variables, and we have labelled input data as well.
# Use the train_test_split to create the training and testing data sets
from sklearn.model_selection import train_test_split
train, test, train_labels, test_labels = train_test_split(features,labels,test_size = 0.33, random_state = 40)

# Import the Naïve Bayes Classifier from Sci-kit learn's library
from sklearn.naive_bayes import GaussianNB
g_nb = GaussianNB()

# The fit method is used to train the model using the training data
```

```
model = g_nb.fit(train, train_labels)
# The predict method runs the learned model on the test data to generate the output
predicts = g_nb.predict(test)
print(predicts)
>>> [1 0 1 1 0 1 1 1 0 1 0 1 1 1 0 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 1 0 0 1 1 1 1
 0 1 1 0 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 1 1 0 0 1
 0 0 0 1 1 1 1 1 0 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1
 1 0 1 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1
 1 0 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 1
 1 1 0]
# We get an output that predicts the type of cancer ('malignant' or 'benign') caused from the given inputs

# Every time an algorithm is created and run, use the accuracy_score to check the working of that algorithm
from sklearn.metrics import accuracy_score
print(accuracy_score(test_labels,predicts))
>>> 0.9680851063829787

# This algorithm model has nearly a 97% accuracy
```

This is a basic example of implementing the Naïve Bayes classifier using Python. We will be looking into feature engineering and selection of elements for bettering the preprocessing of data in the next chapter.

Clustering Algorithms and Performance Measures

Clustering is an unsupervised machine learning method that is generally used for statistical data analysis across various fields and domains. The task is to divide the set of observations into subsets, that are called clusters or groups. Observations that belong to the same group have similar set of features and these characteristics differentiate the clusters from each other.

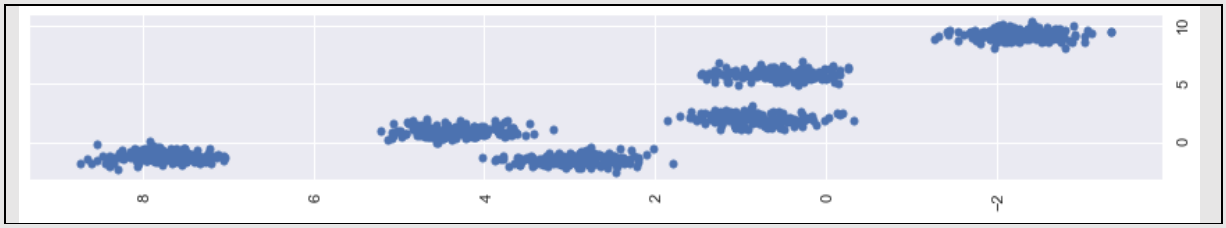
The K-Means Clustering Algorithm

This algorithm assumes that the total number of groups of clusters within the given data are already known. This method is also referred to as flat clustering and it is an iterative mechanism. The functioning of the algorithm works something like this:

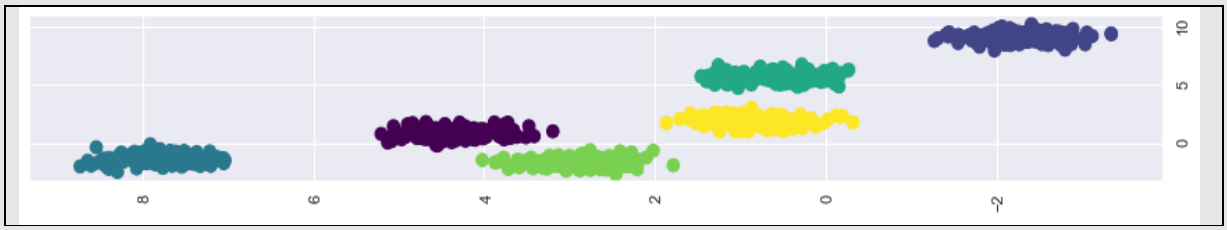
1. The number of assumed clusters within the data need to be defined (K)
2. Classify the data points from the input into these given clusters
3. Continuously adjust the size of clusters by moving the data points to their closest matching cluster
4. With every iteration, the cluster centroids are updated until the centroids reach their optimal locations

```
# Let us now move towards creating a Clustering Algorithm in Python
import matplotlib.pyplot as plt
```

```
import seaborn as sns; sns.set()
import numpy as np
# Import the KMeans package from Sci-Kit Learn
from sklearn.cluster import KMeans

# Creating randomly generated data in the form of Blobs
# Total Samples = 1000, Total Centers = 6, Standard Deviation of Clusters = 0.40
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=1000, centers=6, cluster_std=0.40)

# Plot the clusters on a Seaborn Scatterplot
plt.scatter(X[:, 0], X[:, 1], s=25);
plt.show()
>>>
```

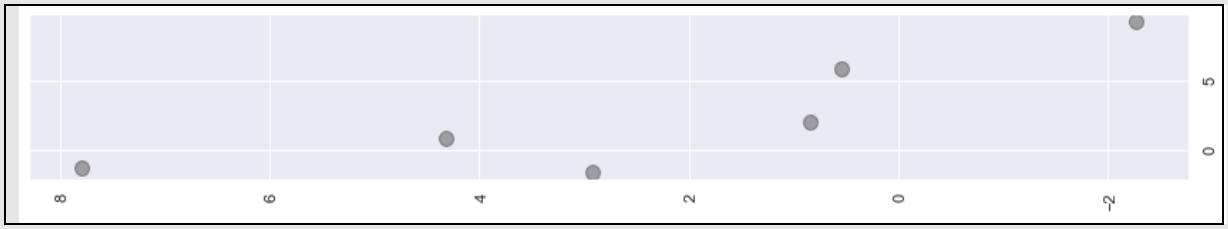



```
# We see six distinct clusters on the plot
# Let us now import KMeans and find the centroids of these clusters
kmeans = KMeans(n_clusters=6)

# Fit the model with the generated data
kmeans.fit(X)
# Run the model on the given data plot the result graph
y_kmeans = kmeans.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=75, cmap='viridis')
centers = kmeans.cluster_centers_
>>>
```




```
# We can now see the clusters uniquely identifiable
# Checking implementation through centroids
```

```
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
plt.show()
>>>
```




```
# Centroids of the six clusters
```

Measuring the Performance of Clustering

Since clustering algorithms are not about predicting a value or deriving an answer, metrics like accuracy score matter a lot. There are various methods through which clustering algorithms can be measured. Since real-world data will likely never be segregated into sections, drawing inference is difficult. We will look at a method called **Silhouette Analysis** to measure the performance of our clustering algorithm.

Analysis of silhouette score: The range of this score is between [-1, 1]. Below is a detailed analysis of measurement.

- Score of +1: A positive score of 1, shows that the predicted cluster data point is far away from its nearest neighboring cluster.
- Score of 0: A neutral or 0 score shows that the given sample is either residing on the decision boundary amongst distinct neighboring clusters or is very close to this boundary.
- Score of -1: A negative score of -1, gives the indication that a sample has been moved to the wrong cluster.

Formula for Calculation: $silhouette\ score=(p-q)/\max(p,q)$

In the above formula, p represents the mean distance between the given points and corresponding data points in the nearest neighboring clusters. q gives the mean distance between all points within the same cluster.

```
# Import all required libraries along with Sci-kit learn metrics function
import matplotlib.pyplot as plt
import seaborn as sns;
import numpy as np
from sklearn.cluster import KMeans
# The metrics function will help us use the silhouette score construct
from sklearn import metrics

# Create the sample space using make_blobs method
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=1500, centers=8, cluster_std=0.60)
```

```
# We have created a sample space with 1500 entries and 8 centers

# Create an empty numpy array and another array with evenly spaced values
scores = []
values = np.arange(4, 20)

# Iterate through the array and initialize each of the values for clustering with given k value
# 'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
for num_clusters in values:
    kmeans = KMeans(init='k-means++', n_clusters=num_clusters, n_init=10)
    kmeans.fit(X)

# Import the silhouette score metric and use the Euclidean distance to check for ideal centroids
score = metrics.silhouette_score(X, kmeans.labels_, metric='euclidean', sample_size=len(X))

# The metric will now update the number of clusters and give the score
print("\nNumber of clusters =", num_clusters)
print("Silhouette score =", score)
scores.append(score)
>>> Number of clusters = 19
>>> Silhouette score = 0.3294464776999117

# Using numpy's argmax function, we can now get the number of optimal clusters needed for this sample data
num_clusters = np.argmax(scores) + values[0]
print('\nIdeal number of clusters =', num_clusters)
>>> Ideal number of clusters = 4
```

Grouping Data and Approaching Nearest Neighbors

All classification and clustering problems, at their root solve the data using a grouping mechanism. This mechanism if often associated with creating clusters of data and trying to predict where the new data point will end up. On similar lines, exists another algorithm which helps in identifying Nearest Neighbors of groups of data. The idea behind finding nearest neighbors is to find the closest point to the input that can be associated with trained data.

- Algorithms that work on data grouping can be used for making predictions where choices are available. For example, recommending a specific type of garment to a user based on his shopping and browsing history. In this scenario, the algorithm uses the user's preferences and history to club together a cluster of choices he/she is like to make.

- Another advantage of using a clustering mechanism is that, once the groups are formed, this algorithm easily adapts to newly presented data. Since clusters are generalized to various shapes and sizes, working on new data becomes faster and less likely to give errors.
- On the other hand, since classification converges choices to one of two choices, the core of these algorithms is also a narrowed set of groups. Therefore, identifying new data is faster.

Coming Up Next

The focus of this chapter was on grouping data into classification and clustering problems. Both methodologies, although follow different approaches, their core idea is of segregating data into groups. We also observed the various use cases in which these algorithms can be used.

1. Understanding Regression in Machine Learning
2. Feature Engineering and Ensemble Modelling in Python
3. Improving Performance of Learning Algorithms