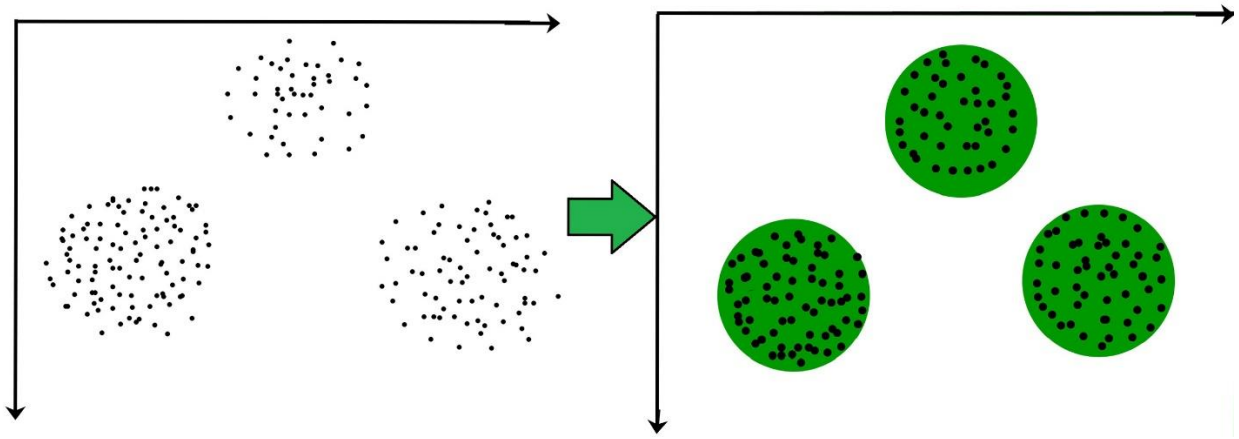
WEEK 10: DATA CLUSTERING K-means

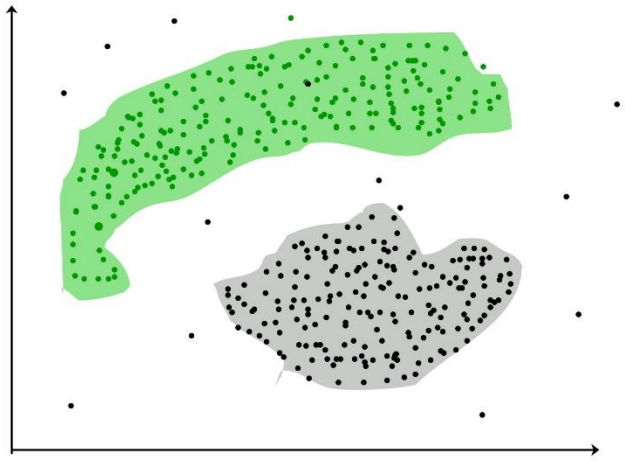
Introduction to Clustering: It is basically a type of *unsupervised learning method*. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labeled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

For example The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.



It is not necessary for clusters to be spherical as depicted below:



DBSCAN: Density-based Spatial Clustering of Applications with Noise

These data points are clustered by using the basic concept that the data point lies within the given constraint from the cluster center. Various distance methods and techniques are used for the calculation of the outliers.

Why Clustering?

Clustering is very much important as it determines the intrinsic grouping among the unlabelled data present. There are no criteria for good clustering. It depends on the user, and what criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), finding "natural clusters" and describing their unknown properties ("natural" data types), in finding useful and suitable groupings ("useful" data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions that constitute the similarity of points and each assumption make different and equally valid clusters.

Clustering Methods:

- **Density-Based Methods:** These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy

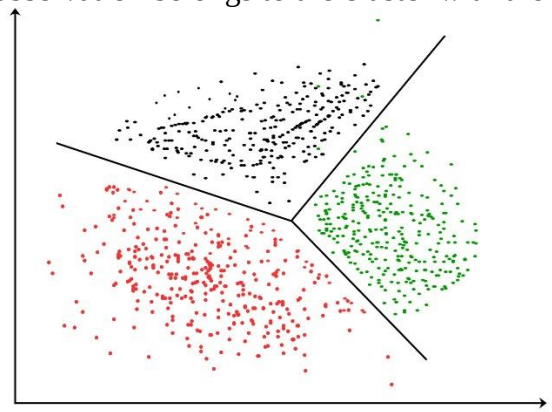
and the ability to merge two clusters. Example *DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure)*, etc.

- **Hierarchical Based Methods:** The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category
    - **Agglomerative** (bottom-up *approach*)
    - **Divisive** (top-down *approach*)

Examples *CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies)*, etc.

- **Partitioning Methods:** These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example *K-means, CLARANS (Clustering Large Applications based upon Randomized Search)*, etc.
- **Grid-based Methods:** In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects example *STING (Statistical Information Grid), wave cluster, CLIQUE (CLustering In Quest)*, etc.

Clustering Algorithms: K-means clustering algorithm – It is the simplest unsupervised learning algorithm that solves clustering problem.K-means algorithm partitions n observations into k clusters where each observation belongs to the cluster with the nearest mean serving as a prototype of the cluster.



1. K-Means Clustering

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

The goal of clustering is to divide the population or set of data points into a number of groups so that the data points within each group are more comparable to one another and different from the data points within the other groups. It is essentially a grouping of things based on how similar and different they are to one another.

We are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorize those items into groups. To achieve this, we will use the K-means algorithm; an unsupervised learning algorithm. 'K' in the name of the algorithm represents the number of groups/clusters we want to classify our items into.

(It will help if you think of items as points in an n-dimensional space). The algorithm will categorize the items into k groups or clusters of similarity. To calculate that similarity, we will use the Euclidean distance as a measurement.

The algorithm works as follows:

1. First, we randomly initialize k points, called means or cluster centroids.
2. We categorize each item to its closest mean and we update the mean's coordinates, which are the averages of the items categorized in that cluster so far.

3. We repeat the process for a given number of iterations and at the end, we have our clusters.

The "points" mentioned above are called means because they are the mean values of the items categorized in them. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set (if for a feature *x*, the items have values in [0,3], we will initialize the means with values for *x* at [0,3]).

The above algorithm in pseudocode is as follows:
Initialize k means with random values
--> For a given number of iterations:
--> Iterate through items:
--> Find the mean closest to the item by calculating the euclidean distance of the item with each of the means
--> Assign item to mean
--> Update mean by shifting it to the average of the items in that cluster

Example

Load the Datasets

```
from sklearn.datasets import load_digits
digits_data = load_digits().data
```

Each handwritten digit in the data is an array of color values of pixels of its image. For better understanding, let's print how the data of the first digit looks like and then display its's respective image.

```
import matplotlib.pyplot as plt
print(digits_data[0])
sample_digit = digits_data[0].reshape(8, 8)
plt.imshow(sample_digit)
plt.title("Digit image")
plt.show()
```

In the next step, we scale the data. Scaling is an optional yet very helpful technique for the faster processing of the model. In our model, we scale the pixel values which are typically between 0 – 255 to -1 – 1, easing the computation and avoiding super large numbers. Another point to consider is that a train test split is not required for this model as it is unsupervised learning with no labels to test. Then, we define the k value, which is 10 as we have 0-9 digits in our data. Also setting up the target variable.

```
from sklearn.preprocessing import scale
scaled_data = scale(digits_data)
print(scaled_data)
Y = load_digits().target
print(Y)
```

Defining k-means clustering:

Now we define the K-means cluster using the KMeans function from the sklearn module.

Method 1: Using a Random initial cluster.

- Setting the initial cluster points as random data points by using the '**init**' argument.
- The argument '**n_init**' is the number of iterations the k-means clustering should run with different initial clusters chosen at random, in the end, the clustering with the least total variance is considered'
- The random state is kept to 0 (any number can be given) to fix the same random initial clusters every time the code is run.

```
from sklearn.cluster import KMeans
k = 10
kmeans_cluster = KMeans(init = "random", n_clusters = k, n_init = 10, random_state = 0)
```

Method 2: Using k-means++

It is similar to method-1 however, it is not completely random, and chooses the initial clusters far away from each other. Therefore, it should require fewer iterations in finding the clusters when compared to the random initialization.

```
kmeans_cluster = KMeans(init="k-means++", n_clusters=k, n_init=10, random_state=0)
```

Model Evaluation

We will use scores like silhouette score, time taken to reach optimum position, v_measure and some other important metrics.

```
def bench_k_means(estimator, name, data):
        initial_time = time()
        estimator.fit(data)
        print("Initial-cluster: " + name)
        print("Time taken: {0:0.3f}".format(time() - initial_time))
        print("Homogeneity: {0:0.3f}".format( metrics.homogeneity_score(Y, estimator.labels_)))
        print("Completeness: {0:0.3f}".format( metrics.completeness_score(Y, estimator.labels_)))
        print("V_measure: {0:0.3f}".format( metrics.v_measure_score(Y, estimator.labels_)))
        print("Adjusted random: {0:0.3f}".format( metrics.adjusted_rand_score(Y, estimator.labels_)))
        print("Adjusted mutual info: {0:0.3f}".format(metrics.adjusted_mutual_info_score(Y, estimator.labels_)))
        print("Silhouette: {0:0.3f}".format(metrics.silhouette_score(data, estimator.labels_, metric='euclidean',
                                        sample_size=300)))
```

We will now use the above helper function to evaluate the performance of our k means algorithm.

```
kmeans_cluster = KMeans(init="random", n_clusters=k, n_init=10, random_state=0)
bench_k_means(estimator=kmeans_cluster, name="random", data=digits_data)
kmeans_cluster = KMeans(init="k-means++", n_clusters=k,n_init=10, random_state=0)
bench_k_means(estimator=kmeans_cluster, name="random", data=digits_data)
```

Visualizing the K-means clustering for handwritten data:

- Plotting the k-means cluster using the scatter function provided by the matplotlib module.
- Reducing the large dataset by using Principal Component Analysis (PCA) and fitting it to the previously defined k-means++ model.
- Plotting the clusters with different colors, a centroid was marked for each cluster.

```
from sklearn.decomposition import PCA
import numpy as np
# Reducing the dataset
pca = PCA(2)
reduced_data = pca.fit_transform(digits_data)
kmeans_cluster.fit(reduced_data)
# Calculating the centroids
centroids = kmeans_cluster.cluster_centers_
label = kmeans_cluster.fit_predict(reduced_data)
unique_labels = np.unique(label)
# plotting the clusters:
plt.figure(figsize=(8, 8))
for i in unique_labels:
        plt.scatter(reduced_data[label == i, 0], reduced_data[label == i, 1], label=i)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', s=169, linewidths=3, color='k', zorder=10)
plt.legend()
plt.show()
```

Conclusion

From the graph, we can observe the clusters of the different digits are approximately separable from one another.

How To Make Clustering in Machine Learning

To cluster data in Scikit-Learn using Python, you must process the data, train multiple classification algorithms and evaluate each model to find the classification algorithm that is the best predictor for your data

1. **Load data**

You can load any labelled dataset that you want to predict on. For instance, you can use fetch_openml('mnist_784') on the Mnist dataset to practice.

2. **Explore the dataset**

Use python pandas functions such as df.describe() and df.isnull().sum() to find how your data need to be processed prior training

3. **Preprocess data**

Drop, fill or impute missing, or unwanted values from your dataset to make sure that you don't introduce errors or bias into your data. Use pandas get_dummies(), drop(), and fillna() functions alongside some sklearn's libraries such as SimpleImputer or OneHotEncoder to preprocess your data.

4. **Split data into training and testing dataset**

To be able to evaluate the accuracy of your models, split your data into training and testing sets using sklearn's train_test_split. This will allow to train your data on the training set and predict and evaluate on the testing set.

5. **Create a pipeline to train multiple clustering algorithms and hyper-parameters**

Run multiple algorithms, and for each algorithm, try various hyper-parameters. This will allow to find the best performing model and the best parameters for that model. Use GridSearchCV() and Pipeline to help you with these tasks

6. **Evaluate the machine learning model**

Evaluate the model on its precision with methods such as the homogeneity_score() and completeness_score() and evaluate elements such as the confusion_matrix() in Scikit-learn

Reference

https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means-708505d204eb

https://www.analyticsvidhya.com/blog/2021/05/k-mean-getting-the-optimal-number-of-clusters/

https://www.geeksforgeeks.org/clustering-distance-measures/

Questions

1. Consider the following three variables for 20 different basketball players: points, assists, and rebounds. Perform k-means clustering manually with K=2, using Euclidean distance. Show the working for one iteration in your Lab Observation Book by using Euclidean distance.

points	assists	rebounds
18.0	3.0	15
19.0	4.0	14
14.0	5.0	10
14.0	4.0	8
11.0	7.0	14
20.0	8.0	13
28.0	7.0	9
30.0	6.0	5
31.0	9.0	4
35.0	12.0	11
33.0	14.0	6
25.0	9.0	5
25.0	4.0	3
27.0	3.0	8
29.0	4.0	12
30.0	12.0	7
19.0	15.0	6
23.0	11.0	5

Write a Python function (without using the scikit-learn library) to create a DataFrame containing the three variables (points, assists, and rebounds) for 20 different basketball players.
Apply the K-means algorithm to identify clusters with K=1, 2, K=3, and K=4, using distance formulas such as Euclidean distance, Manhattan distance, and Minkowski distance. Perform the following tasks:
a. Create a scatter plot of the data points in blue.
b. Plot the clusters with data points in different colors for K=1, 2, 3, and 4 in separate graphs.
c. Create a plot showing the number of clusters on the x-axis and the Sum of Squared Errors (SSE) on the y-axis. Compute SSE for all iterations. Show the table of given data points against SSE for every iteration and use the total sum of SSE in the graph of K vs. SSE .
d. Show the optimal value of K using the Elbow method and mark the same in the graph.

2. Redo 1(a)-1(d) using Manhattan distance.

3. Redo 1(a)-1(d) using Minkowski distance.

Additional questions

1.Write a Python function (with scikit-learn library) to create a DataFrame containing the three variables (points, assists, and rebounds) for 20 different basketball players.
Apply the K-means algorithm to identify clusters with K=1, 2, K=3, and K=4, using distance formulas such as Euclidean distance, Manhattan distance, and Minkowski distance. Perform the following tasks:
a. Create a scatter plot of the data points in blue.
b. Plot the clusters with data points in different colors for K=1, 2, 3, and 4 in separate graphs.
c. Create a plot showing the number of clusters on the x-axis and the Sum of Squared Errors (SSE) on the y-axis. Compute SSE for all iterations. Show the table of given data points against SSE for every iteration and use the total sum of SSE in the graph of K vs. SSE .
d. Show the optimal value of K using the Elbow method and mark the same in the graph.

2. Redo 1(a)-1(d) using Manhattan distance.

3. Redo 1(a)-1(d) using Minkowski distance.