Lab 4 Solution

April 19, 2024

1 LAB 04 - Python version

Luca Catalano, Daniele Rege Cambrin and Eleonora Poeta

1.0.1 Disclaimer

The purpose of creating this material is to enhance the knowledge of students who are interested in learning how to solve problems presented in laboratory classes using Python. This decision stems from the observation that some students have opted to utilize Python for tackling exam projects in recent years.

To solve these exercises using Python, you need to install Python (version 3.9.6 or later) and some libraries using pip or conda.

Here's a list of the libraries needed for this case:

- `os`: Provides operating system dependent functionality, commonly used for file operations such as reading and writing files, interacting with the filesystem, etc.
- `pandas`: A data manipulation and analysis library that offers data structures and functions to efficiently work with structured data.
- `numpy`: A numerical computing library that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- `matplotlib.pyplot`: A plotting library for creating visualizations like charts, graphs, histograms, etc.
- `sklearn`: Machine learning algorithms and tools.
- `sklearn_extra`: Additional machine learning algorithms and extensions.
- `nltk`: The Natural Language Toolkit, a library for natural language processing tasks such as tokenization, stemming, part-of-speech tagging, and more.
- `xlrd`: A Python library used for reading data and formatting information from Excel files (.xls and .xlsx formats). It provides functionality to extract data from Excel worksheets, including cells, rows, columns, and formatting details.

You can download Python from here and follow the installation instructions for your operating system.

For installing libraries using pip or conda, you can use the following commands:

- Using pip:

```
pip install pandas numpy matplotlib nltk scikit-learn xlrd scikit-learn-extra
```

- Using conda:

  ```
  conda install pandas numpy matplotlib nltk scikit-learn xlrd scikit-learn-extra
  ```

Make sure to run these commands in your terminal or command prompt after installing Python. You can also execute them in a cell of a Jupyter Notebook file (`.ipynb`) by starting the command with '!'.

2 Exercise 1

Import some libraries

[1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import TruncatedSVD
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

from sklearn_extra.cluster import KMedoids
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.cluster import DBSCAN



from sklearn.metrics import silhouette_score
```

2.1 Read file excel

To read the Excel file using a function integrated into the pandas library, you can use the `pd.read_excel()` function. Rewrite the instruction with the argument as the path of the file to be read

[2]:
```
dataset = pd.read_excel("/Users/luca/Library/Mobile Documents/
↪com~apple~CloudDocs/Business Intelligence per Big Data/Laboratories/LAB03/
↪Lab3Materiale/UsersSmall.xls")
```

In a Jupyter Notebook cell, you can print a subset of the representation by simply calling the name of the variable containing the DataFrame.

[3]:
```
dataset
```

[3]:
```
    Age         Workclass  Education      Marital Status         Occupation  \
0   -15         State-gov  Bachelors       Never-married       Adm-clerical
1   150           Private        PhD       Never-married    Exec-managerial
2    39         State-gov  Bachelors       Never-married       Adm-clerical
3    50  Self-emp-not-inc  Bachelors  Married-civ-spouse    Exec-managerial
```

```
4     38           Private    HS-grad            Divorced  Handlers-cleaners
..   …               …        …                  …               …
297   65           Private    HS-grad  Married-civ-spouse   Transport-moving
298   37      Self-emp-inc  Bachelors            Divorced             Sales
299   39                 ?    Masters  Married-civ-spouse                 ?
300   24           Private    HS-grad       Never-married       Craft-repair
301   38           Private    HS-grad            Divorced             Sales

      Relationship                Race     Sex Native Country  Response
0    Not-in-family               White    Male  United-States  Negative
1                ?               White  Female        Jamaica  Negative
2    Not-in-family               White    Male  United-States  Negative
3          Husband               White    Male  United-States  Negative
4    Not-in-family               White    Male  United-States  Negative
..             …                   …     …             …       …
297        Husband               White    Male  United-States  Negative
298  Not-in-family               White  Female  United-States  Negative
299           Wife  Asian-Pac-Islander  Female              ?  Negative
300      Own-child               White    Male  United-States  Negative
301  Not-in-family               White    Male  United-States  Negative

[302 rows x 10 columns]
```

2.2 How to handle Missing values?

Find if there are missing values.

Usually in a real dataset the missing values are stored with a nan value. In this case we have ? as missing values representation.

So first of all we can replace each '?' symbol in a nan value. The we will apply some important and classical functions.

[4]:
```
dataset.replace(to_replace = '?', value = np.nan, inplace = True)
```

[5]:
```
dataset.isnull().sum() # count the number of missing values for each column
```

[5]:
```
Age               0
Workclass        16
Education         0
Marital Status    0
Occupation       16
Relationship      1
Race              0
Sex               0
Native Country    8
Response          0
dtype: int64
```

As you have seen in class there are different methodologies for filling the nan values. Here we will use the average for the numerical data and the most frequent string for non-numerical columns

[6]:
```
# Replace NaN values with the average value for numerical columns
for col in dataset.select_dtypes(include=np.number).columns:
    dataset[col].fillna(dataset[col].mean(), inplace=True) # Get the average␣
 ↪value for the column and replace NaN values with it

# Replace NaN values with the most frequent value for non-numerical columns
for col in dataset.select_dtypes(exclude=np.number).columns:
    mode_val = dataset[col].mode()[0]  # Get the most frequent string value
    dataset[col].fillna(mode_val, inplace=True) # Get the most frequent value␣
 ↪for the column and replace NaN values with it
```

[7]:
```
dataset
```

[7]:

	Age	Workclass	Education	Marital Status	Occupation	\
0	-15	State-gov	Bachelors	Never-married	Adm-clerical	
1	150	Private	PhD	Never-married	Exec-managerial	
2	39	State-gov	Bachelors	Never-married	Adm-clerical	
3	50	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	
4	38	Private	HS-grad	Divorced	Handlers-cleaners	
..	…	…	…	…	…	
297	65	Private	HS-grad	Married-civ-spouse	Transport-moving	
298	37	Self-emp-inc	Bachelors	Divorced	Sales	
299	39	Private	Masters	Married-civ-spouse	Prof-specialty	
300	24	Private	HS-grad	Never-married	Craft-repair	
301	38	Private	HS-grad	Divorced	Sales	

	Relationship	Race	Sex	Native Country	Response
0	Not-in-family	White	Male	United-States	Negative
1	Husband	White	Female	Jamaica	Negative
2	Not-in-family	White	Male	United-States	Negative
3	Husband	White	Male	United-States	Negative
4	Not-in-family	White	Male	United-States	Negative
..	…	…	…	…	…
297	Husband	White	Male	United-States	Negative
298	Not-in-family	White	Female	United-States	Negative
299	Wife	Asian-Pac-Islander	Female	United-States	Negative
300	Own-child	White	Male	United-States	Negative
301	Not-in-family	White	Male	United-States	Negative

[302 rows x 10 columns]

We can now check the number of missing values in the dataset.

We can see that there are no missing values in the dataset.

[8]:
```
dataset.isnull().sum()
```

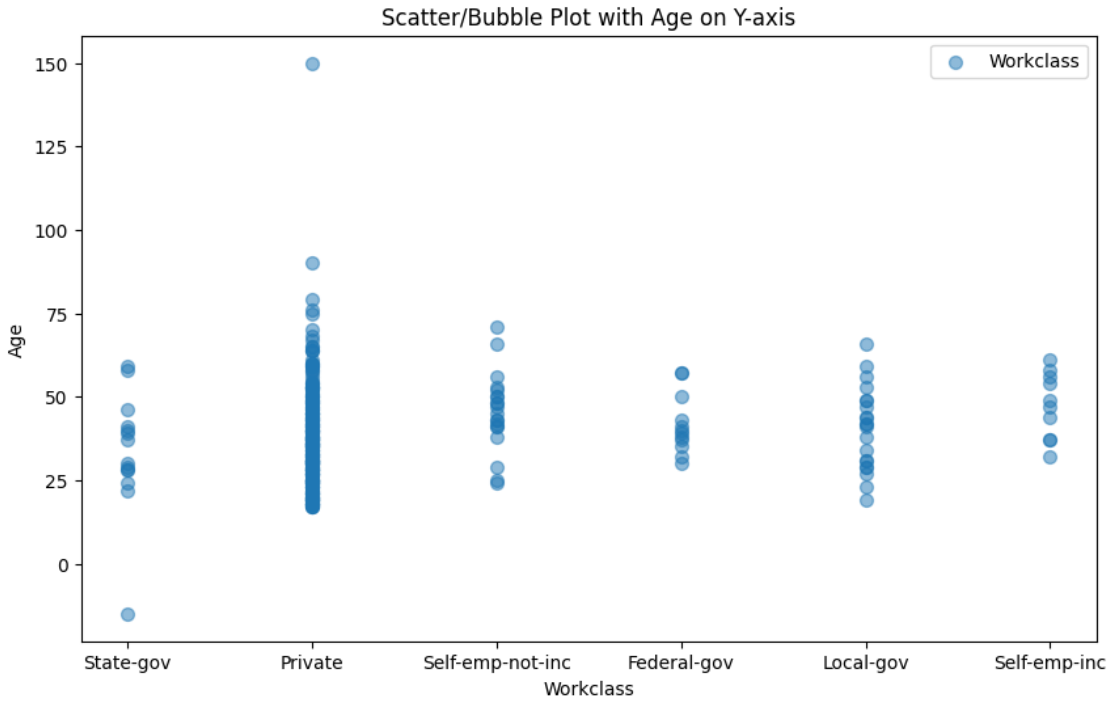
[8]:
```
Age               0
Workclass         0
Education         0
Marital Status    0
Occupation        0
Relationship      0
Race              0
Sex               0
Native Country    0
Response          0
dtype: int64
```

2.3 Outlier detection

You can plot a scatter/bubble plot to identify some outliers

[9]:
```
# Fix the 'Age' attribute on the y-axis
age_values = dataset['Age']

# Plot scatter/bubble plot with an attribute on the x-axis. Ypu caan choose␣
 ↪what ever attribute you want
attribute = "Workclass"
plt.figure(figsize=(10, 6))
plt.scatter(dataset[attribute], age_values, s=50, alpha=0.5, label=attribute)
plt.xlabel(attribute, fontsize=10)  # Adjusted x-axis label font size
plt.ylabel('Age')
plt.title('Scatter/Bubble Plot with Age on Y-axis')
plt.legend()
plt.show()
```



As evident, the 'Age' attribute in our dataset contains errors, such as improbable values like 150 for age or an age less than 0. To ensure the integrity of our data, we need to perform cleaning by filtering out such rows from the dataset.

[10]:
```
condition = (dataset['Age'] >= 0) & (dataset['Age'] < 101) # Get the condition␣
↪for the age values (between 0 and 105 years old)
dataset = dataset[condition].reset_index(drop=True) # Apply the condition to␣
↪the dataset and store the result in the dataset variable
```

2.4 Select attributes

Remove 'Response attribute' (that is in the last column) from the variable dataset

The `.iloc` function in Pandas is used for integer-location based indexing. It allows you to select rows and columns from a DataFrame by their integer position, rather than by label. This function provides a way to select data by position, similar to indexing in NumPy arrays.

2.4.1 Syntax

`DataFrame.iloc[row_indexer, column_indexer]`

- `row_indexer`: Specifies the rows to select. It can be:
  - An integer, e.g., `2`.
  - A list or array of integers, e.g., `[1, 3, 5]`.
  - A slice object with integers, e.g., `1:4`.
  - A boolean array.

- `column_indexer`: Specifies the columns to select. It follows the same rules as `row_indexer`.

2.4.2 Example Usage

```
import pandas as pd

# Creating a sample DataFrame
data = {'A': [1, 2, 3, 4],
        'B': [5, 6, 7, 8],
        'C': [9, 10, 11, 12]}
df = pd.DataFrame(data)

# Selecting specific rows and columns using iloc
selected_data = df.iloc[1:3, 0:2]
print(selected_data)
```

2.4.3 Output

```
   A  B
1  2  6
2  3  7
```

2.4.4 Notes

- `.iloc` is exclusive of the end index when using slices, similar to Python indexing conventions.
- If you want to select specific rows and columns by label instead of position, you should use the `.loc` function.

[11]:
```
dataset = dataset.iloc[:, :-1] # Remove the last column from the dataset
```

[12]:
```
dataset
```

[12]:

	Age	Workclass	Education	Marital Status	Occupation	Relationship	Race	Sex	Native Country
0	39	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States
1	50	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States
2	38	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States
3	53	Private	11th	Married-civ-spouse	Handlers-cleaners				
4	28	Private	Bachelors	Married-civ-spouse	Prof-specialty				
..	…	…	…	…	…				
295	65	Private	HS-grad	Married-civ-spouse	Transport-moving				
296	37	Self-emp-inc	Bachelors	Divorced	Sales				
297	39	Private	Masters	Married-civ-spouse	Prof-specialty				
298	24	Private	HS-grad	Never-married	Craft-repair				
299	38	Private	HS-grad	Divorced	Sales				

```
3          Husband                Black    Male  United-States
4             Wife                Black  Female           Cuba
..             …                    …     …              …
295        Husband                White    Male  United-States
296  Not-in-family                White  Female  United-States
297           Wife  Asian-Pac-Islander  Female  United-States
298      Own-child                White    Male  United-States
299  Not-in-family                White    Male  United-States

[300 rows x 9 columns]
```

2.5 Normalization

Normalize age attribute

[13]:
```
# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Normalize the 'age' attribute
dataset['Age'] = scaler.fit_transform(dataset['Age'].values.reshape(-1, 1)) #␣
 ↪Standardize the 'Age' column
```

Age in range [0-1]

[14]:
```
dataset
```

[14]:
```
          Age         Workclass  Education      Marital Status  \
0    0.301370         State-gov  Bachelors       Never-married
1    0.452055  Self-emp-not-inc  Bachelors  Married-civ-spouse
2    0.287671           Private    HS-grad            Divorced
3    0.493151           Private       11th  Married-civ-spouse
4    0.150685           Private  Bachelors  Married-civ-spouse
..        …                 …        …                  …
295  0.657534           Private    HS-grad  Married-civ-spouse
296  0.273973      Self-emp-inc  Bachelors            Divorced
297  0.301370           Private    Masters  Married-civ-spouse
298  0.095890           Private    HS-grad       Never-married
299  0.287671           Private    HS-grad            Divorced

            Occupation   Relationship                Race     Sex  \
0         Adm-clerical  Not-in-family               White    Male
1      Exec-managerial        Husband               White    Male
2    Handlers-cleaners  Not-in-family               White    Male
3    Handlers-cleaners        Husband               Black    Male
4       Prof-specialty           Wife               Black  Female
..                 …            …                  …     …
295   Transport-moving        Husband               White    Male
```

```
296              Sales  Not-in-family                White  Female
297     Prof-specialty           Wife  Asian-Pac-Islander  Female
298       Craft-repair      Own-child                White    Male
299              Sales  Not-in-family                White    Male

    Native Country
0    United-States
1    United-States
2    United-States
3    United-States
4             Cuba
..             …
295  United-States
296  United-States
297  United-States
298  United-States
299  United-States

[300 rows x 9 columns]
```

2.6 Kmedoids Clustering

Agglomerative clustering is a strategy of hierarchical clustering. Hierarchical clustering (also known as Connectivity based clustering) is a method of cluster analysis which seeks to build a hierarchy of clusters. Hierarchical clustering, is based on the core idea of objects being more related to nearby objects than to objects farther away.

[15]:
```
dataset_copy = dataset.copy()

label_encoder = LabelEncoder()
for column in dataset_copy.select_dtypes(include=['object']).columns:
    dataset_copy[column] = label_encoder.fit_transform(dataset_copy[column])

kmedoids = KMedoids(n_clusters=3)
# Fit the agglomerative clustering model to the data
cluster_labels = kmedoids.fit(dataset_copy)
cluster_labels.labels_
```

[15]:
```
array([0, 0, 0, 1, 2, 0, 1, 0, 2, 0, 0, 2, 0, 1, 0, 1, 0, 2, 1, 0, 2, 2,
       1, 1, 2, 2, 0, 2, 0, 0, 1, 2, 0, 0, 2, 1, 2, 0, 2, 1, 1, 2, 2, 0,
       0, 2, 2, 0, 0, 1, 0, 2, 2, 0, 0, 2, 1, 1, 0, 2, 2, 1, 2, 2, 2, 0,
       0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 0, 1, 1, 1, 2, 2, 2, 2, 0, 2, 2, 2,
       1, 2, 1, 0, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 2, 2, 0, 0, 1, 1, 0, 0,
       2, 2, 2, 2, 2, 1, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 2, 0, 1, 2,
       0, 2, 1, 2, 0, 0, 2, 0, 0, 2, 2, 1, 2, 2, 2, 2, 1, 2, 0, 0, 0, 0,
       2, 0, 0, 2, 0, 2, 1, 2, 0, 0, 2, 2, 0, 2, 1, 1, 2, 2, 2, 0, 2, 2,
       0, 0, 2, 2, 0, 0, 0, 1, 0, 0, 0, 2, 2, 2, 2, 2, 0, 2, 2, 1, 0, 0,
```

```
2, 2, 2, 2, 0, 2, 0, 1, 2, 0, 2, 1, 2, 0, 2, 0, 1, 0, 1, 2, 2, 1,
2, 1, 2, 0, 2, 2, 1, 0, 0, 0, 1, 2, 0, 0, 2, 2, 0, 0, 0, 2, 2, 2,
1, 2, 0, 0, 0, 2, 1, 0, 2, 0, 1, 0, 2, 0, 2, 2, 0, 0, 2, 2, 1, 2,
1, 0, 2, 0, 2, 0, 0, 1, 0, 2, 2, 0, 0, 0, 2, 2, 2, 0, 1, 2, 2, 2,
2, 2, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2])
```

Silhouette score is a metric used to evaluate the quality of clustering in unsupervised learning. It quantifies how similar an object is to its own cluster compared to other clusters. The silhouette score ranges from -1 to 1.

[16]:
```
silhouette_score(dataset_copy, kmedoids.labels_)
```

[16]:
```
0.2737583262844644
```

2.7 SVD

These codes use dimensionality reduction technique, SVD, to reduce the dimensionality of the dataset to 3 dimensions, and then visualize the data points in a 3D scatter plot. Each data point is colored according to its assigned cluster label obtained from a clustering algorithm (`cluster_labels.labels_`).

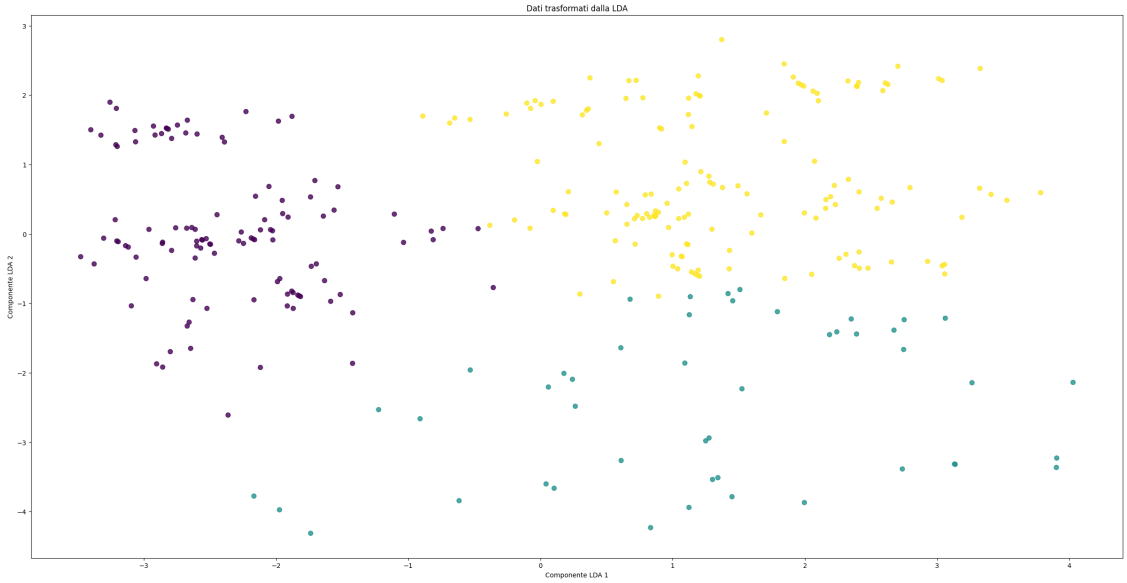
[19]:
```
svd = TruncatedSVD(n_components=2)
X_svd = svd.fit_transform(dataset_copy, kmedoids.labels_)

fig = plt.figure(figsize=(30, 15))
plt.scatter(X_svd[:, 0], X_svd[:, 1], c=kmedoids.labels_, cmap='viridis',␣
↪marker='o', s=50, alpha=0.8)
plt.title('Dati trasformati dalla LDA')
plt.xlabel('Componente LDA 1')
plt.ylabel('Componente LDA 2')
plt.show()
```

2.8 LDA

[20]:
```
lda = LDA(n_components=2)
X_lda = lda.fit_transform(dataset_copy, kmedoids.labels_)

# Plotting the data in 2D
fig = plt.figure(figsize=(30, 15))
plt.scatter(X_lda[:, 0], X_lda[:, 1], c=kmedoids.labels_, cmap='viridis',␣
 ↪marker='o', s=50, alpha=0.8)
plt.title('Dati trasformati dalla LDA')
plt.xlabel('Componente LDA 1')
plt.ylabel('Componente LDA 2')
plt.show()
```



2.9 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular clustering algorithm used in machine learning and data mining. It groups together points that are closely packed together based on their density in a high-dimensional space. Unlike other clustering algorithms, DBSCAN doesn't require the number of clusters to be specified in advance. Instead, it defines clusters as continuous regions of high density separated by regions of low density.

The key parameters of DBSCAN are:

- Epsilon ( ): A distance threshold that determines the neighborhood of a point.
- MinPts: The minimum number of points required to form a dense region (cluster).

DBSCAN works by iteratively exploring the neighborhood of each point. A point is classified as a core point if it has at least MinPts points within its -neighborhood. Core points are then used to expand clusters by adding neighboring points to the same cluster. Points that are not core points themselves but are within the -neighborhood of a core point are classified as border points and are included in the cluster. Points that are not core points and don't have enough neighboring points are considered noise and are not assigned to any cluster.

DBSCAN is particularly useful for clustering data with irregular shapes and handling noise effectively. It's robust to outliers and doesn't require specifying the number of clusters beforehand, making it suitable for various applications, including spatial data analysis, anomaly detection, and image segmentation. However, choosing appropriate values for and MinPts can be challenging and may significantly affect the clustering results.

[ ]:
```
dataset_copy = dataset.copy()
dataset_copy = pd.get_dummies(dataset_copy)
# Initialize DBSCAN
dbscan = DBSCAN(eps=1.0, min_samples=3)

# Fit DBSCAN
cluster_labels = dbscan.fit_predict(dataset_copy)

# Analyze the clusters
# For example, print the number of clusters and number of points in each cluster
num_clusters = len(set(cluster_labels)) - (1 if -1 in cluster_labels else 0)
print("Number of clusters:", num_clusters)
print("Number of noise points:", list(cluster_labels).count(-1))
```

```
Number of clusters: 8
Number of noise points: 270
```

Silhouette score is a metric used to evaluate the quality of clustering in unsupervised learning. It quantifies how similar an object is to its own cluster compared to other clusters. The silhouette score ranges from -1 to 1.

[ ]:
```
silhouette_score(dataset_copy, cluster_labels)
```

[ ]:
```
-0.1554790613038654
```

3 Exercise 2

Import some libraries

[ ]:
```
import os
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import cosine_similarity
```

3.1 Data preprocessing

Preprocess text files stored in a specific folder, tokenize them, remove stopwords, perform stemming, and then convert them into a TF-IDF (Term Frequency-Inverse Document Frequency) matrix using scikit-learn's TfidfVectorizer.

[ ]:
```
# Download NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

# Define the folder path containing the text files
folder_path = "/Users/luca/Library/Mobile Documents/com~apple~CloudDocs/
 ↪Business Intelligence per Big Data/Laboratories/LAB03/Lab3Materiale/
 ↪wikipedia"
# List to hold preprocessed text from each file
preprocessed_texts = []
# List to hold file names
file_names = []
# stemming
stemmer = SnowballStemmer("english")
# file with the stopwords
file_italian_stopwords = open("/Users/luca/Library/Mobile Documents/
 ↪com~apple~CloudDocs/Business Intelligence per Big Data/Laboratories/LAB03/
 ↪Lab3Materiale/stopwordsEnglish.txt")
italian_stopwords = set(file_italian_stopwords.read().splitlines())

# Loop through files in the folder
for file_name in os.listdir(folder_path):
    file_path = os.path.join(folder_path, file_name)
    with open(file_path, 'r', encoding='utf-8') as file:
        # Read the text from the file
        text = file.read()
        # Tokenization
        tokens = word_tokenize(text)
        # Transform to lowercase
        tokens_lower = [token.lower() for token in tokens]
        # Remove stopwords
        tokens_filtered = [token for token in tokens_lower if token not in␣
 ↪italian_stopwords]
        # Stemming
        tokens_stemmed = [stemmer.stem(token) for token in tokens_filtered]
        # Join the tokens back to form preprocessed text
```

```
        preprocessed_text = ' '.join(tokens_stemmed)
        # Append the preprocessed text to the list
        preprocessed_texts.append(preprocessed_text)
        # Append the file name to the list
        file_names.append(file_name)

# Initialize the TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
# Fit and transform the preprocessed text data
tfidf_matrix = tfidf_vectorizer.fit_transform(preprocessed_texts)

# Convert the TF-IDF matrix to a DataFrame for better visualization
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_vectorizer.
 ↪get_feature_names_out(), index=file_names)
tfidf_df
```

```
[nltk_data] Downloading package punkt to /Users/luca/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /Users/luca/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[ ]:

	10	1154	1300	1564	1642	16th	\
text9.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text8.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text6.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text7.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text5.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text4.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text12.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text1.txt	0.044282	0.000000	0.000000	0.044282	0.044282	0.000000	
text11.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text3.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.072377	
text2.txt	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
text10.txt	0.000000	0.030929	0.030929	0.000000	0.000000	0.000000	

	1777	17th	1809	1820s	…	without	word	\
text9.txt	0.000000	0.000000	0.000000	0.00000	…	0.031102	0.000000	
text8.txt	0.000000	0.000000	0.000000	0.00000	…	0.000000	0.000000	
text6.txt	0.000000	0.000000	0.000000	0.00000	…	0.019887	0.000000	
text7.txt	0.000000	0.000000	0.000000	0.04408	…	0.000000	0.081651	
text5.txt	0.000000	0.000000	0.000000	0.00000	…	0.000000	0.039850	
text4.txt	0.000000	0.033360	0.000000	0.00000	…	0.000000	0.000000	
text12.txt	0.000000	0.000000	0.000000	0.00000	…	0.024365	0.000000	
text1.txt	0.044282	0.000000	0.044282	0.00000	…	0.074896	0.027342	
text11.txt	0.000000	0.000000	0.000000	0.00000	…	0.020116	0.000000	
text3.txt	0.000000	0.031079	0.000000	0.00000	…	0.000000	0.089378	
text2.txt	0.000000	0.000000	0.000000	0.00000	…	0.014780	0.016188	

```
text10.txt  0.000000  0.000000  0.000000  0.00000  …  0.000000  0.000000

               work     world   written      yale       yam      year  \
text9.txt   0.000000  0.170318  0.000000  0.000000  0.055168  0.112698
text8.txt   0.000000  0.023779  0.000000  0.000000  0.000000  0.000000
text6.txt   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
text7.txt   0.000000  0.027217  0.000000  0.088159  0.000000  0.000000
text5.txt   0.000000  0.039850  0.000000  0.000000  0.000000  0.000000
text4.txt   0.029468  0.000000  0.033360  0.000000  0.000000  0.026450
text12.txt  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
text1.txt   0.033594  0.000000  0.114091  0.000000  0.000000  0.030154
text11.txt  0.000000  0.044063  0.000000  0.000000  0.000000  0.024297
text3.txt   0.027454  0.000000  0.000000  0.000000  0.000000  0.000000
text2.txt   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
text10.txt  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000

              yield  zealand
text9.txt   0.000000  0.00000
text8.txt   0.000000  0.00000
text6.txt   0.000000  0.00000
text7.txt   0.000000  0.04408
text5.txt   0.000000  0.00000
text4.txt   0.000000  0.00000
text12.txt  0.000000  0.00000
text1.txt   0.000000  0.00000
text11.txt  0.000000  0.00000
text3.txt   0.000000  0.00000
text2.txt   0.000000  0.00000
text10.txt  0.030929  0.00000

[12 rows x 1096 columns]
```

3.2 KMeans

K-means is a popular clustering algorithm used in unsupervised machine learning for partitioning a dataset into a predetermined number of clusters. It aims to group similar data points together while maximizing the distance between different clusters. The algorithm iteratively assigns each data point to the nearest cluster centroid and recalculates the centroids based on the mean of the points in each cluster. This process continues until the centroids no longer change significantly, indicating convergence. K-means is sensitive to the initial placement of centroids, and different initializations can lead to different clustering results. Therefore, multiple runs with random initializations are often performed to mitigate this issue. While K-means is computationally efficient and easy to implement, it assumes spherical clusters and struggles with non-linear or irregularly shaped clusters. Additionally, it may not perform well with datasets of varying densities or clusters of unequal sizes. Despite these limitations, K-means remains widely used for clustering tasks in various domains due to its simplicity and scalability.

[ ]:
```
# K-Means clustering
k = 3  # Numero di cluster
max_iter = 50  # Numero massimo di iterazioni
kmeans = KMeans(n_clusters=k, max_iter=max_iter)
kmeans.fit(tfidf_matrix)

# Ottenere i centroidi dei cluster
centroids = kmeans.cluster_centers_

# Calcolare la similarità coseno tra i documenti e i centroidi dei cluster
similarity_to_centroids = cosine_similarity(tfidf_matrix, centroids)

# Assegnare i documenti ai cluster basati sulla massima similarità coseno
cluster_assignments = similarity_to_centroids.argmax(axis=1)

# Stampa dei risultati
for i in range(k):
    print(f"Cluster {i + 1}:")
    for idx, testo in enumerate(tfidf_df.index):
        if cluster_assignments[idx] == i:
            print(f"- {testo}")
    print()
```

```
Cluster 1:
- text4.txt
- text1.txt
- text3.txt
- text2.txt

Cluster 2:
- text8.txt
- text6.txt
- text7.txt
- text5.txt

Cluster 3:
- text9.txt
- text12.txt
- text11.txt
- text10.txt
```