Week 3 Report

Summary of the Report

- Unsupervised learning
- Measuring distance
    - Distance metrics
        - Euclidean distance
        - Cosine distance
        - Mahalanobis distance
        - Cityblock/Manhattan distance
        - Minkowski distance
        - Jaccard distance
- Clustering algorithms
- How KMeans works
- Evaluation of clustering
    - External assessment
    - Internal assessment
    - Rand index
    - Purity
    - Mutual information
    - Silhouette coefficient
- Limitation of KMeans
- Clustering with KMeans++
    - Guarantee of KMeans++
- Other clustering algorithms
    - Hierarchical clustering
    - DBSCAN
    - Shape-based clustering
- KMeans clustering with python
- Evaluating performance of KMeans clustering
- DBSCAN and Hierarchical clustering

References for the report

- https://www.ibm.com/topics/unsupervised-learning
- https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa
- https://www.freecodecamp.org/news/8-clustering-algorithms-in-machine-learning-that-all-data-scientists-should-know/
- https://neptune.ai/blog/k-means-clustering
- https://www.geeksforgeeks.org/clustering-performance-evaluation-in-scikit-learn/
- https://dm.cs.tu-dortmund.de/en/mlbits/cluster-kmeans-limitations/
- https://towardsdatascience.com/understanding-k-means-k-means-and-k-medoids-clustering-algorithms-ad9c9fbf47ca

3.1 Unsupervised Machine Learning

Unsupervised machine learning is where the model learns from unlabeled data. This means that the model does not have any pre-existing knowledge about the data.

The two forms of unsupervised learning are:

- Clustering
- Dimensionality Reduction

3.2 Measuring Distances

Measuring similarity or distances between different data points is fundamental to many ML algorithms.

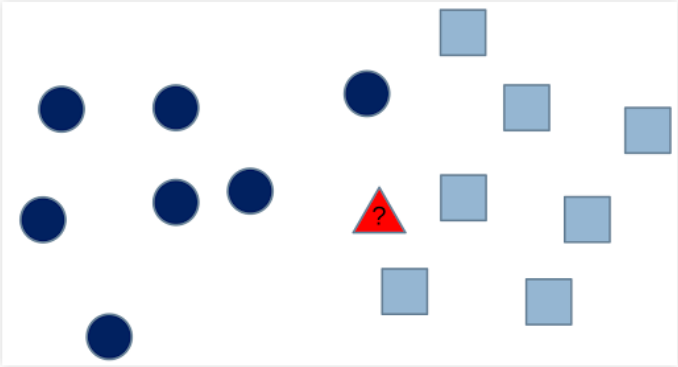
- **Distance Metrics**

  Machine learning algorithms frequently use distance measurements. Distance measurements are functions that specify the separation between any two occurrences of data.

  These include the below ML models:

  - Clustering algorithms
  - K-Nearest-Neighbor
  - Support vector machine (SVM)
  - Data visualization
  - Information retrieval
  - Ranking

Example 1: Nearest Neighbor Classification



We can use the distance to find the nearest neighbor and classify the data to the class label of this neighbor.

Example 2: Image retrieval

With the help of distance measurements, given a new image like the image of a cat, we can fetch all cat images from the dataset.

- **Types of Distance Measurements**

  The different types of distance measurements:

  - **Euclidean Distance**

    The normal straight-line distance between any two places in Euclidean space is known as the Euclidean Distance.

    $$d_{Euclidean}(\mathbf{x}_i, \mathbf{x}_j) = \left((\mathbf{x}_{i,1} - \mathbf{x}_{j,1})^2 + \ldots + (\mathbf{x}_{i,D} - \mathbf{x}_{j,D})^2\right)^{\frac{1}{2}}$$

  - **Cosine Distance**

    The cosine distance is a measure of similarity between two vectors in a multi-dimensional space. It is calculated as the cosine of the angle between the two vectors.

    $$d_{Cosine}(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \cdot \|\mathbf{x}_j\|_2}$$

  - **Mahalanobis Distance**

    The Mahalanobis distance (MD) is the distance between two points in multivariate space.

$$d_{Mahalanobis}(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j) M^{-1} (\mathbf{x}_i - \mathbf{x}_j)^T$$

- **Cityblock/Manhattan Distance**

  The cityblock distance, is a metric that measures the distance between two points in a grid. It is calculated by summing the absolute differences between the two points coordinates.

  $$d_{Cityblock}(\mathbf{x}_i, \mathbf{x}_j) = |\mathbf{x}_{i,1} - \mathbf{x}_{j,1}| + \ldots + |\mathbf{x}_{i,D} - \mathbf{x}_{j,D}|$$

- **Minkowski Distance**

  The minkowski distance defines a distance between two points in a normed vector space.

  $$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=0}^{n-1} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

- **Jaccard Distance**

  The Jaccard distance is a distance used to measure diversity of any two sets.

  $$d_{Jaccard}(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{|\mathbf{x}_i \cap \mathbf{x}_j|_1}{|\mathbf{x}_i \cup \mathbf{x}_j r|_1}$$

3.3 Clustering and its Applications

Data points are grouped together using clustering. When grouping unclassified material with unknown aims, it makes advantage of attributes similarities and differences. Due to its ability to handle large volumes of uncategorized data, it is particularly beneficial in unsupervised learning. However, because it forms group, it is also helpful in supervised learning.

The goals of clustering algorithms are to:

- Group unlabeled data objects with similar properties together
- Discover interesting perhaps unexpected clusters in the data
- Fina a valid or useful organization of the data

3.4 How Kmeans Works

In this algorithm, the center of each cluster is represented by K. We start with these centroids and measure each data point to determine which one is closest to the center. In other words, K-means stores k centroids to define clusters. If a point is closer to a specific cluster's centroid than any other centroid, it is said to be in that cluster. K-means searches for the best centroids by alternating between two methods:

1. Assigning data points to clusters based on the current defined centroids.
2. Choosing centroids based on the current assignment of data points to clusters.

Step 1 and 2 repeat until you find a useful grouping of data points.

3.5 Evaluation of Clustering

There are two main categories of evaluation methods for clustering:

- **External Assessment**

  Compare clustering performance against a known clustering.

- **Internal Assessment**

  Determine if clustering follows certain intrinsic assumptions.

  - Examples:
    - Silhouette Coefficient
    - Dunn index

Rand Index

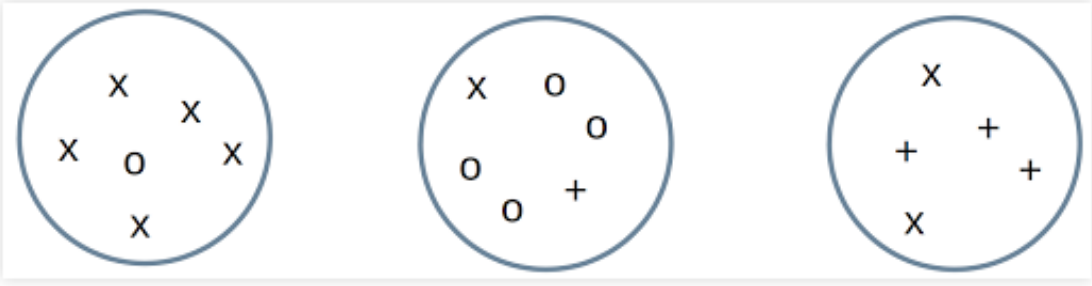
The Rand index is a measure of the similarity between two data clusters. The Rand index is a function that measures the similarity of the two assignments, ignoring their permutations.

The Rand index is computed as $R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}}$, where

- a = the number of pairs of data instances that are in the same cluster in both C, C'.
- b = the number of pairs of data instances that are in the different clusters in C and in different clusters in C'.
- c = the number of pairs of data instances that are in the same cluster in C but in different clusters in C'.
- d = the number of pairs of data instances that are in the different clusters in C but in the same clusters in C'.

Purity

Purity is a way of quality measurement in clustering methods. Purity measures the accuracy for all clusters in terms of class labels of the data in each cluster.



$$Purity = \frac{1}{17} \times (5 + 4 + 3) \approx 0.71$$

Mutual Information

Mutual information is a function that measures the agreement of the two clustering assignments in terms of how informative one is about the other, ignoring permutations.

$$MI(C, C') = \sum_{i=1}^{K} \sum_{j=1}^{K'} P(i, j) log \frac{P(i, j)}{P(i) P'(j)}$$

Silhouette Coefficient

The Silhouette value is a measure of how similar an object is to its own cluster compared to other clusters. The Silhouette coefficient contrasts the average distance between the instances of the same cluster with the average distance between the instances of different clusters.

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

3.6 Limitations of Kmeans

The most important limitations of Kmeans are:

- Random initialization means that you may get different clusters each time.
- We must supply the number of clusters beforehand.
- It cannot find clusters of arbitrary shapes.
- It cannot detect noisy data points.

Finding a useful number of clusters

- **Elbow Method**

  The Elbow Method is a method for finding the appropriate number of clusters. The Elbow method interprets and validates consistency within a cluster analysis to find the appropriate number of clusters in a dataset.

  $$SSE_K = \sum_{i=1}^{n} \sum_{k=1}^{K} z_{ik} ||x_i - \mu_k||^2$$

3.7 Clustering with Kmeans ++

Kmeans++ is an algorithm for choosing the initial cluster's centre values or centroids for the Kmeans clustering algorithm.

K-means begins by assigning cluster centers at random and then searches for better answers. K-means++ begins by assigning a cluster center at random and then looks for additional centers using the first one as a starting point.

- **Guarantee of Kmeans++:**
  - Every time the algorithm runs, it gets closer to (and not further away from) the best solution.
  - For the best solution, the objective function takes value $J_{optinum}$
  - Kmeans the objective function coverages to $J_{converged}$

$$\frac{E(J_{converged})}{J_{optimum}} \leq 8(\log K + 2)$$

3.8 Other Clustering algorithms

- **Hierarchical clustering**

  Hierarchical clustering finds clusters that have a predetermined order.

  There are two types of Hierarchical Clustering:

  - **Agglomerative Clustering (botton-up)**

    A bottom-up approach in which each observation starts in its own cluster, and pair of clusters are merged as one moves up the hierarchy.
  - **Divisive Clustering (top-down)**

    A top down approach in which all observations starts in one cluster, and splits are performed as one moves down the hierarchy.
- **DBSCAN (Density Based Spatial Clustering of Application with Noise)**

  DBSCAN is a clustering algorithm that clusters certain items in a group based on a given data points.
  - Calculate the distance from each points in the dataset to every other point. A point is considered a “Core Point” if it has at least the same number of data points within the defined distance.
  - Data points which cannot be considered core points but are under the defined distance of the core point are called border points. All other points are regarded as “Noise”.
  - The next step is to combine all core and border points within dis of each other into a single cluster.
- **Shape-based Clustering, VAT, iVAT**

- **VAT**

  Vat is a visualization technique that transforms the distance matrix of a dataset into a visual representation in the form of a reordered matrix. The reordering is done in such a way that the dissimilarities between the data points are emphasized in a way that reveals the underlying clustering structure of the data. If the data has a clear clustering structure, then the re-ordered matrix will exhibit block-like structures along the diagonal, indicating the presence of clusters.

- **iVAT**

  iVAT is an extension of VAT that involves repeatedly applying the VAT algorithm to the re-ordered matrix in order to refine the clustering structure.

3.9 Kmeans Clustering with Python

Load CSV file for KMeans Clustering

```
import pandas as pd
import numpy as np
digit_zero=pd.read_csv("data/digitData0.csv",header=None)
digit_one=pd.read_csv("data/digitData1.csv",header=None)
digit_zero.shape
digit_zero.head()
```

The output of the code is as follows

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	64
0	0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0	1.0
2	0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0	2.0
3	0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0	3.0
4	0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0	4.0

Merge two data frames and rename the header name

```
Feature_set=pd.concat([digit_zero,digit_one],join="inner")
```

Target as a ground truth of Kmean clustering

```
cols=["feature_"+str(i+1) for i in range(Feature_set.shape[1]-1)]
cols.append("Target")
Feature_set.columns=cols

Feature_set.head()
```

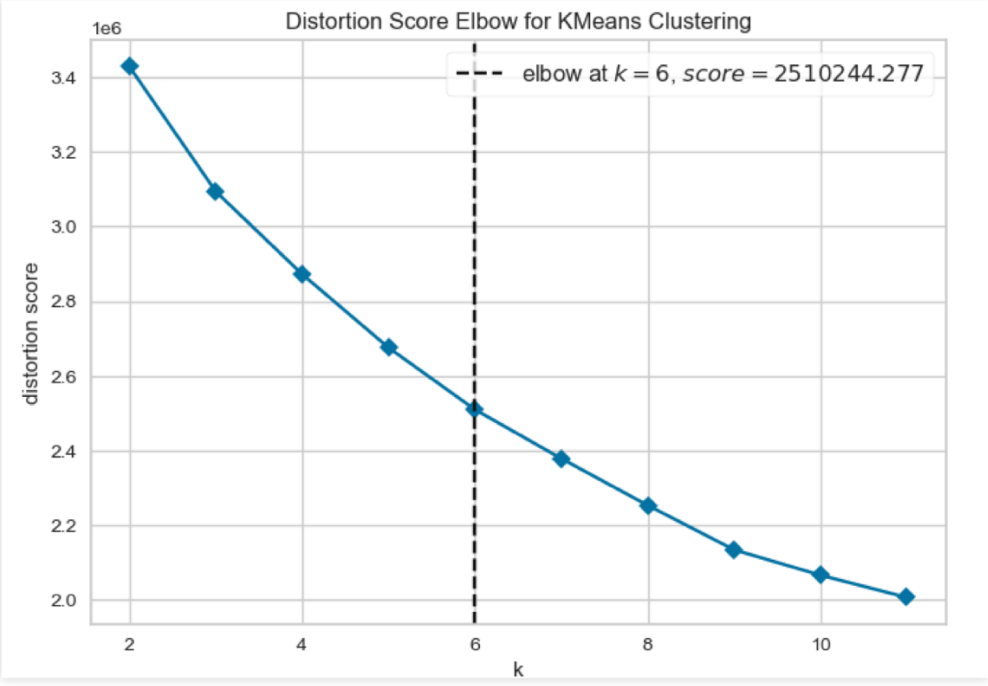
The output of the above code:

feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	...	feature_56	feature_57	feature_58	feature_59	feature_60	feature_61	feature_62	feature_63	feature_64	Target
0.0	0.0	5.0	13.0	9.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	6.0	13.0	10.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	12.0	13.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	11.0	16.0	10.0	0.0	0.0	1.0
0.0	0.0	0.0	4.0	15.0	12.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	3.0	11.0	16.0	9.0	0.0	2.0
0.0	0.0	7.0	15.0	13.0	1.0	0.0	0.0	0.0	8.0	...	0.0	0.0	0.0	7.0	13.0	13.0	9.0	0.0	0.0	3.0
0.0	0.0	0.0	1.0	11.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	2.0	16.0	4.0	0.0	0.0	4.0

Defining a number of clusters using Elbow number

```python
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

from yellowbrick.cluster import KElbowVisualizer

X=Feature_set.iloc[:,0:-1]
y=Feature_set.iloc[:,-1]

# Instantiate the clustering model and visualizer
model = KMeans()
visualizer = KElbowVisualizer(
    model, k=(2,12), metric='distortion', timings=False
) #distortion same as Euclidean distance

visualizer.fit(X)        # Fit the data to the visualizer
visualizer.show()
```

The output of the above code is:



3.10 Evaluating Performance of KMeans Clustering

Train and Test KMeans clustering

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

from yellowbrick.cluster import KElbowVisualizer

# Generate synthetic dataset with 8 random clusters
X=Feature_set.iloc[:,0:-1]
y_true=Feature_set.iloc[:,-1]
model = KMeans(n_clusters=2)
y_pred=model.fit_predict(X)
```

Performance Metrics

- Purity
- Mutual Information
- Silhouette Coefficient
- F1
- Recall
- Accuracy
- Precision

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred))
```

Performance of the KMeans:

	precision	recall	f1-score	support
0.0	0.00	0.01	0.00	320
1.0	0.02	0.06	0.03	327
2.0	0.00	0.00	0.00	318
3.0	0.00	0.00	0.00	328
4.0	0.00	0.00	0.00	323
5.0	0.00	0.00	0.00	325
6.0	0.00	0.00	0.00	323
7.0	0.00	0.00	0.00	320
8.0	0.00	0.00	0.00	313
9.0	0.00	0.00	0.00	321
accuracy			0.01	3218
macro avg	0.00	0.01	0.00	3218
weighted avg	0.00	0.01	0.00	3218

Purity

```python
from sklearn import metrics

def purity_score(y_true, y_pred):
    # compute contingency matrix (also called confusion matrix)
    contingency_matrix = metrics.cluster.contingency_matrix(y_true, y_pred)
    # return purity
    return np.sum(np.amax(contingency_matrix, axis=0)) / np.sum(contingency_matrix)
print(
    "purity_score: %0.3f"
    % purity_score(y_true, y_pred)
)
```

Purity of KMeans:

```
purity_score: 0.202
```

Markup

3.11DBSCAN and Hierarchical Clustering

DBSCAN

```python
from sklearn.cluster import DBSCAN
X=Feature_set.iloc[:,0:-1]
y_true=Feature_set.iloc[:,-1]
model = DBSCAN(eps=0.3, min_samples=10)
y_pred=model.fit_predict(X)

print(
    "purity_score: %0.3f"
    % purity_score(y_true, y_pred)
)
```

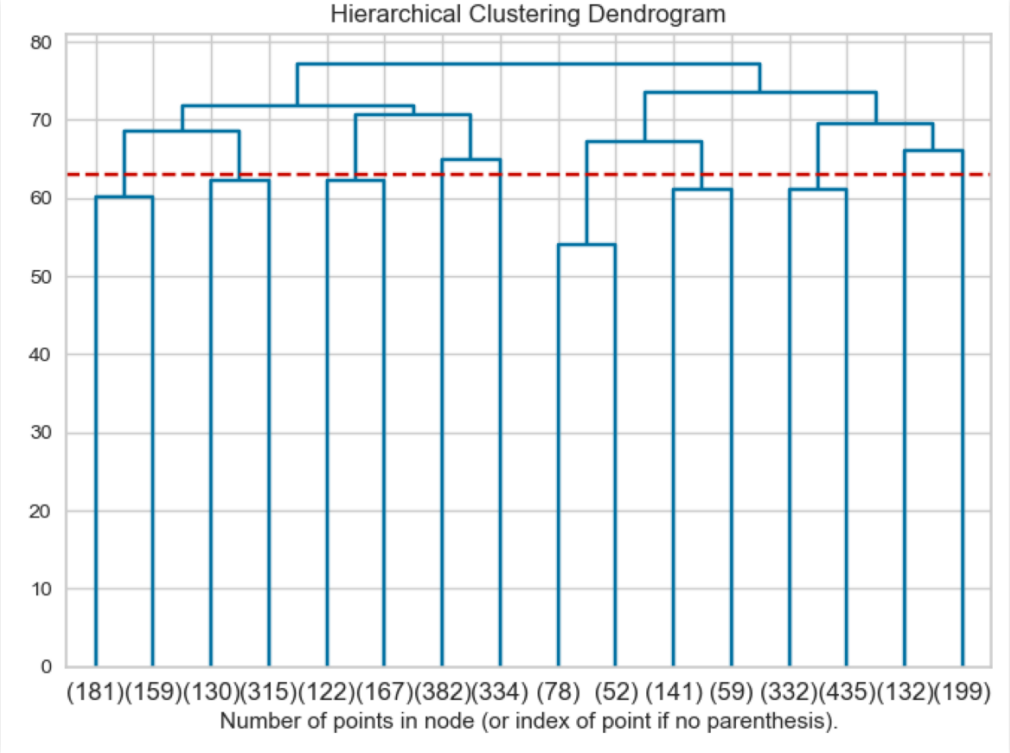
Performance of the DBSCAN model:

```
purity_score: 0.519
```

Markup

Hierarchical Clustering

```python
import numpy as np

from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering


def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)


X=Feature_set.iloc[:,0:-1]
y_true=Feature_set.iloc[:,-1]
# setting distance_threshold=0 ensures we compute the full tree.
model = AgglomerativeClustering(distance_threshold=0,linkage="complete", affinity="cosine",n_clusters=None)
model = model.fit(X)

plt.title("Hierarchical Clustering Dendrogram")
# plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode="level", p=3)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```



```
model = AgglomerativeClustering( linkage="complete", affinity="cosine",n_clusters=2)

y_pred = model.fit_predict(X)
print(
    "purity_score: %0.3f"
    % purity_score(y_true, y_pred)
)
```

Purity of the Hierarchical Clustering

```
purity_score: 0.519
```

Markup

QUIZ SCREENSHOT