# Web Mining Lab - 10

**Aadhitya Swarnesh**

[redacted]

- 28 April 2021

## Question 1

**Use the IMDB Movie review dataset and perform the Clustering process and identify the popular terms in the clusters. Use the IMDB Movie review dataset and perform the Clustering process and identify the popular terms in the clusters.**

We will first obtain the dataset from Kaggle, we use the **IMDB 50k Movie Review Datase**t. We can directly import this and begin with our Classification procedure, but we will first observe the dataset, pre-process it and save it into a **Comma Separated File** format, so as to import it easily.

For the pre-processing part, we had observed that there were a few hyperlinks and other symbols, emoji's etc in the reviews denoting the emotion of the comment. We will just be removing such tokens, as they do not contribute to the sentiment. We will also perform other pre-processing steps like removing numbers, conversion to lower case, removing stop words, etc.

Now that the dataset is ready, we will proceed to the actual Classification Process, we will in this lab use the **K-Means Algorithm** to perform text analysis on the data to predict the sentiment that is wished to be conveyed. We will before using this, apply the **TF-IDF Vectorizer** on the reviews in order to form a large TF-IDF matrix which is then fed into the K-Means algorithm.

The implementation of these algorithms is done in the **Python Programming Language**. The implementation of the TF-IDF, and the K-Means is used from the sk-learn package which provides an extensive set of library functions for this purpose.

After the model has trained on the TF-IDF, we have tested it on a few sentences, and the results look promising. They have been shown below in the results section. We have used 2 clusters because the model also has 2 classes of output — positive or negative. We have also found and displayed the top words in all the clusters.

## Result :

After the training process, the following sentences were fed into the model for testing and we have obtained accurate results :

```
["tf and idf is awesome!", "bad movie"]
array([1, 0], dtype=int32)
```

This output shows that the first sentence denotes a positive sentiment and the second denotes a negative sentiment.

The next image shows a list of the top words of the clusters of the trained K-Means model.

```
Top terms per cluster:
Cluster 0:
 movie
 bad
 like
 movies
 one
 good
 really
 even
 see
 would
```

```
Cluster 1:
 film
 one
 movie
 like
 good
 story
 time
 well
 show
 would
```

## Code :

The **code** that has delivered these results are as follows —

# Question

Use the IMDB Movie review dataset and perform theClustering process and identify the popular terms in the clusters. Use the IMDB Movie review dataset and perform the Clustering process and identify the popular terms in the clusters.

# Movie Review Dataset

In this notebook, we will take up the task of performing a `K Means` Clustering process on the `IMDB 50K Moovie Review Dataset` , and identify the most popular terms in the clusters.

- Reference (https://medium.com/@MSalnikov/text-clustering-with-k-means-and-tf-idf-f099bcf95183)

In [31]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import homogeneity_score

import numpy as np
import pandas as pd

from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')
nltk.download('punkt')

import re

import pickle
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

## 1. Import the Dataset

In this section, we import the dataset and analyse it.

In [4]:

```
df = pd.read_csv("IMDB Dataset.csv")
df.head()
```

Out[4]:

| | review | sentiment |
|---|---|---|
| **0** | One of the other reviewers has mentioned that ... | positive |
| **1** | A wonderful little production. <br /><br />The... | positive |
| **2** | I thought this was a wonderful way to spend ti... | positive |
| **3** | Basically there's a family where a little boy ... | negative |
| **4** | Petter Mattei's "Love in the Time of Money" is... | positive |

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

In [6]:

```
df.describe()
```

Out[6]:

| | review | sentiment |
|---|---|---|
| **count** | 50000 | 50000 |
| **unique** | 49582 | 2 |
| **top** | Loved today's show!!! It was a variety and not... | negative |
| **freq** | 5 | 25000 |

In [7]:

```
# Convert the sentiment column into numerical values

sentiment_map = {
    'positive' : 0,
    'negative' : 1
}

df['sentiment'] = [sentiment_map[item] for item in df['sentiment']]
```

In [8]:

```
# Get the number of output class

num_target_classes = len(df['sentiment'].unique())
```

In [9]:

```
df.head()
```

Out[9]:

| | review | sentiment |
|---|---|---|
| **0** | One of the other reviewers has mentioned that ... | 0 |
| **1** | A wonderful little production. <br /><br />The... | 0 |
| **2** | I thought this was a wonderful way to spend ti... | 0 |
| **3** | Basically there's a family where a little boy ... | 1 |
| **4** | Petter Mattei's "Love in the Time of Money" is... | 0 |

# 2. Pre-Processing Function

In [10]:

```
def preprocess(review) :

    # Remove HTML tags
    TAG_RE = re.compile(r'<[^>]+>')
    review = TAG_RE.sub('', review)

    # Remove punctuations and numbers
    review = re.sub('[^a-zA-Z]', ' ', review)

    # Single character removal
    review = re.sub(r"\s+[a-zA-Z]\s+", ' ', review)

    # Removing multiple spaces
    review = re.sub(r'\s+', ' ', review)

    # Convert to lower case
    review = review.lower()

    # Delete extra spaces
    review = review.strip()

    # Delete stop words
    stop_words = set(stopwords.words("english"))
    words = nltk.tokenize.word_tokenize(review)
    filtered_words = [word for word in words if word not in stop_words]
    review = " ".join(filtered_words)

    # Return the processed text
    return review
```

## 3. TF-IDF Vectorization

In [11]:

```
# Initialzie the vectorizer instance with the pre-processing function written ab
ove

tfidf_vectorizer = TfidfVectorizer(preprocessor=preprocess)
```

In [12]:

```
# Convert the reviews in the dataset into an array to feed into this vectorizer

reviews_list = df['review'].tolist()
```

In [13]:

```
# Apply the vectorizer on the reviews

tfidf = tfidf_vectorizer.fit_transform(reviews_list)
```

## 4. K Means Clustering

In [14]:

```
# Initialize the model

model = KMeans(n_clusters=num_target_classes)
```

In [15]:

```
# Fit the model on the prepared tfidf

model.fit(tfidf)
```

Out[15]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=30
0,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='a
uto',
       random_state=None, tol=0.0001, verbose=0)
```

In [16]:

```
# Save the trained model

pickle.dump(model, open("q1_model.pkl", "wb"))
```

## 5. Make Predictions

In this section, we can use the trained model to make predictions for any text we write.

In [24]:

```
# Enter a review

predict_reviews = ["tf and idf is awesome!", "bad movie"]
```

In [25]:

```
# Make the predictions

model.predict(tfidf_vectorizer.transform(predict_reviews))
```

Out[25]:

```
array([1, 0], dtype=int32)
```

We can notice here that the model has accurately predicted the class of the first sentence as `positive` and the second as `negative` .

## 6. Identify top terms

In this section, we identify the top terms in each cluster, to understand the trends of the dataset

In [30]:

```
print("Top terms per cluster:")

order_centroids = model.cluster_centers_.argsort()[:, ::-1]

terms = tfidf_vectorizer.get_feature_names()
for i in range(num_target_classes):
    print("Cluster %d:" % i)
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind])
    print("-----------------------------")
```

```
Top terms per cluster:
Cluster 0:
 movie
 bad
 like
 movies
 one
 good
 really
 even
 see
 would
-----------------------------
Cluster 1:
 film
 one
 movie
 like
 good
 story
 time
 well
 show
 would
-----------------------------
```

In [ ]:

# Question 2

## Perform Agglomerative Clustering and plot dendrogram for the Credit Card Fraud dataset.

We will first obtain the dataset from Kaggle, we use the **Credit Card Dataset**. We can directly import this and begin with our Classification procedure, but we will first observe the dataset, pre-process it and save it into a **Comma Separated File** format, so as to import it easily.

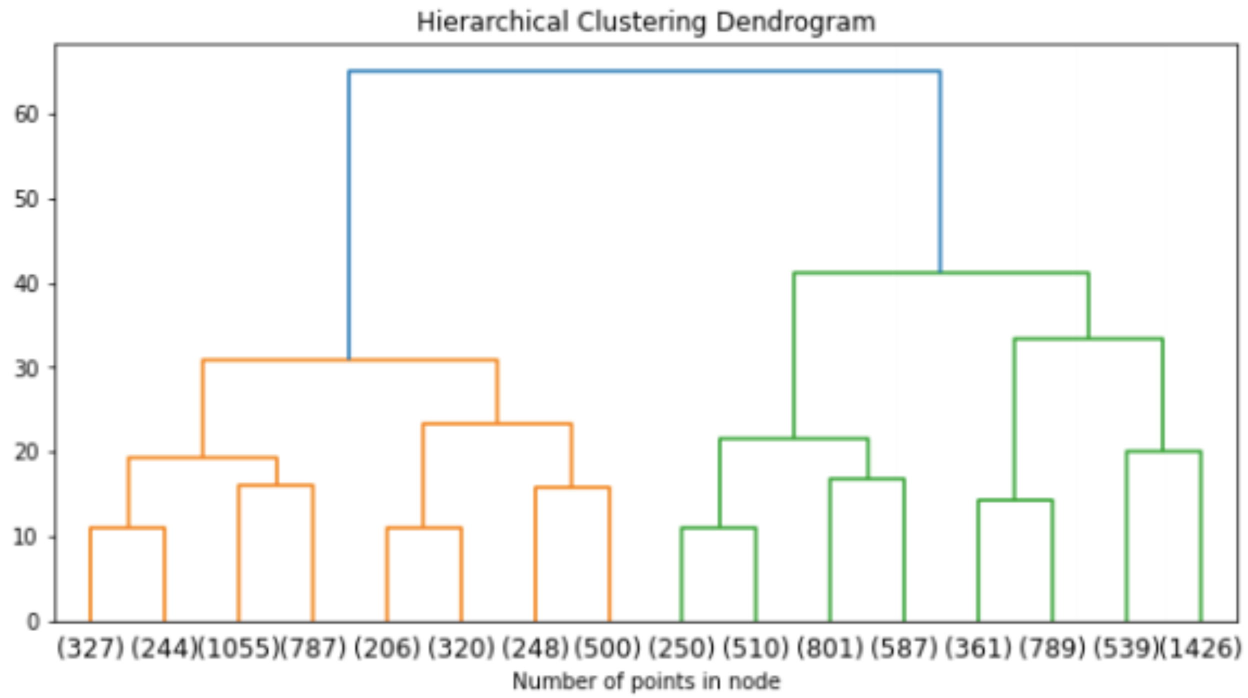
We have analysed the dataset and have performed the necessary pre-processing steps like filling NULL values by the mean value of the column, dropping the columns which do not contribute to the predictions, perform log transformations for better analysis and model training accuracy, normalisation and standardisation of the values.

Now that the dataset is ready, we will proceed to the actual Classification Process, we will in this lab use the **Agglomerative Clustering Algorithm** which is a type of Hierarchical Clustering to perform text analysis on the data. The implementation of these algorithms is done in the **Python Programming Language**. The implementation of the Agglomerative clustering is used from the **sk-learn** package which provides an extensive set of library functions for this purpose.

We have then used the Matplotlib library and the SciPy library to plot the dendrogram for the hierarchical clustering done previously. We can notice that the SciPy library has a dendrogram function which is used after forming the linkage matrix from the model trained before. The dendrogram is shown in the results section below.

# Result :

After the Agglomerative Clustering model is trained and fit on the dataset, we can use the SciPy library to plot the dendrogram for the model trained. The dendrogram for the model trained on this credit card dataset looks as follows :




# Code :

The **code** that has delivered these results are as follows —

# Agglomerative Clustering

In this Notebook, we will be performing an `Agglomerative Clustering` on the `Credit Card Dataset`.

In [30]:

```
import numpy as np
import pandas as pd

import sklearn.preprocessing as pp
from sklearn.cluster import AgglomerativeClustering

from scipy.cluster.hierarchy import dendrogram
import matplotlib.pyplot as plt
```

## 1. Import the Dataset

In [17]:

```
df = pd.read_csv("CC GENERAL.csv")
df.head()
```

Out[17]:

| | CUST_ID | BALANCE | BALANCE_FREQUENCY | PURCHASES | ONEOFF_PURCHASES | INSTA |
|---|---|---|---|---|---|---|
| **0** | C10001 | 40.900749 | 0.818182 | 95.40 | 0.00 | |
| **1** | C10002 | 3202.467416 | 0.909091 | 0.00 | 0.00 | |
| **2** | C10003 | 2495.148862 | 1.000000 | 773.17 | 773.17 | |
| **3** | C10004 | 1666.670542 | 0.636364 | 1499.00 | 1499.00 | |
| **4** | C10005 | 817.714335 | 1.000000 | 16.00 | 16.00 | |

## 2. Data Preprocessing

In this section, we will prepare the dataset for the clustering process in the further sections.

In [18]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   CUST_ID                           8950 non-null   object 
 1   BALANCE                           8950 non-null   float64
 2   BALANCE_FREQUENCY                 8950 non-null   float64
 3   PURCHASES                         8950 non-null   float64
 4   ONEOFF_PURCHASES                  8950 non-null   float64
 5   INSTALLMENTS_PURCHASES            8950 non-null   float64
 6   CASH_ADVANCE                      8950 non-null   float64
 7   PURCHASES_FREQUENCY               8950 non-null   float64
 8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 11  CASH_ADVANCE_TRX                  8950 non-null   int64  
 12  PURCHASES_TRX                     8950 non-null   int64  
 13  CREDIT_LIMIT                      8949 non-null   float64
 14  PAYMENTS                          8950 non-null   float64
 15  MINIMUM_PAYMENTS                  8637 non-null   float64
 16  PRC_FULL_PAYMENT                  8950 non-null   float64
 17  TENURE                            8950 non-null   int64  
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

## 2.1 Fill NULL Values

In [19]:

```
# Fill the null values in Minimum Payment with the average of the column

df['MINIMUM_PAYMENTS'].fillna(value=df['MINIMUM_PAYMENTS'].mean(), inplace = Tru
e)
```

In [20]:

```
# Fill the null values in Credit Limit with the average of the column

df['CREDIT_LIMIT'].fillna(value=df['CREDIT_LIMIT'].mean(), inplace = True)
```

## 2.2 Drop the ID Column

The `ID` column is not useful for Machine learning, and would only come in the way, so we remove this.

In [21]:

```
df = df.drop('CUST_ID', axis = 1)
```

## 2.3 Log Transformations

Many of the features have values that are either `1` or `0`. However the features that deal with dollar figures vary quite a bit. So we can choose to log transform these values to reduce the scale into a normal distribution. It also helps with scaling and grouping the data when analyzing the clusters.

In [22]:

```
cols = [
    'BALANCE',
    'PURCHASES',
    'ONEOFF_PURCHASES',
    'INSTALLMENTS_PURCHASES',
    'CASH_ADVANCE',
    'CASH_ADVANCE_TRX',
    'PURCHASES_TRX',
    'CREDIT_LIMIT',
    'PAYMENTS',
    'MINIMUM_PAYMENTS',
]

# We add 1 for each value to avoid "inf" values
df[cols] = np.log(1 + df[cols])
```

In [23]:

```
df.head()
```

Out[23]:

| | BALANCE | BALANCE_FREQUENCY | PURCHASES | ONEOFF_PURCHASES | INSTALLMENTS_PU |
|---|---|---|---|---|---|
| **0** | 3.735304 | 0.818182 | 4.568506 | 0.000000 | |
| **1** | 8.071989 | 0.909091 | 0.000000 | 0.000000 | |
| **2** | 7.822504 | 1.000000 | 6.651791 | 6.651791 | |
| **3** | 7.419183 | 0.636364 | 7.313220 | 7.313220 | |
| **4** | 6.707735 | 1.000000 | 2.833213 | 2.833213 | |

## 2.4 Normalize and Standardize the values

In [24]:

```
df_columns = df.columns.tolist()
```

In [25]:

```
# Scaling

scaler = pp.StandardScaler()
df = scaler.fit_transform(df)
```

In [26]:

```
# Normalization

df = pd.DataFrame(
    pp.normalize(df),
    columns = df_columns
)
```

In [27]:

```
df.head()
```

Out[27]:

| | BALANCE | BALANCE_FREQUENCY | PURCHASES | ONEOFF_PURCHASES | INSTALLMENTS_PU |
|---|---|---|---|---|---|
| **0** | -0.372087 | -0.077008 | -0.035051 | -0.304744 | |
| **1** | 0.231440 | 0.032762 | -0.409714 | -0.240750 | |
| **2** | 0.199784 | 0.125461 | 0.145475 | 0.257184 | |
| **3** | 0.119529 | -0.194597 | 0.158344 | 0.242210 | |
| **4** | 0.091097 | 0.173988 | -0.237930 | -0.038388 | |

# 3. Model

In [28]:

```
# Initializing the model

model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
```

In [29]:

```
# Fit the model on the data frame

model.fit(df)
```

Out[29]:

```
AgglomerativeClustering(distance_threshold=0, n_clusters=None)
```

# 4. Plotting Dendrogram

In [33]:

```
counts = np.zeros(model.children_.shape[0])
n_samples = len(model.labels_)
```

In [34]:

```
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
```

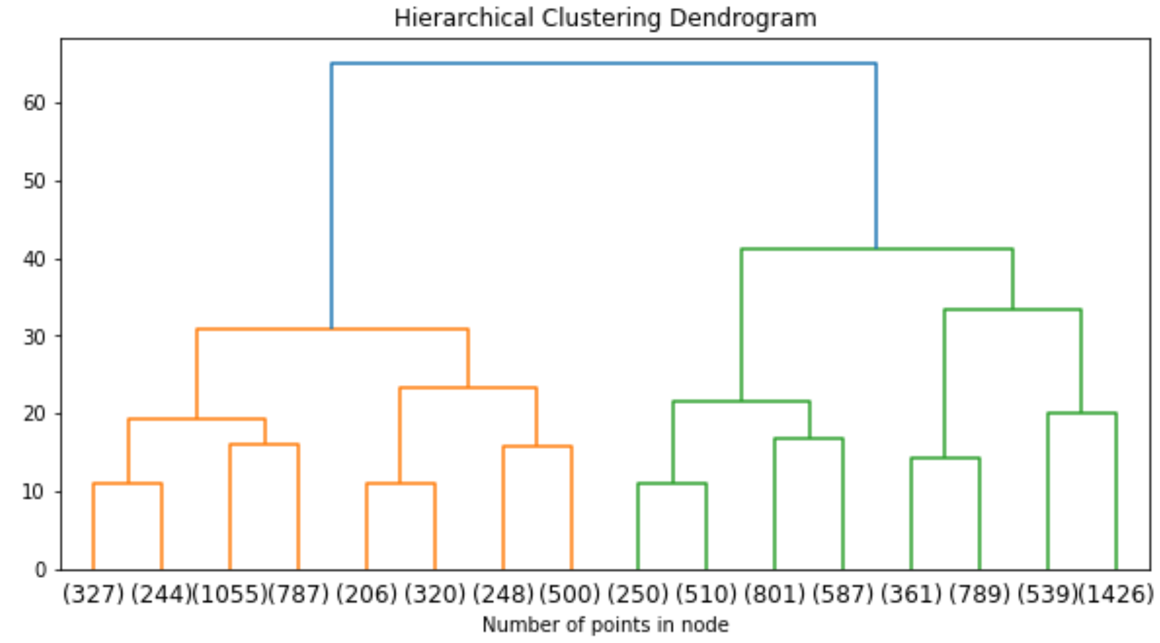
In [35]:

```
linkage_matrix = np.column_stack([model.children_, model.distances_, counts]).as
type(float)
```

In [81]:

```
# Plot the corresponding dendrogram

dendrogram(linkage_matrix, truncate_mode='level', p=3)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel("Number of points in node")
plt.rcParams['figure.figsize'] = [9, 5]
plt.show()
```




In [ ]: