UNIVERSITI TENAGA NASIONAL

The Energy University

CISB5123 Text Analytics
Lab 8
Text Clustering

Text clustering groups similar documents together based on their content, allowing you to discover patterns, trends, and insights within large collections of text data.

Any text clustering approach involves broadly the following steps:

- Text pre-processing: Text can be noisy, hiding information between stop words, inflexions and sparse representations. Pre-processing makes the dataset easier to work with.
- Feature Extraction: One of the commonly used techniques to extract the features from textual data is calculating the frequency of words/tokens in the document/corpus.
- Clustering: We can then cluster different text documents based on the features we have generated.

TEXT CLUSTERING USING TF-IDF VECTORIZER

Step 1: Import the libraries

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from tabulate import tabulate
from collections import Counter
```

Step 2: Create the documents

```
dataset = ["I love playing football on the weekends",
        "I enjoy hiking and camping in the mountains",
        "I like to read books and watch movies",
        "I prefer playing video games over sports",
        "I love listening to music and going to concerts"]
```

Step 3: Vectorize the dataset

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(dataset)
```

Step 4: Perform clustering

```
k = 2  # Define the number of clusters
km = KMeans(n_clusters=k)
km.fit(X)

# Predict the clusters for each document
y_pred = km.predict(X)

# Display the document and its predicted cluster in a table
table_data = [["Document", "Predicted Cluster"]]
table_data.extend([[doc, cluster] for doc, cluster in zip(dataset, y_pred)])
print(tabulate(table_data, headers="firstrow"))
```

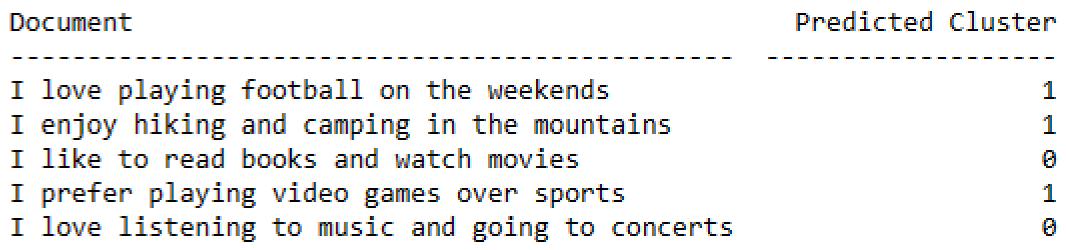
```
# Print top terms per cluster
print("\nTop terms per cluster:")
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names_out()
for i in range(k):
    print("Cluster %d:" % i)
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind])
    print()
```

Step 5: Evaluate results

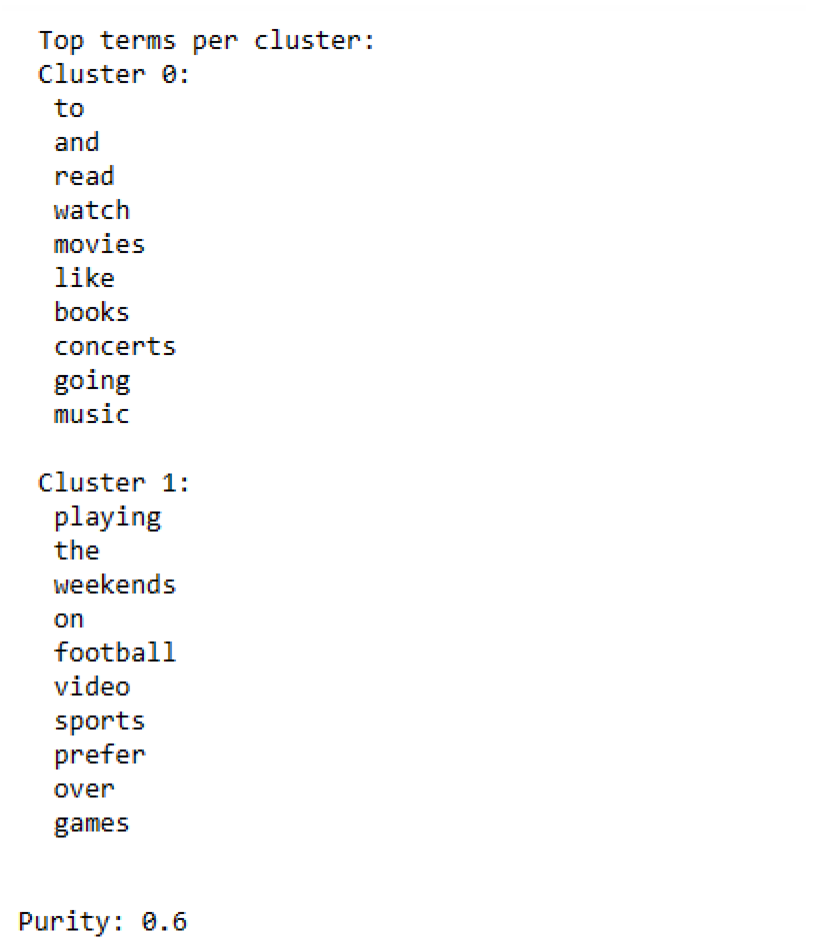
```
# Calculate purity
total_samples = len(y_pred)
cluster_label_counts = [Counter(y_pred)]
purity = sum(max(cluster.values()) for cluster in cluster_label_counts) / total_samples

print("Purity:", purity)
```

OUTPUT:



```
Document                                          Predicted Cluster
----------------------------------------------  -------------------
I love playing football on the weekends                           1
I enjoy hiking and camping in the mountains                       1
I like to read books and watch movies                             0
I prefer playing video games over sports                          1
I love listening to music and going to concerts                   0
```



```
 Top terms per cluster:
 Cluster 0:
  to
  and
  read
  watch
  movies
  like
  books
  concerts
  going
  music

 Cluster 1:
  playing
  the
  weekends
  on
  football
  video
  sports
  prefer
  over
  games


Purity: 0.6
```

TEXT CLUSTERING USING WORD2VEC VECTORIZER

Step 1: Import the libraries

```
import numpy as np
from sklearn.cluster import KMeans
from gensim.models import Word2Vec
from tabulate import tabulate
from collections import Counter
```

Step 2: Create the documents

```
dataset = ["I love playing football on the weekends",
          "I enjoy hiking and camping in the mountains",
          "I like to read books and watch movies",
          "I prefer playing video games over sports",
          "I love listening to music and going to concerts"]
```

Step 3: Train Word2Vec model

```
tokenized_dataset = [doc.split() for doc in dataset]
word2vec_model = Word2Vec(sentences=tokenized_dataset, vector_size=100, window=5, min_count=1, workers=4)
```

Step 4: Create document embeddings

```
X = np.array([np.mean([word2vec_model.wv[word] for word in doc.split() if word in word2vec_model.wv], axis=0) for doc in dataset])
```

Step 5: Perform clustering

```
k = 2  # Define the number of clusters
km = KMeans(n_clusters=k)
km.fit(X)

# Predict the clusters for each document
y_pred = km.predict(X)

# Tabulate the document and predicted cluster
table_data = [["Document", "Predicted Cluster"]]
table_data.extend([[doc, cluster] for doc, cluster in zip(dataset, y_pred)])
print(tabulate(table_data, headers="firstrow"))
```

Step 5: Evaluate results

```
# Calculate purity
total_samples = len(y_pred)
cluster_label_counts = [Counter(y_pred)]
purity = sum(max(cluster.values()) for cluster in cluster_label_counts) / total_samples

print("Purity:", purity)
```

EXERCISE:

1. Modify the codes for both TF-IDF & Word2Vec vectorizer by adding text preprocessing steps.

   Do the Purity differ when applying text preprocessing before vectorization?

2. Perform text clustering on 'customer_complaints_1.csv' dataset, specifically the Text column.