Contents

Introduction

Based on the selected dataset, A Million News Headlines Dataset, formulate, design and implement a machine learning solution, evaluate its performance, possibly improve the method.

Because of computational resource problems, the dataset will downsize to the first 50K headlines (~5% samples).

Objectives

Based on the Million News Headlines Dataset, we decided to cluster the 50K headlines. We aimed at developing a machine learning application in Python to cluster the headlines. The input of the application will be the headline dataset, and the output will be the headlines with cluster labels.

With the above objectives, we will use different natural language processing model and approaches to complete the task.

Evaluation Method

To evaluate the clustering result of our model, we decided to use the most common evaluation method, Silhouette Score, to check the performance of our model.

Silhouette score

Silhouette score is a metric used to evaluate the quality of clustering results. Its range is from -1 to +1, where +1 indicates that the sample is well-clustered and -1 is badly clustered.

Scikit library provided a function (silhouette_score()) to calculate the silhouette score of the clustering result. We applied this function to check every clustering results of different models and approaches.

Best silhouette score	0.73011 [Tfidf + PCA + KMeans(11)]

Libraries

```
import pandas as pd
import os
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.stem.snowball import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from gensim.models import Word2Vec
import numpy as np

nltk.download('punkt_tab')
nltk.download('stopwords')
```

We imported the above libraries for all of our processes.

Pandas

A common library for data manipulation and analysis. Mainly used to store the headlines data.

Os

Standard library to access and interact with operating system. Mainly for accessing the dataset file stored locally.

NLTK

A natural language processing library for tokenization, stemming, lemmatization and stopword handling.

Matplotlib

A library for plotting. It is used when visualizing the result of clustering.

Scikit

A machine learning library for data mining and analysis. Mainly for converting the headlines into a matrix of TF-IDF features and the major usage for the model.

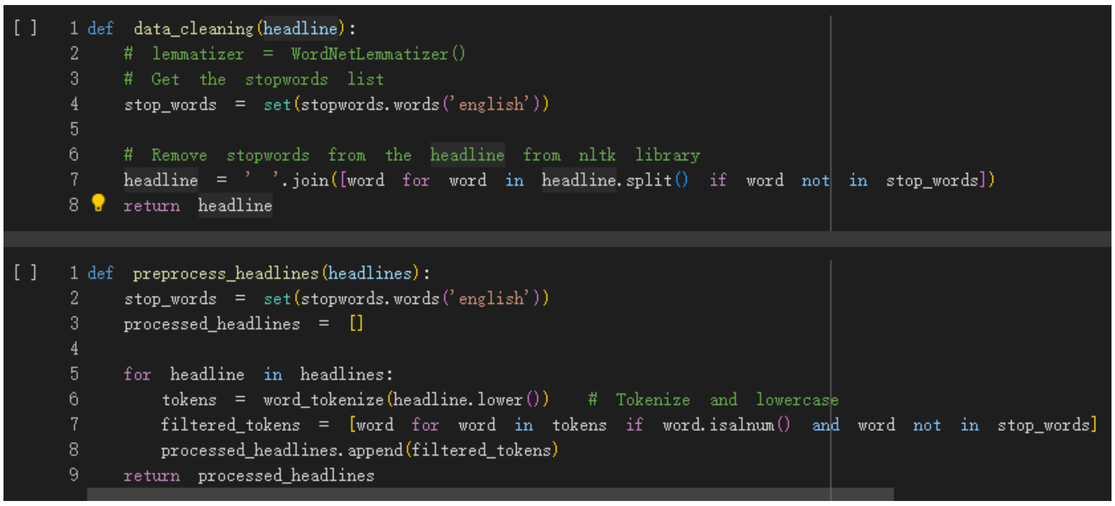
Gensim

A library for topic modeling and document similarity analysis. It is imported for the Word2Vec calculation.

Numpy

It provides functions for mathematical calculations among the dataset.

Data Cleaning



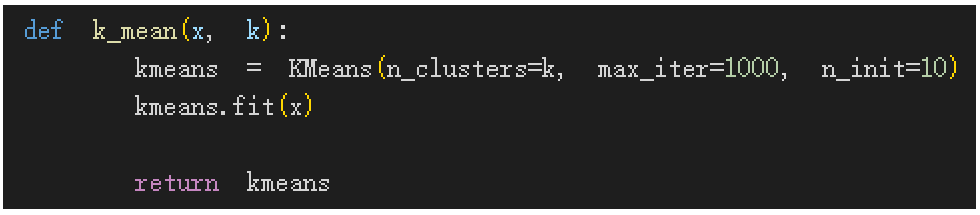
```
[ ]  def data_cleaning(headline):
         # lemmatizer = WordNetLemmatizer()
         # Get the stopwords list
         stop_words = set(stopwords.words('english'))

         # Remove stopwords from the headline from nltk library
         headline = ' '.join([word for word in headline.split() if word not in stop_words])
         return headline
```

```
[ ]  def preprocess_headlines(headlines):
         stop_words = set(stopwords.words('english'))
         processed_headlines = []

         for headline in headlines:
             tokens = word_tokenize(headline.lower())   # Tokenize and lowercase
             filtered_tokens = [word for word in tokens if word.isalnum() and word not in stop_words]
             processed_headlines.append(filtered_tokens)
         return processed_headlines
```

Data cleaning is an important procedure before doing the actual clustering. Based on the given dataset, the headlines are already lowercase and without punctuation. We remove the stopwords of the headlines by importing the stopword list from nltk library.

Model

K-Means Clustering



```
def k_mean(x, k):
        kmeans = KMeans(n_clusters=k, max_iter=1000, n_init=10)
        kmeans.fit(x)

        return kmeans
```

The k-means algorithm clusters data by trying to separate samples in n groups of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. This algorithm requires the number of clusters to be specified. It scales well to large numbers of samples and has been used across a large range of application areas in many different fields.

n_clusters: default as 5

The number of clusters to form as well as the number of centroids to generate.

max_iter: default as 300

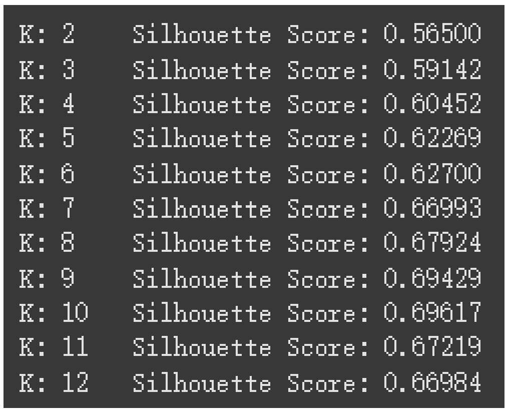
Maximum number of iterations of the k-means algorithm for a single run.

n_init: default as 10

Number of times the k-means algorithm is run with different centroid seeds.

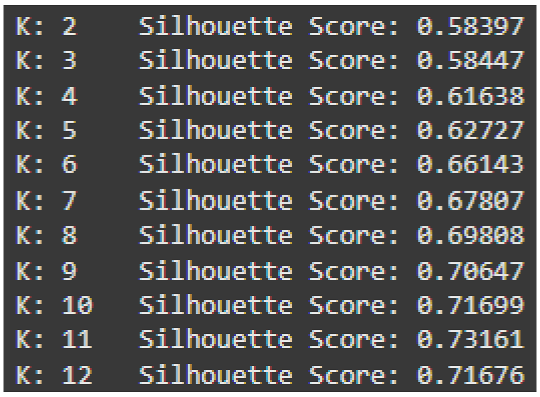
Use a for loop to find the optimal number of clusters (k)

Optimal K for LSA


```
K: 2    Silhouette Score: 0.56500
K: 3    Silhouette Score: 0.59142
K: 4    Silhouette Score: 0.60452
K: 5    Silhouette Score: 0.62269
K: 6    Silhouette Score: 0.62700
K: 7    Silhouette Score: 0.66993
K: 8    Silhouette Score: 0.67924
K: 9    Silhouette Score: 0.69429
K: 10   Silhouette Score: 0.69617
K: 11   Silhouette Score: 0.67219
K: 12   Silhouette Score: 0.66984
```

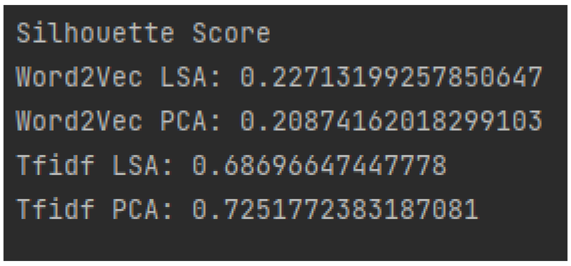
The best k is 10, provides the highest silhouette score (0.69617)

Optimal K for PCA


```
K: 2    Silhouette Score: 0.58397
K: 3    Silhouette Score: 0.58447
K: 4    Silhouette Score: 0.61638
K: 5    Silhouette Score: 0.62727
K: 6    Silhouette Score: 0.66143
K: 7    Silhouette Score: 0.67807
K: 8    Silhouette Score: 0.69808
K: 9    Silhouette Score: 0.70647
K: 10   Silhouette Score: 0.71699
K: 11   Silhouette Score: 0.73161
K: 12   Silhouette Score: 0.71676
```

The best k is 11, provides the highest silhouette score (0.73161)

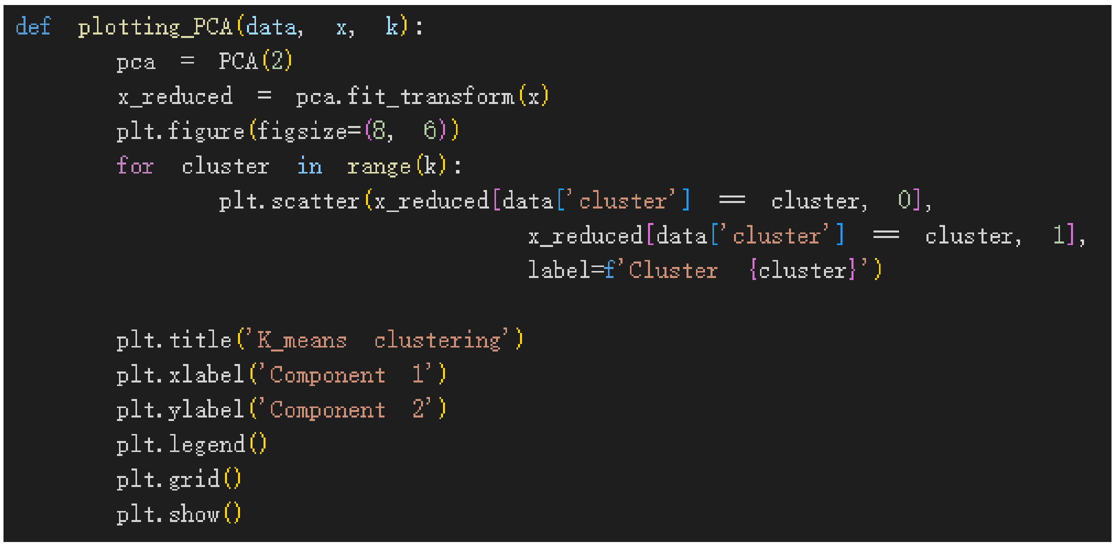
Hierarchical Clustering


```
Silhouette Score
Word2Vec LSA: 0.22713199257850647
Word2Vec PCA: 0.20874162018299103
Tfidf LSA: 0.68696647447778
Tfidf PCA: 0.7251772383187081
```

Besides K-Means Clustering, we tried Hierarchical Clustering to see if there is a better result. We used agglomerative method to do bottom-up clustering and end up with 10 clusters.

Improvement Method

Principal Component Analysis (PCA)

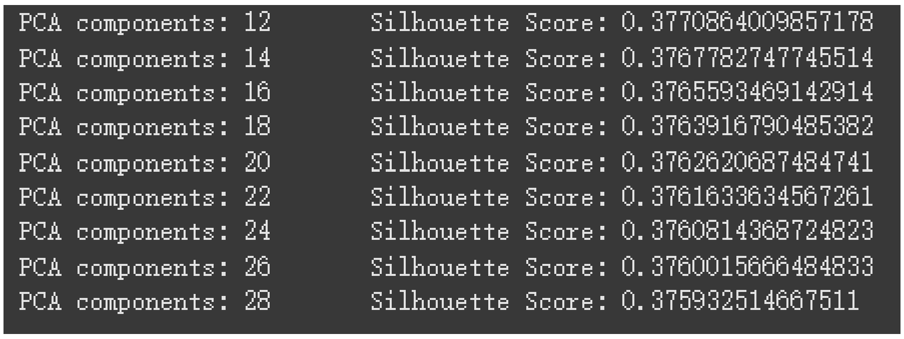


```
def  plotting_PCA(data,  x,  k):
        pca  =  PCA(2)
        x_reduced  =  pca.fit_transform(x)
        plt.figure(figsize=(8,  6))
        for  cluster  in  range(k):
                plt.scatter(x_reduced[data['cluster']  ==  cluster,  0],
                                        x_reduced[data['cluster']  ==  cluster,  1],
                                        label=f'Cluster  {cluster}')

        plt.title('K_means  clustering')
        plt.xlabel('Component  1')
        plt.ylabel('Component  2')
        plt.legend()
        plt.grid()
        plt.show()
```

Since the silhouette scores at the start of the development, we decided to use Principal Component Analysis to reduce the dimensions of the data and find a more compact representation of the data. At the same time, account for the variance of data and reduce noise.

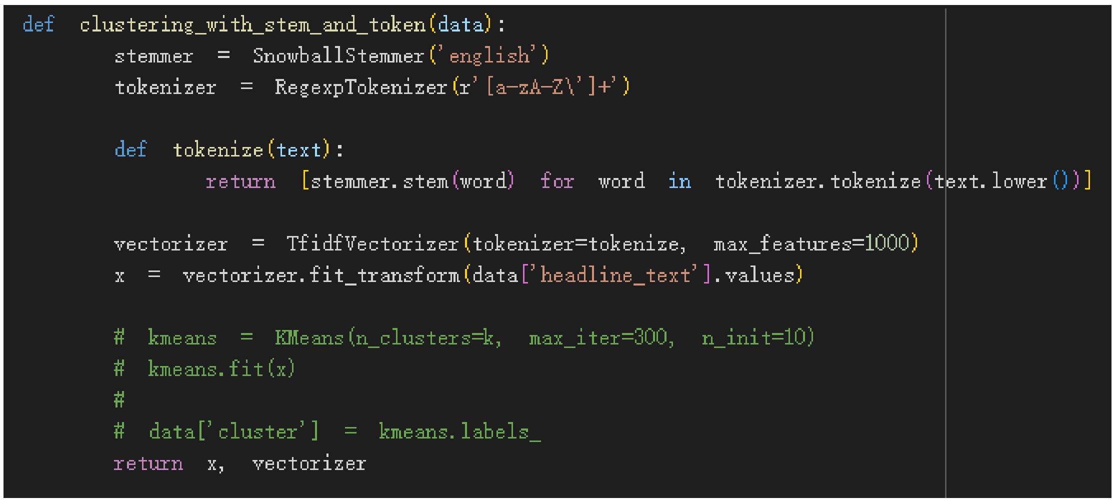
```
sentence_vectors = np.array([get_sentence_vector(sentence)  for
sentence in processed_data])
for k in range(12, 30, 2):
  pca = PCA(n_components=k)
  pca_vectors = pca.fit_transform(sentence_vectors)


  kmean = KMeans(n_clusters=3, random_state=42)
  kmean.fit(pca_vectors)

  score = calculate_silhouette(pca_vectors, kmean.labels_)
  print(f'PCA components: {k}\t Silhouette Score: {score}')
```

We used for loop to find the best k of the PCA algorithm. The sentence_vectors is the result after applying Word2Vec to convert and embed the headlines into vectors.



```
PCA components: 12      Silhouette Score: 0.3770864009857178
PCA components: 14      Silhouette Score: 0.3767782747745514
PCA components: 16      Silhouette Score: 0.3765593469142914
PCA components: 18      Silhouette Score: 0.3763916790485382
PCA components: 20      Silhouette Score: 0.3762620687484741
PCA components: 22      Silhouette Score: 0.3761633634567261
PCA components: 24      Silhouette Score: 0.3760814368724823
PCA components: 26      Silhouette Score: 0.3760015666484833
PCA components: 28      Silhouette Score: 0.375932514667511
```

Stemming



```
def clustering_with_stem_and_token(data):
    stemmer = SnowballStemmer('english')
    tokenizer = RegexpTokenizer(r'[a-zA-Z\']+')

    def tokenize(text):
        return [stemmer.stem(word) for word in tokenizer.tokenize(text.lower())]

    vectorizer = TfidfVectorizer(tokenizer=tokenize, max_features=1000)
    x = vectorizer.fit_transform(data['headline_text'].values)

    # kmeans = KMeans(n_clusters=k, max_iter=300, n_init=10)
    # kmeans.fit(x)
    #
    # data['cluster'] = kmeans.labels_
    return x, vectorizer
```

Tried to use SnowballStemmer to reduce variations of the same word to a common stem.

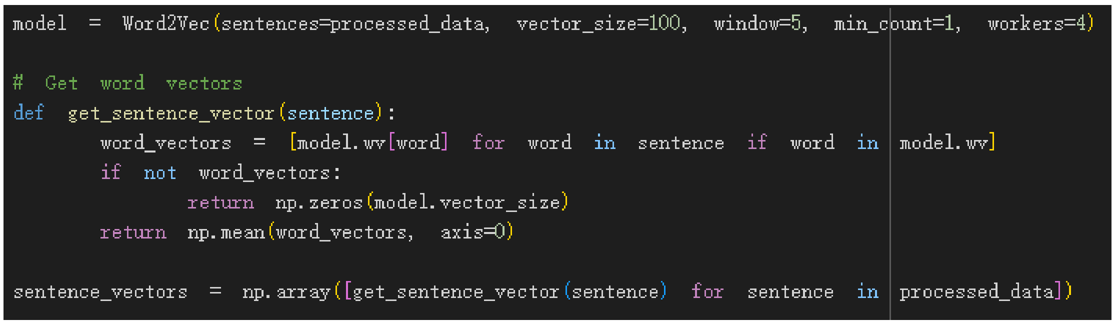
Latent Semantic Analysis (LSA)

Latent Semantic Analysis (LSA) is a technique in natural language processing of analyzing relationships between a set of documents, in this case is set of headlines, and the terms they contain by producing a set of concepts related to the documents and terms. LSA captures latent semantic structures and reduces the dimension. It preserves the most significant features and helps with eliminating noise and irrelevant information in the headline. It works effectively with KMeans clustering because it provides KMeans with a more informative representation of the data.

Tf-Idf Tokenization

Considering reducing noise, tried to use Tf-Idf Tokenization to reduce the weight of stop words, such as 'a, the, of, for, with'. It allows clustering algorithms to focus on the more meaningful aspects of the text. Also, Tf-Idf transforms text into a numerical format (vectors) that clustering algorithms like KMeans can directly work with.

Word2Vec Tokenization



```
model = Word2Vec(sentences=processed_data, vector_size=100, window=5, min_count=1, workers=4)

# Get word vectors
def get_sentence_vector(sentence):
    word_vectors = [model.wv[word] for word in sentence if word in model.wv]
    if not word_vectors:
        return np.zeros(model.vector_size)
    return np.mean(word_vectors, axis=0)

sentence_vectors = np.array([get_sentence_vector(sentence) for sentence in processed_data])
```

Tried to use Word2Vec to convert words into numerical vectors, which allows algorithms to understand and manipulate textual data. The text was preprocessed before applying

Word2Vec.

Results

KMeans

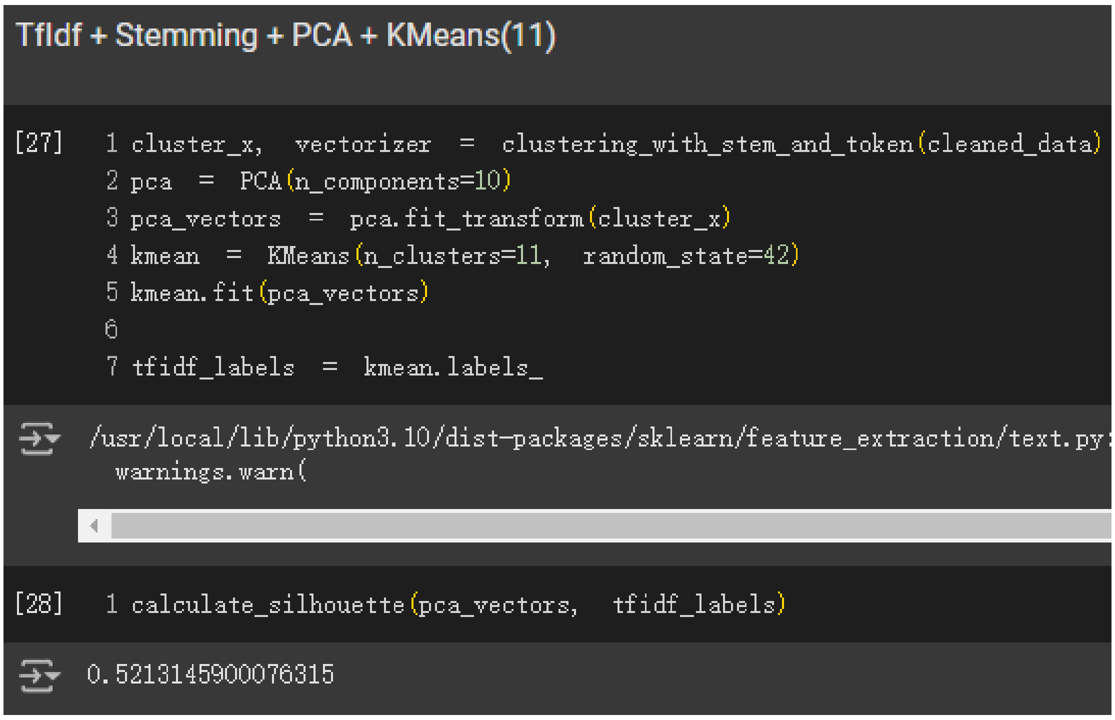
Approaches	**Silhouette score**	**Explanation**
Tfidf + stemming + PCA + KMeans(11)	0.52131	TF-IDF highlights important terms, but stemming and PCA can strip context and detail.
Tfidf + stemming + LSA + KMeans(10)	0.60361	TF-IDF with LSA can capture meaningful topics, but stemming may still remove important context.
Tfidf + LSA + KMeans(10)	0.69462	This combination harnesses TF-IDF's term weighting with LSA's dimensionality reduction, preserving important patterns while simplifying data.
Tfidf + PCA + KMeans(11)	**0.73011**	TF-IDF highlights key terms, and PCA focuses on variance reduction, capturing distinct document features.
Tfidf + KMeans(3)	0.0013360	Directly uses term importance for clustering, but without dimensionality reduction, it may capture too much noise or irrelevant features.
Word2Vec + PCA + KMeans(3)	0.37920	Word2Vec captures word semantics, and PCA reduces dimensionality for clustering.
Word2Vec + LSA + KMeans(3)	0.38110	Combines semantic richness of Word2Vec with LSA's ability to reveal underlying structures, preserving more context.
Word2Vec + KMeans(3)	0.37603	Utilizes semantic vectors directly for clustering, capturing nuanced meanings but potentially high-dimensional noise.

**Note:* All results are rounded to 5 significant figures.*

Hierarchical Clustering

Approaches	**Silhouette score**
Word2Vec + LSA	0.22713
Word2Vec + PCA	0.20874
Tfidf + LSA	0.68697
Tfidf + PCA	**0.72518**

Tfidf + stemming + PCA + KMeans(11)



Tfldf + Stemming + PCA + KMeans(11)

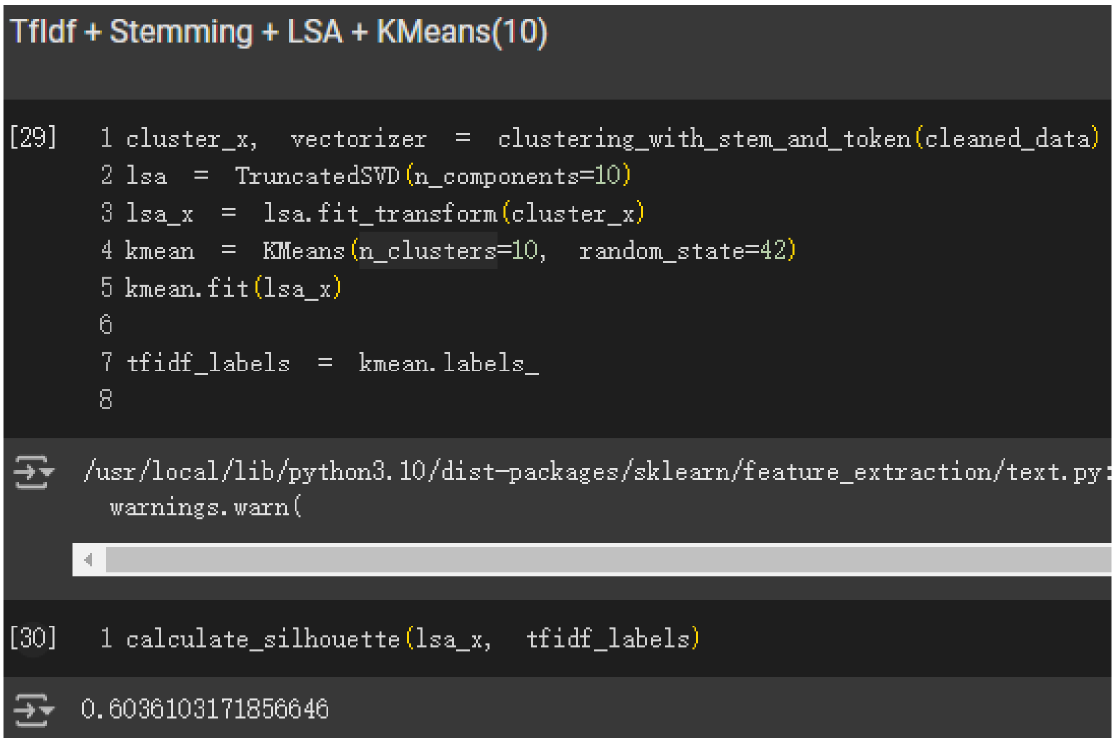
```
[27]
cluster_x, vectorizer = clustering_with_stem_and_token(cleaned_data)
pca = PCA(n_components=10)
pca_vectors = pca.fit_transform(cluster_x)
kmean = KMeans(n_clusters=11, random_state=42)
kmean.fit(pca_vectors)

tfidf_labels = kmean.labels_
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/feature_extraction/text.py:
  warnings.warn(
```

```
[28]
calculate_silhouette(pca_vectors, tfidf_labels)
```

```
0.5213145900076315
```

Tfidf + stemming + LSA + KMeans(10)



Tfldf + Stemming + LSA + KMeans(10)

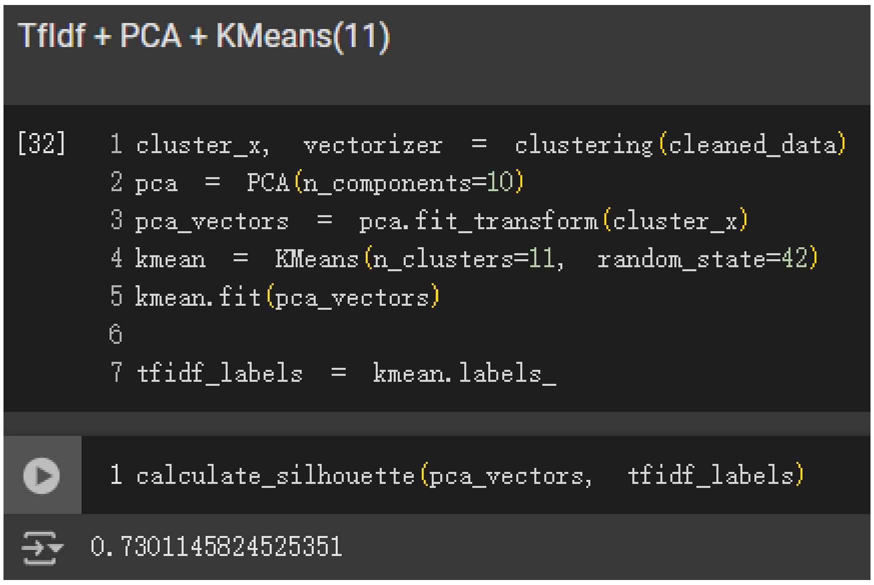
```
[29]
cluster_x, vectorizer = clustering_with_stem_and_token(cleaned_data)
lsa = TruncatedSVD(n_components=10)
lsa_x = lsa.fit_transform(cluster_x)
kmean = KMeans(n_clusters=10, random_state=42)
kmean.fit(lsa_x)

tfidf_labels = kmean.labels_

```

```
/usr/local/lib/python3.10/dist-packages/sklearn/feature_extraction/text.py:
  warnings.warn(
```

```
[30]
calculate_silhouette(lsa_x, tfidf_labels)
```

```
0.6036103171856646
```

TfIdf + PCA + KMeans(11)



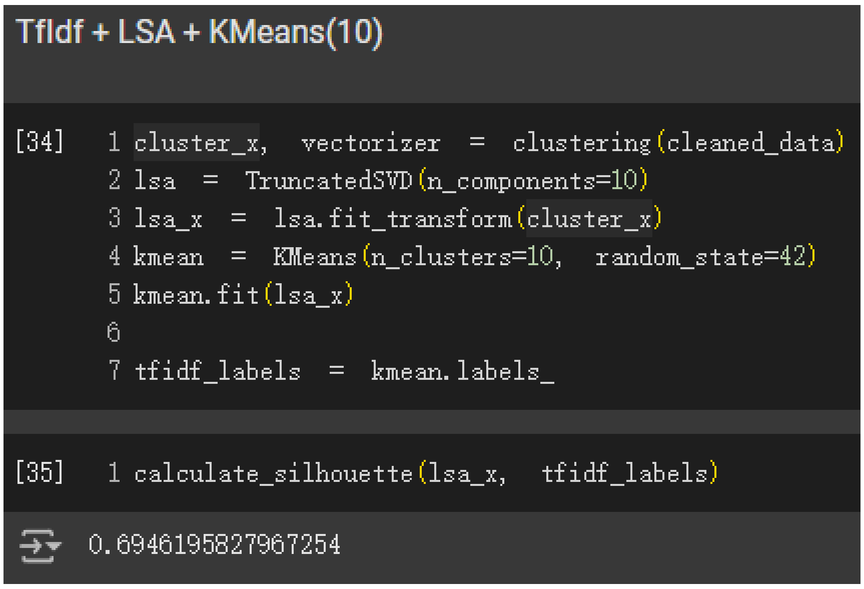
TfIdf + PCA + KMeans(11)

```
[32]
cluster_x, vectorizer = clustering(cleaned_data)
pca = PCA(n_components=10)
pca_vectors = pca.fit_transform(cluster_x)
kmean = KMeans(n_clusters=11, random_state=42)
kmean.fit(pca_vectors)

tfidf_labels = kmean.labels_
```

```
calculate_silhouette(pca_vectors, tfidf_labels)
```

```
0.7301145824525351
```

TfIdf + LSA + KMeans(10)



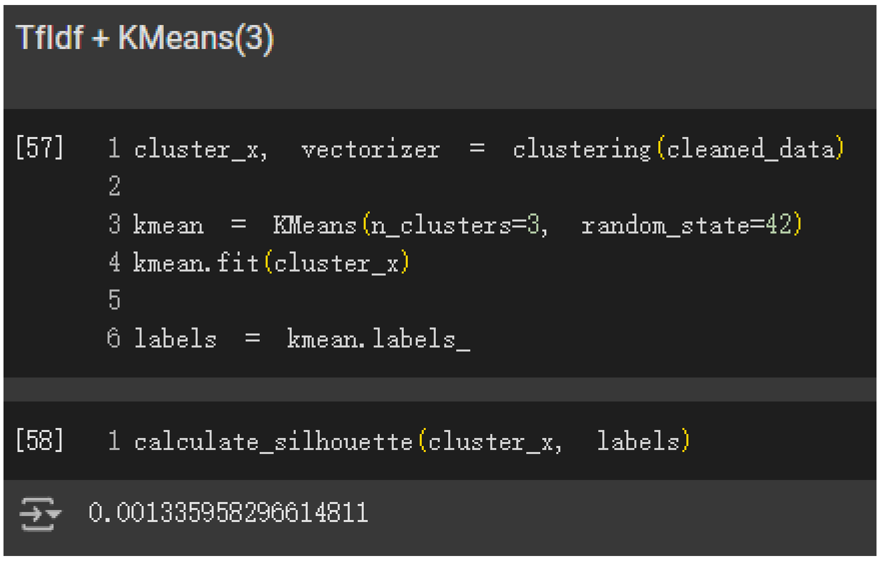
TfIdf + LSA + KMeans(10)

```
[34]
cluster_x, vectorizer = clustering(cleaned_data)
lsa = TruncatedSVD(n_components=10)
lsa_x = lsa.fit_transform(cluster_x)
kmean = KMeans(n_clusters=10, random_state=42)
kmean.fit(lsa_x)

tfidf_labels = kmean.labels_
```

```
[35]
calculate_silhouette(lsa_x, tfidf_labels)
```

```
0.6946195827967254
```

Tfidf + **KMeans(3)**



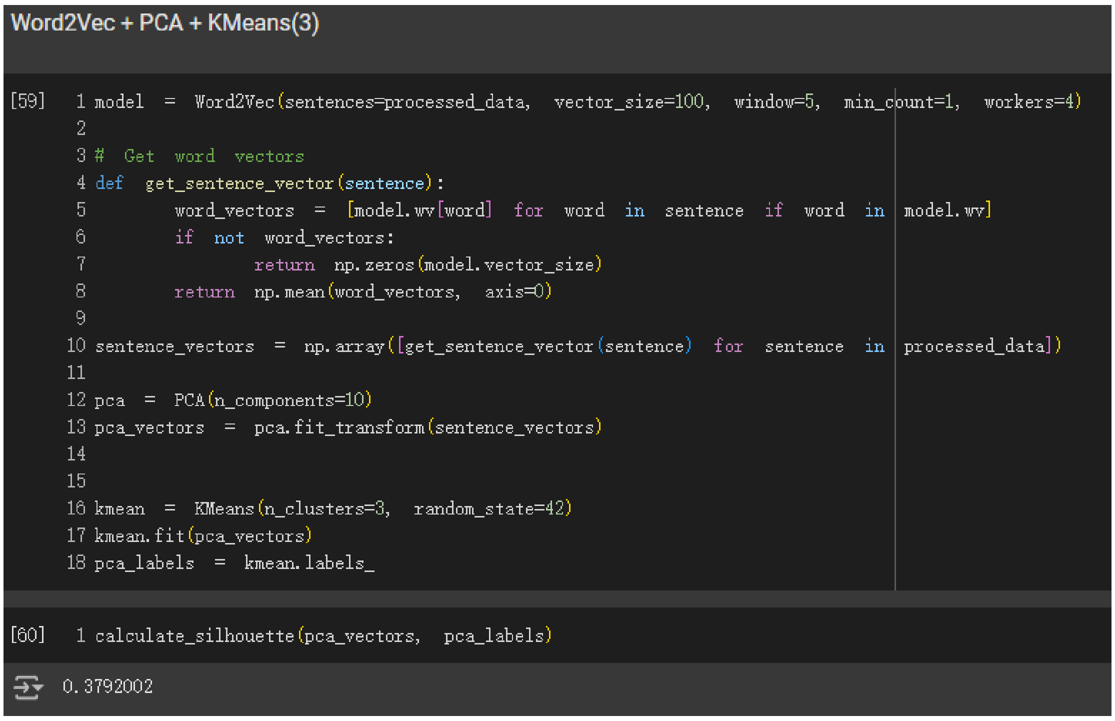
Tfldf + KMeans(3)

```
[57]  cluster_x, vectorizer = clustering(cleaned_data)

      kmean = KMeans(n_clusters=3, random_state=42)
      kmean.fit(cluster_x)

      labels = kmean.labels_
```

```
[58]  calculate_silhouette(cluster_x, labels)
```

```
0.001335958296614811
```

Word2Vec + PCA + **KMeans(3)**



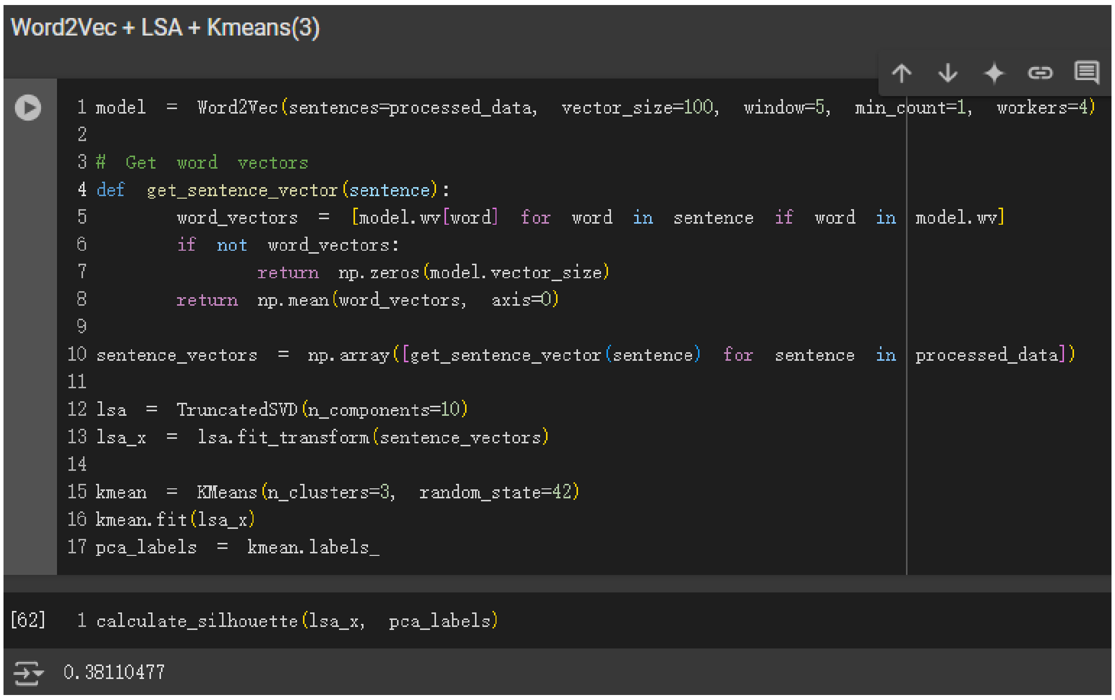
Word2Vec + PCA + KMeans(3)

```
[59] model = Word2Vec(sentences=processed_data, vector_size=100, window=5, min_count=1, workers=4)

     # Get word vectors
     def get_sentence_vector(sentence):
         word_vectors = [model.wv[word] for word in sentence if word in model.wv]
         if not word_vectors:
             return np.zeros(model.vector_size)
         return np.mean(word_vectors, axis=0)

     sentence_vectors = np.array([get_sentence_vector(sentence) for sentence in processed_data])

     pca = PCA(n_components=10)
     pca_vectors = pca.fit_transform(sentence_vectors)


     kmean = KMeans(n_clusters=3, random_state=42)
     kmean.fit(pca_vectors)
     pca_labels = kmean.labels_
```

```
[60] calculate_silhouette(pca_vectors, pca_labels)
```

```
0.3792002
```

Word2Vec + LSA + **KMeans(3)**



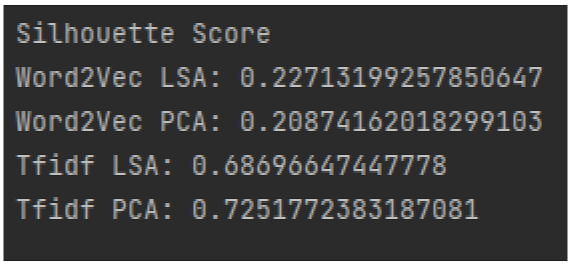
Word2Vec + LSA + Kmeans(3)

```
model = Word2Vec(sentences=processed_data, vector_size=100, window=5, min_count=1, workers=4)

# Get word vectors
def get_sentence_vector(sentence):
    word_vectors = [model.wv[word] for word in sentence if word in model.wv]
    if not word_vectors:
        return np.zeros(model.vector_size)
    return np.mean(word_vectors, axis=0)

sentence_vectors = np.array([get_sentence_vector(sentence) for sentence in processed_data])

lsa = TruncatedSVD(n_components=10)
lsa_x = lsa.fit_transform(sentence_vectors)

kmean = KMeans(n_clusters=3, random_state=42)
kmean.fit(lsa_x)
pca_labels = kmean.labels_
```

```
[62] calculate_silhouette(lsa_x, pca_labels)
```

```
0.38110477
```

Hierarchical Clustering



```
Silhouette Score
Word2Vec LSA: 0.22713199257850647
Word2Vec PCA: 0.20874162018299103
Tfidf LSA: 0.686966474477778
Tfidf PCA: 0.7251772383187081
```

Conclusion

- PCA performs better in both clustering methods

PCA's ability to capture essential features and reduce noise makes it a powerful preprocessing step that enhances the performance of both KMeans and Hierarchical Clustering by providing clearer, more distinct data representations.

- TF-IDF generally performs better in both clustering methods

TF-IDF's focus on term importance consistently aids in distinguishing document clusters, making it effective for both clustering methods.