Text Clustering

WITH SIMILIARITY MEASUREMENT

Salsabila Aurora Octavia	3322600005
Mutiara Sanny	3322600017
Bayu Kurniawan	3322600019
Calysta Moza Salsabilla	3322600028

Text Mining

WHAT WE DO

Develop program menggunakan algoritma k-means clustring untuk data 20newsgroups 3 kelas :

Sport, Religion dan Technology.

A. Bag of Words

- Euclidian
- Cosine Similiarity

B. TF/IDF

- Euclidian
- Cosine Similiarity

C. Count Vectorizer

- Euclidian
- Cosine Similiarity

D. Jaccard Index

Text Mining

Cleaning Data

UNSTRUCTURED > STRUCTURED

Salsabila Aurora Octavia	3322600005
Mutiara Sanny	3322600017
Bayu Kurniawan	3322600019
Calysta Moza Salsabilla	3322600028

Text Mining

IMPORT LIBRARY YANG DIBUTUHKAN

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import jaccard_score
import numpy as np
import pandas as pd
import re
import string
import nltk
from nltk.corpus import stopwords
from sklearn.metrics import davies_bouldin_score, silhouette_score
import matplotlib.pyplot as plt
```

PRE-PROCESSING

```
categories = [
 'comp.graphics',
 'rec.sport.baseball',
 'soc.religion.christian'
]
dataset = fetch_20newsgroups(subset='train',
                             categories=categories,
                             shuffle=True, remove=
                             ('headers', 'footers', 'quotes'))
df = pd.DataFrame(dataset.data, columns=["corpus"])
df
```

	corpus
0	\n\n\n\nAs far as I can see, one of the big di...

Ambil Cetgories yang diinginkan kemudian ersihkan data dari stopwords, special character, angka, dan transformasi kata menjadi lowercase

```
def preprocess_text(text: str, remove_stopwords: bool) -> str:
    """This utility function sanitizes a string by:
    - removing links
    - removing special characters
    - removing numbers
    - removing stopwords
    - transforming in lowercase
    - removing excessive whitespaces
    Args:
        text (str): the input text you want to clean
        remove_stopwords (bool): whether or not to remove stopwords
    Returns:
        str: the cleaned text
    """

    # remove links
    text = re.sub(r"http\S+", "", text)
    # remove special chars and numbers
    text = re.sub("[^A-Za-z]+", " ", text)
    # remove stopwords
    if remove_stopwords:
        # 1. tokenize
        tokens = nltk.word_tokenize(text)
        # 2. check if stopword
        tokens = [w for w in tokens if not w.lower() in stopwords.words("english")]
        # 3. join back together
        text = " ".join(tokens)
    # return text in lower case and stripped of whitespaces
    text = text.lower().strip()
    return text
```

```
df['cleaned'] = df['corpus'].apply(lambda x: preprocess_text(x, remove_stopwords=True))
df
```

Text Mining

Bag of Words

EUCLIDIAN AND COSINE SIMILIARITY

Salsabila Aurora Octavia	3322600005
Mutiara Sanny	3322600017
Bayu Kurniawan	3322600019
Calysta Moza Salsabilla	3322600028

Text Mining

A. BAG OF WORDS

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import pairwise_distances

# Menggunakan CountVectorizer untuk Bag of Words
count_vectorizer = CountVectorizer(binary = True)
X_bow = count_vectorizer.fit_transform(df['cleaned']).toarray()
bow_df = pd.DataFrame(X_bow)
bow_df
```

	0	1	2	3	4	5	6	7	8	9	...	31700	31701	31702	31703	31704	31705	31706	31707	31708	31709
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2959	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2960	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2961	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2962	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	0	0
2963	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

2964 rows × 31710 columns

Kode yang diberikan menggunakan `CountVectorizer` dengan parameter `binary=True` untuk mengonversi teks yang telah di-preprocess ke dalam representasi Bag of Words (BoW) dalam bentuk biner. Hasilnya, setiap dokumen dalam dataset diubah menjadi vektor biner di mana setiap elemen vektor menandai kehadiran atau ketiadaan kata tertentu dalam dokumen tersebut. Dalam representasi ini, setiap baris dalam DataFrame `bow_df` mewakili sebuah dokumen, sementara setiap kolom mewakili keberadaan sebuah kata dalam BoW.

A. BAG OF WORDS W/ EUCLIDIAN

```python
from sklearn.metrics import davies_bouldin_score, silhouette_score
from scipy.spatial.distance import euclidean

# Perform K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_bow)

# Predict cluster labels for each data point
cluster_labels = kmeans.labels_
cluster_labels
# Calculate Davies-Bouldin Index (DBI) with Euclidean Distance
dbi = davies_bouldin_score(X_bow, cluster_labels)

# Calculate Silhouette Score
silhouette_avg = silhouette_score(X_bow, cluster_labels, metric='euclidean')

# Print cluster results and metrics
print(f"Davies-Bouldin Index: {dbi}")
print(f"Silhouette Score: {silhouette_avg}")
```

```
Davies-Bouldin Index: 6.283776867884444
Silhouette Score: 0.13695305040592806
```

Text Mining

Hasil evaluasi yang memberikan nilai sekitar 6.28, serta Silhouette Score sebesar 0.137, menunjukkan performa klasterisasi pada data Bag of Words (BoW).

Dalam kasus ini, DBI Score sebesar 6.28 menunjukkan adanya penyebaran yang besar antar klaster, mengindikasikan bahwa klasterisasi tidak optimal.

sedangkan dengan Silhoutte scor sekitar 0.137, hasil menunjukkan bahwa sebagian besar titik data tidak terlalu dekat dengan batas klasternya, namun penempatan titik dalam klaster tidak optimal secara keseluruhan. Nilai Silhouette Score yang mendekati nol menandakan adanya ketidakcocokan dalam pembentukan klaster.

Secara keseluruhan, kedua metrik tersebut menunjukkan adanya ruang untuk perbaikan dalam klasterisasi yang dilakukan oleh algoritma KMeans pada data Bag of Words.

A. BAG OF WORDS W/ COSINE

```python
from sklearn.metrics.pairwise import pairwise_distances
# Calculate Davies-Bouldin Index (DBI) with Cosine Similarity
centroids = kmeans.cluster_centers_
cosine_distances = pairwise_distances(X_bow, centroids, metric='cosine')

# Calculate DBI
dbi = 0
for i in range(len(centroids)):
    max_d = -1
    for j in range(len(centroids)):
        if i != j:
            d = (cosine_distances[i] + cosine_distances[j]).mean() / cosine_distances[i, j]
            if d > max_d:
                max_d = d
    dbi += max_d

dbi /= len(centroids)

# Calculate Silhouette Score with Cosine Similarity
silhouette_avg = silhouette_score(X_bow, cluster_labels, metric='cosine')

# Print cluster results and metrics
print(f"Davies-Bouldin Index: {dbi}")
print(f"Silhouette Score: {silhouette_avg}")
```

```
Davies-Bouldin Index: 2.1591759609009853
Silhouette Score: 0.011146670004861308
```

(DBI) yang menghasilkan angka sekitar 2.159 menunjukkan tingkat pemisahan dan kekompakan yang cukup dalam klaster yang dibentuk dari data tersebut. angka yang diperoleh menunjukkan adanya sejumlah tumpang tindih atau ketidakjelasan dalam batas klaster.

Selain itu, Skor Silhouette, yang mendekati 0.011, menunjukkan bahwa model klasterisasi mungkin tidak optimal untuk dataset tersebut. Skor yang rendah ini mengindikasikan bahwa klaster tidak terpisah dengan baik dan titik data mungkin tidak dikelompokkan secara akurat dalam klaster mereka.

Secara keseluruhan, kedua metrik evaluasi ini menyarankan kinerja klasterisasi yang kurang optimal.

TF/IDF

EUCLIDIAN AND COSINE SIMILIARITY

Salsabila Aurora Octavia	3322600005
Mutiara Sanny	3322600017
Bayu Kurniawan	3322600019
Calysta Moza Salsabilla	3322600028

Text Mining

B. TF-IDF

```
# TF-IDF Feature Generation
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.tokenize import RegexpTokenizer

# Initialize regex tokenizer
tokenizer = RegexpTokenizer(r'\w+')

# # Vectorize document using TF-IDF
tf_idf_vect = TfidfVectorizer(lowercase=True,
                        stop_words='english',
                        ngram_range = (1,1),
                        tokenizer = tokenizer.tokenize)

# Fit and Transfrom Text Data
X_train_counts = tf_idf_vect.fit_transform(twenty_train.data)

# Check Shape of Count Vector
X_train_counts.shape
```

```
C:\Users\yunit\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py:525: UserWarning: The p
arameter 'token_pattern' will not be used since 'tokenizer' is not None'
  warnings.warn(
```

```
(1780, 27715)
```

Metode lain untuk membuat vektor dari kalimat adalah TF-IDF. Berbeda dengan bow, metode ini dapat mengevaluasi seberapa penting suatu kata dalam tiap dokumen dengan mempertimbangkan dokumen-dokumen lainnya. Vektor TF-IDF dibuat menggunakan fungsi TfidfVectorizer. Sebelumnya dilakukan inisiasi tokenizer untuk menghilangkan karakter non-alphanumerik dan memisahkan kata-kata berdasarkan spasi atau karakter non-alphanumerik. Dilakukan juga perubahan text menjadi huruf kecil, dan penghilangan stop word. Hasil akhirnya, didapatkan matriks tf-idf dengan ukuran 1780x27715

B. TF-IDF W/ EUCLIDIAN

```python
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import euclidean_distances

# Compute pairwise euclidean distences
euclidean_distance = euclidean_distances(X_train_counts)

# Create K-Means object
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=0)

# Fit K-Means
kmeans.fit(euclidean_distance)

# Get labels for each data point
pred_labels = kmeans.labels_
```

```python
from sklearn import metrics
# Compute DBI score
dbi_ed = metrics.davies_bouldin_score(euclidean_distance, pred_labels)

# Compute Silhoutte Score
ss_ed = metrics.silhouette_score(euclidean_distance, pred_labels , metric='euclidean')

# Print the DBI and Silhoutte Scores
print("DBI Score: ", dbi_ed, "\nSilhoutte Score: ", ss_ed)
```

```
DBI Score:  5.538175144470997
Silhoutte Score:  0.033782641484301795
```

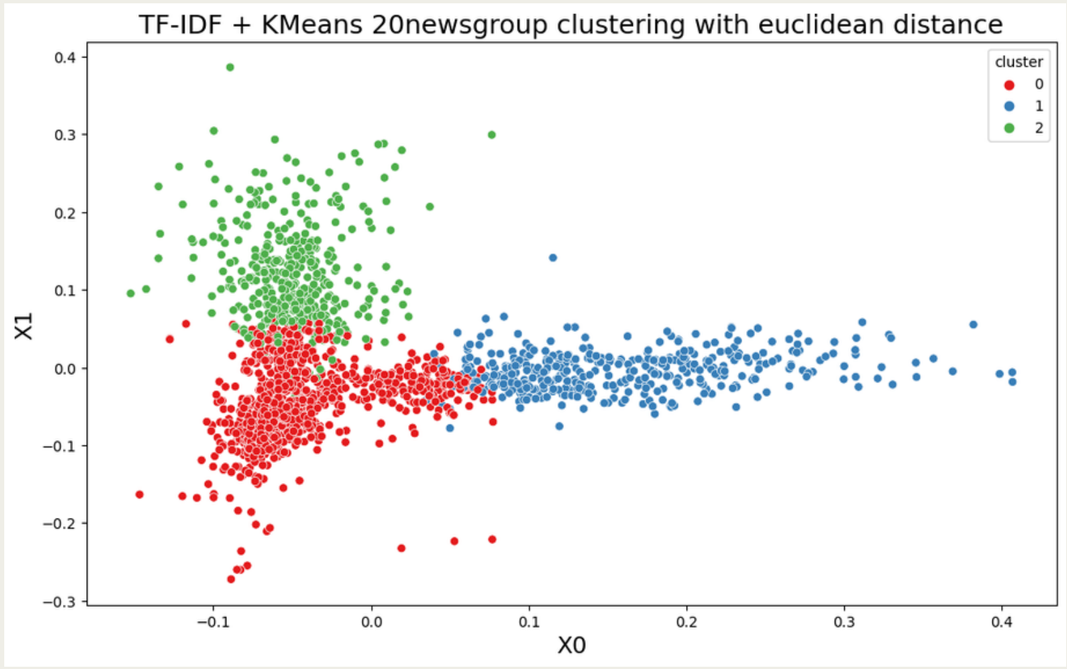
Text Mining

Dilakukan clustering K-Means dengan rumus jarak euclidean distance. Jarak Euclidean adalah metrik yang mengukur jarak antara dua titik dalam ruang n-dimensi. Jarak Euclidean dihitung dengan mengambil akar kuadrat dari jumlah kuadrat perbedaan antara setiap pasangan elemen dalam vektor. Semakin kecil nilai jarak Euclidean, semakin dekat kedua titik dalam ruang n-dimensi tersebut.

Hasil score dari model tersebut menunjukkan bahwa klustering data mungkin tidak optimal. Nilai DBI Score yang tinggi seperti 5.538 menunjukkan bahwa terdapat tumpang tindih antara kluster atau bahwa kluster mungkin tidak kompak, sedangkan nilai Silhouette Score yang sangat rendah, seperti 0.034, menunjukkan bahwa objek dalam kluster memiliki jarak yang serupa dengan kluster tetangga terdekat.

B. TF-IDF W/ EUCLIDIAN

```
from sklearn.decomposition import PCA
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# initialize PCA with 2 components
pca = PCA(n_components=2, random_state=42)
# pass our X to the pca and store the reduced vectors into pca_vecs
pca_vecs = pca.fit_transform(X_train_counts.toarray())
# save our two dimensions into x0 and x1
x0 = pca_vecs[:, 0]
x1 = pca_vecs[:, 1]

data = {'cluster' : pred_labels,'x0': x0, 'x1' :x1}
df = pd.DataFrame(data)

# Set image size
plt.figure(figsize=(12, 7))

# Set a title
plt.title("TF-IDF + KMeans 20newsgroup clustering with euclidean distance", fontdict={"fontsize": 18})

# Set axes names
plt.xlabel("X0", fontdict={"fontsize": 16})
plt.ylabel("X1", fontdict={"fontsize": 16})

# Create scatter plot with Seaborn, where 'cluster' is the hue variable used for coloring
sns.scatterplot(data=df, x='x0', y='x1', hue='cluster', palette="Set1")

# Show the plot
plt.show()
```



Untuk melihat hasil claster, dapat dilakukan visualisasi dengan scatterplot seperti berikut. Sebelumnya, dilakukan PCA untuk mereduksi dimensi data menjadi 2 dimensi.

Seperti hasil scorenya, dapat dilihat bahwa masih terdapat tumpang tindih antara klaster dan batas antar cluster juga kurang jelas.

B. TF-IDF W/ COSINE

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


# Compute pairwise cosine similarity
cosine_similarities = cosine_similarity(X_train_counts)

# Create K-Means object
num_clusters = 3
kmeans = KMeans(n_clusters=num_clusters, random_state=0)

# Fit K-Means
kmeans.fit(cosine_similarities)


# Get labels for each data point
pred_labels = kmeans.labels_
```

```python
from sklearn import metrics
# Compute DBI score
dbi_cosine = metrics.davies_bouldin_score(cosine_similarities, pred_labels)

# Compute Silhoutte Score
ss_cosine = metrics.silhouette_score(cosine_similarities, pred_labels , metric='cosine')

# Print the DBI and Silhoutte Scores
print("DBI Score: ", dbi_cosine, "\nSilhoutte Score: ", ss_cosine)
```

```
DBI Score:  3.748140297166317
Silhoutte Score:  0.03250370506893042
```

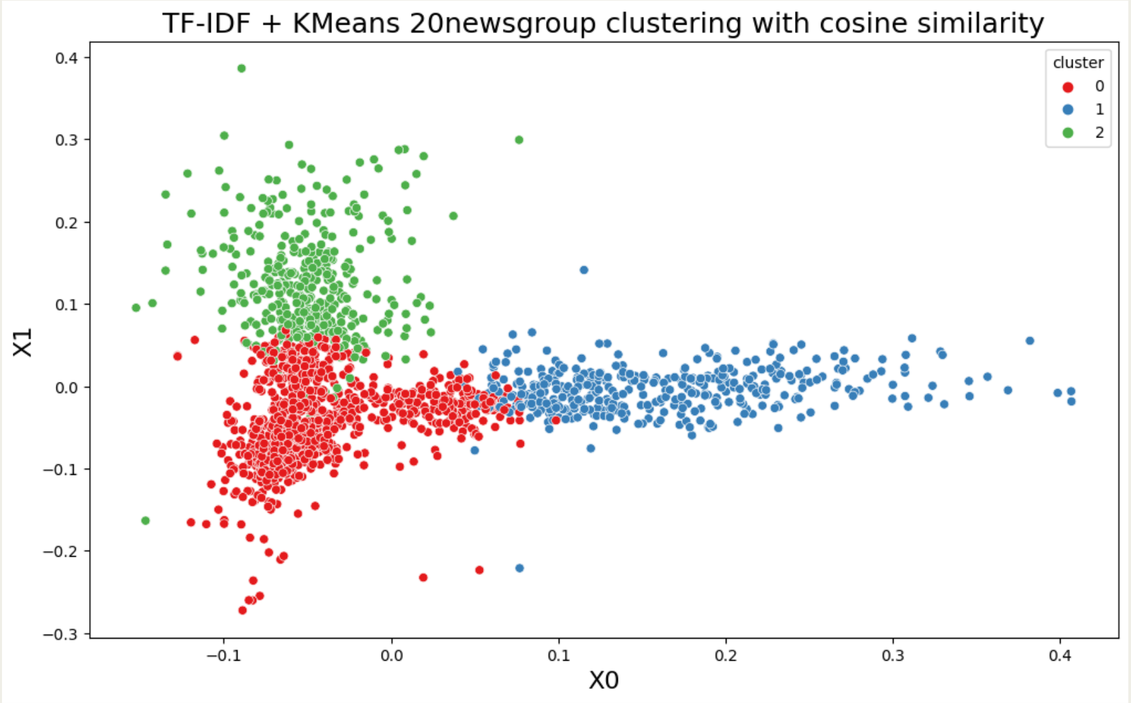
Text Mining

Selanjutnya, dilakukan klustering K-means dengan rumus jarak cosine similarity, Cosine similarity adalah metrik yang mengukur sejauh mana dua vektor sejajar dalam ruang vektor. Perhitungan cosine similarity antara dua vektor A dan B dihitung sebagai dot product(perkalian) vektor A dan B, dibagi oleh hasil perkalian panjang vektor A dan B. Hasilnya berkisar dari -1 (sangat berlawanan arah) hingga 1 (sangat sejajar). Nilai 1 menunjukkan bahwa kedua vektor sepenuhnya sejajar, 0 menunjukkan bahwa vektor tersebut saling tegak lurus, dan -1 menunjukkan bahwa keduanya berlawanan arah. Semakin besar nilai cosine similarity, semakin mirip kedua vektor tersebut dalam arah.

Hasil score dari model tersebut kurang lebih hampir mirip dengan hasil klastering menggunakan euclidean distance sebelumnya, namun pada model ini nilai dbi scorenya lebih kecil.

B. TF-IDF W/ EUCLIDIAN

```python
from sklearn.decomposition import PCA
import seaborn as sns
import pandas as pd

# initialize PCA with 2 components
pca = PCA(n_components=2, random_state=42)
# pass our X to the pca and store the reduced vectors into pca_vecs
pca_vecs = pca.fit_transform(X_train_counts.toarray())
# save our two dimensions into x0 and x1
x0 = pca_vecs[:, 0]
x1 = pca_vecs[:, 1]

data = {'cluster' : pred_labels,'x0': x0, 'x1' :x1}
df = pd.DataFrame(data)

# Set image size
plt.figure(figsize=(12, 7))

# Set a title
plt.title("TF-IDF + KMeans 20newsgroup clustering with cosine similarity", fontdict={"fontsize": 18})

# Set axes names
plt.xlabel("X0", fontdict={"fontsize": 16})
plt.ylabel("X1", fontdict={"fontsize": 16})

# Create scatter plot with Seaborn, where 'cluster' is the hue variable used for coloring
sns.scatterplot(data=df, x='x0', y='x1', hue='cluster', palette="Set1")

# Show the plot
plt.show()
```

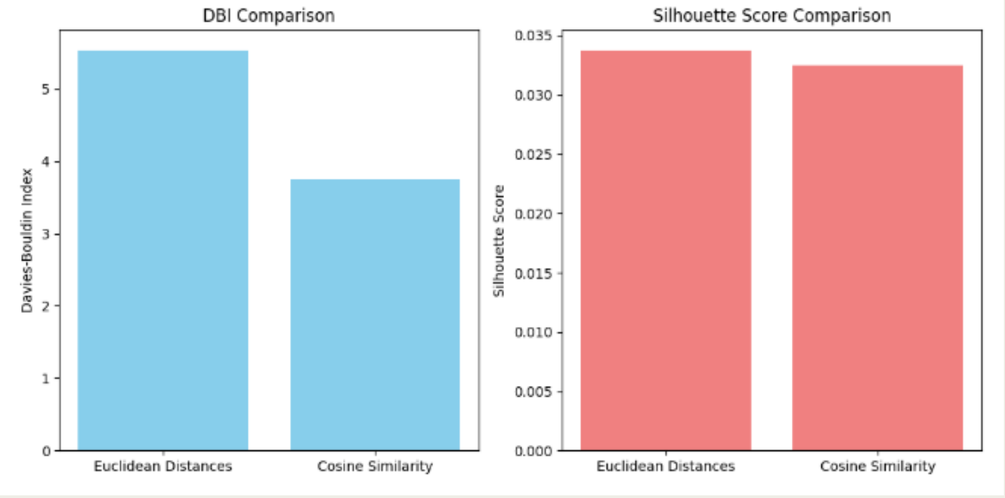


Ketika divisualisasikan dengan scatterplot, hasil claster ini juga hampir sama seperti sebelumnya yaitu masih terdapat tumpang tindih antara klaster dan batas antar cluster juga kurang jelas.

Text Mining

B. PERBANDINGAN SCORE

```
jarak = ['Euclidean Distances', 'Cosine Similarity']
dbi_scores = [dbi_ed, dbi_cosine]
ss = [ss_ed, ss_cosine]

plt.figure(figsize=(10, 5))

# DBI Plot
plt.subplot(1, 2, 1)
plt.bar(jarak, dbi_scores, color='skyblue')
plt.ylabel('Davies-Bouldin Index')
plt.title('DBI Comparison')

# Silhouette Score Plot
plt.subplot(1, 2, 2)
plt.bar(jarak, ss, color='lightcoral')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score Comparison')

plt.tight_layout()
plt.show()
```



Text Mining

Dari grafik perbandingan score disamping, dapat disimpulkan bahwa meskipun model dengan rumus jarak Cosine Similarity memiliki nilai DBI yang tinggi, perbandingan dengan Model Euclidean Distance menunjukkan bahwa model dengan cosine similarity memiliki kualitas klustering yang sedikit lebih baik. Namun model dengan rumus jarak Euclidean distance memiliki Silhouette Score yang lebih tinggi, menunjukkan bahwa objek dalam kluster lebih berdekatan satu sama lain. Oleh karena itu, pemilihan model tergantung pada tujuan analisinya, apakah ingin lebih memperhatikan pemisahan kluster (DBI) atau kompaknya objek dalam kluster (Silhouette Score) sesuai dengan metrik yang digunakan.

Walau begitu, kedua model masih memiliki nilai DBI tinggi dan Silhouette Score yang rendah, yang menandakan bahwa kualitas klusteringnya bisa ditingkatkan.

Count Vectorizer

EUCLIDIAN AND COSINE SIMILIARITY

Salsabila Aurora Octavia	3322600005
Mutiara Sanny	3322600017
Bayu Kurniawan	3322600019
Calysta Moza Salsabilla	3322600028

Text Mining

COUNT VECTORIZER

```
# Menggunakan Count Vectorizer untuk mengubah teks dokumen menjadi vektor fitur
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(df['corpus'])

# Ubah array X menjadi matriks sparse CSR
X_sparse = csr_matrix(X)

# Normalisasi matriks sparse berdasarkan Cosine Similarity
from sklearn.preprocessing import normalize
X_normalized_sparse = normalize(X_sparse, norm='l2')

# Melakukan K-Means Clustering dengan Euclidean Distance
n_clusters = 3  # Ganti dengan jumlah cluster yang sesuai
kmeans_euclidean = KMeans(n_clusters=n_clusters, random_state=0).fit(X_normalized_sparse)
labels_euclidean = kmeans_euclidean.labels_

print('Hasil K-Means Clustering dengan Euclidean Distance :', labels_euclidean)
```

Hasil K-Means Clustering dengan Euclidean Distance : [1 1 2 ... 2 1 2]

K-Means Clustering dengan metode Euclidean Distance pada data yang telah dinormalisasi

- Count Vectorizer: Menggunakan CountVectorizer untuk mengubah teks dokumen menjadi vektor fitur berdasarkan frekuensi kata-kata. Ini mengubah teks dokumen menjadi representasi numerik yang dapat digunakan untuk clustering.

- Matriks Sparse (X_sparse): Hasil dari CountVectorizer adalah matriks sparse yang menggambarkan frekuensi kata dalam dokumen. Matriks ini digunakan untuk meneruskan informasi ke algoritma clustering.
- Normalisasi (X_normalized_sparse): Melakukan normalisasi pada matriks sparse berdasarkan Cosine Similarity. Normalisasi ini penting ketika ingin menggunakan metrik jarak Cosine Similarity dalam clustering.
- K-Means Clustering dengan Euclidean Distance: Melakukan K-Means Clustering dengan Euclidean Distance menggunakan KMeans. Dalam menentukan jumlah cluster (n_clusters) yang ingin dihasilkan, dalam hal ini 3 cluster. Hasil clustering disimpan dalam labels_euclidean.

COUNT VECTORIZER

```
# Menghitung Cosine Similarity
cosine_similarities = cosine_similarity(X_normalized_sparse)

# Mengisi diagonal dengan nol pada matriks jarak Cosine Similarity
np.fill_diagonal(cosine_similarities, 0)

# Melakukan K-Means Clustering dengan Cosine Similarity
kmeans_cosine = KMeans(n_clusters=n_clusters, init='k-means++').fit(cosine_similarities)
labels_cosine = kmeans_cosine.labels_

print('Hasil K-Means Clustering dengan Cosine Similarity :', labels_cosine)
```

```
C:\Users\bayuk\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\Local
te-packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of `n_init` will
Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

```
Hasil K-Means Clustering dengan Cosine Similarity : [0 0 1 ... 2 2 1]
```

K-Means Clustering dengan metode Cosine Similarity pada data yang telah dinormalisasi

1. Menghitung Cosine Similarity: Menggunakan `cosine_similarity` untuk menghitung Cosine Similarity antara vektor-vektor yang telah dinormalisasi dalam matriks `X_normalized_sparse`. Ini menghasilkan matriks jarak Cosine Similarity antara semua pasangan dokumen dalam dataset.

1. Mengisi Diagonal dengan Nol: Mengisi diagonal matriks jarak Cosine Similarity dengan nol menggunakan `np.fill_diagonal(cosine_similarities, 0)`. Hal ini dilakukan karena diagonal matriks tersebut akan selalu memiliki nilai 1 (similarity maksimum antara dokumen dengan diri mereka sendiri), dan dalam analisis clustering, biasanya diagonal diabaikan.
2. K-Means Clustering dengan Cosine Similarity: Melakukan K-Means Clustering dengan metode Cosine Similarity menggunakan `KMeans`. Menggunakan `cosine_similarities` sebagai data input. Jumlah cluster yang ditentukan adalah `n_clusters` (dalam hal ini, 3 cluster). Hasil clustering disimpan dalam `labels_cosine`.

COUNT VECTORIZER

```
# Mengelompokkan dokumen ke dalam 3 kelas (atau sesuai dengan n_clusters yang Anda tentukan)
for i in range(n_clusters):
    print(f"Kelas {i + 1} (Euclidean Distance):")
    for j in np.where(labels_euclidean == i)[0][:5]:
        print(f"- Dokumen {j + 1}: {df['corpus'][j]}")

for i in range(n_clusters):
    print(f"Kelas {i + 1} (Cosine Similarity):")
    for j in np.where(labels_cosine == i)[0][:5]:
        print(f"- Dokumen {j + 1}: {df['corpus'][j]}")
```

Mengelompokkan dokumen ke dalam kelas (cluster) berdasarkan hasil clustering yang telah dilakukan baik dengan metode Euclidean Distance maupun Cosine Similarity.

1. For Loop untuk Kelas (Clusters): Loop for digunakan untuk iterasi sebanyak n_clusters kali. n_clusters adalah jumlah cluster yang telah ditentukan sebelumnya (dalam contoh ini, 3 cluster).

1. Mencetak Kelas (Clusters): Pada setiap iterasi, kode mencetak informasi tentang kelas (cluster) yang sedang diproses, baik untuk metode Euclidean Distance maupun Cosine Similarity.
2. For Loop untuk Dokumen: Setelah mencetak informasi tentang kelas, ada loop for lagi untuk iterasi sebanyak 5 kali. Ini bertujuan untuk mencetak informasi tentang 5 dokumen pertama yang termasuk dalam kelas tersebut.
3. Hasilnya: Untuk setiap kelas, kode mencetak informasi tentang kelas tersebut (misalnya, "Kelas 1 (Euclidean Distance)") dan kemudian mencetak 5 dokumen pertama yang termasuk dalam kelas tersebut. Ini membantu melihat bagaimana dokumen-dokumen dikelompokkan dalam setiap cluster.

COUNT VECTORIZER

```python
# Menghitung DBI dan Silhouette Score
dbi_euclidean = davies_bouldin_score(X_normalized_sparse.toarray(), labels_euclidean)
dbi_cosine = davies_bouldin_score(cosine_similarities, labels_cosine)
silhouette_euclidean = silhouette_score(X_normalized_sparse.toarray(), labels_euclidean, metric='euclidean')
silhouette_cosine = silhouette_score(pairwise_distances(cosine_similarities,
                                                        metric='cosine'), labels_cosine, metric='precomputed')

# Menampilkan hasil analisis
print("DBI (Euclidean Distance):", dbi_euclidean)
print("DBI (Cosine Similarity):", dbi_cosine)
print("Silhouette Score (Euclidean Distance):", silhouette_euclidean)
print("Silhouette Score (Cosine Similarity):", silhouette_cosine)

# Analisis hasil berdasarkan DBI dan Silhouette Score
if dbi_euclidean < dbi_cosine:
    print("Metode Euclidean Distance lebih baik berdasarkan DBI.")
else:
    print("Metode Cosine Similarity lebih baik berdasarkan DBI.")

if silhouette_euclidean > silhouette_cosine:
    print("Metode Euclidean Distance lebih baik berdasarkan Silhouette Score.")
else:
    print("Metode Cosine Similarity lebih baik berdasarkan Silhouette Score.")
```

```
DBI (Euclidean Distance): 8.15170299550008
DBI (Cosine Similarity): 2.4772374297512485
Silhouette Score (Euclidean Distance): 0.014445223697533108
Silhouette Score (Cosine Similarity): -0.020892957021890875
Metode Cosine Similarity lebih baik berdasarkan DBI.
Metode Euclidean Distance lebih baik berdasarkan Silhouette Score.
```

1. nilai DBI untuk metode Cosine Similarity (2.4772) lebih rendah daripada Euclidean Distance (8.1517), menunjukkan bahwa metode Cosine Similarity menghasilkan clustering yang lebih baik dari pada Euclidean Distance berdasarkan metrik ini.
2. metode Euclidean Distance memiliki Silhouette Score yang positif (0.0144), sedangkan metode Cosine Similarity memiliki Silhouette Score yang lebih rendah dan negatif (-0.0209). Ini mengindikasikan bahwa metode Euclidean Distance mungkin lebih baik dalam menempatkan objek-data dalam cluster mereka.

KESIMPULAN

Berdasarkan evaluasi DBI, metode Cosine Similarity lebih baik dalam hal kualitas clustering. Namun, berdasarkan evaluasi Silhouette Score, metode Euclidean Distance tampaknya lebih baik dalam menempatkan objek-data dalam cluster mereka. Oleh karena itu, kesimpulan dari penelitian ini dapat bervariasi tergantung pada metrik evaluasi yang lebih diutamakan. Dalam pengambilan keputusan akhir, penting untuk mempertimbangkan tujuan analisis clustering dan karakteristik data yang sedang diproses. Jika tujuan adalah untuk memaksimalkan kualitas clustering berdasarkan DBI, maka metode Cosine Similarity mungkin lebih disarankan. Namun, jika prioritas adalah memastikan objek-data ditempatkan dengan baik dalam cluster mereka, maka metode Euclidean Distance mungkin lebih sesuai

Jaccard

Salsabila Aurora Octavia	3322600005
Mutiara Sanny	3322600017
Bayu Kurniawan	3322600019
Calysta Moza Salsabilla	3322600028

Text Mining

JACCARD

```
# Rata-rata skor Jaccard
average_similarity = np.mean(jaccard_scores)
print(f"\nAverage Jaccard Similarity: {average_similarity}")
```

```
Jaccard Similarity Scores:
Jaccard Similarity between 'comp.graphics' and 'rec.sport.baseball': 0.051175584859170994
Jaccard Similarity between 'comp.graphics' and 'soc.religion.christian': 0.05840499937120912
Jaccard Similarity between 'rec.sport.baseball' and 'soc.religion.christian': 0.06601406271827481

Average Jaccard Similarity: 0.37235436598858995
```

Konversi dokumen teks menjadi vektor fitur yang dapat digunakan untuk analisis

1. Perhitungan Skor Jaccard: untuk mengukur kesamaan antara himpunan dokumen dari setiap pasangan topik yang dianalisis.
2. Rata-rata Skor Jaccard: rata-rata dari semua skor Jaccard yang telah dihitung sebagai ukuran kesamaan antara topik-topik yang ada.

```
# Daftar topik
topics = ['comp.graphics', 'rec.sport.baseball', 'soc.religion.christian']

# Dataset 20 Newsgroups
newsgroups = fetch_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'),
                                categories=topics)

# Inisialisasi Vectorizer
vectorizer = CountVectorizer()

# Ubah dokumen menjadi vektor fitur
X = vectorizer.fit_transform(newsgroups.data)

# Inisialisasi matriks untuk menyimpan skor Jaccard
jaccard_scores = np.zeros((len(topics), len(topics)))

# Perhitungan skor Jaccard antar setiap topik
for i in range(len(topics)):
    for j in range(i, len(topics)):
        topic1_index = i
        topic2_index = j

        topic1_documents = X[newsgroups.target == topic1_index]
        topic2_documents = X[newsgroups.target == topic2_index]

        min_num_documents = min(topic1_documents.shape[0], topic2_documents.shape[0])

        topic1_bin = (topic1_documents > 0).astype(int)[:min_num_documents]
        topic2_bin = (topic2_documents > 0).astype(int)[:min_num_documents]

        jaccard = jaccard_score(topic1_bin, topic2_bin, average='micro')

        jaccard_scores[i, j] = jaccard
        jaccard_scores[j, i] = jaccard

# Skor Jaccard
print("Jaccard Similarity Scores:")
for i in range(len(topics)):
    for j in range(i + 1, len(topics)):
        topic1 = topics[i]
        topic2 = topics[j]
        similarity_score = jaccard_scores[i, j]
        print(f"Jaccard Similarity between '{topic1}' and '{topic2}': {similarity_score}")
```

JACCARD

```python
# Calculate Davies-Bouldin Index (DBI)
dbi = davies_bouldin_score(X.toarray(), newsgroups.target)
print(f"Davies-Bouldin Index: {dbi}")

# Calculate Silhouette Score
silhouette_avg = silhouette_score(X.toarray(), newsgroups.target)
print(f"Silhouette Score: {silhouette_avg}")
```

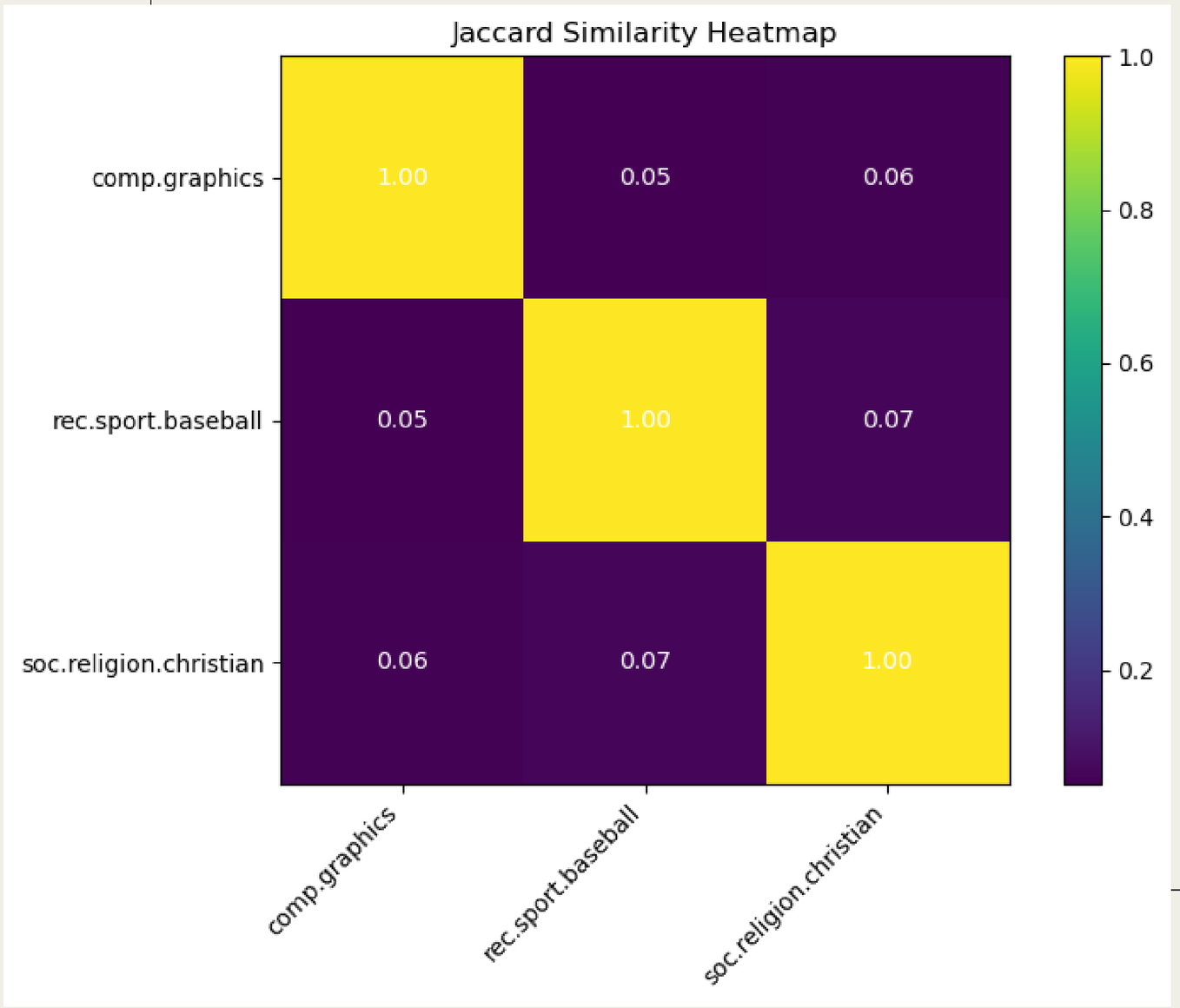
```
Davies-Bouldin Index: 5.175732401958945
Silhouette Score: -0.08983795239954451
```

Nilai DBI 5.175732401958945 menunjukkan bahwa klaster dalam analisis memiliki sejumlah overlap dan kurang optimal. Nilai DBI ini dapat dianggap cukup tinggi, yang mengindikasikan adanya kekurangan dalam kualitas klaster.

DBI Score and Sillhoutte Score

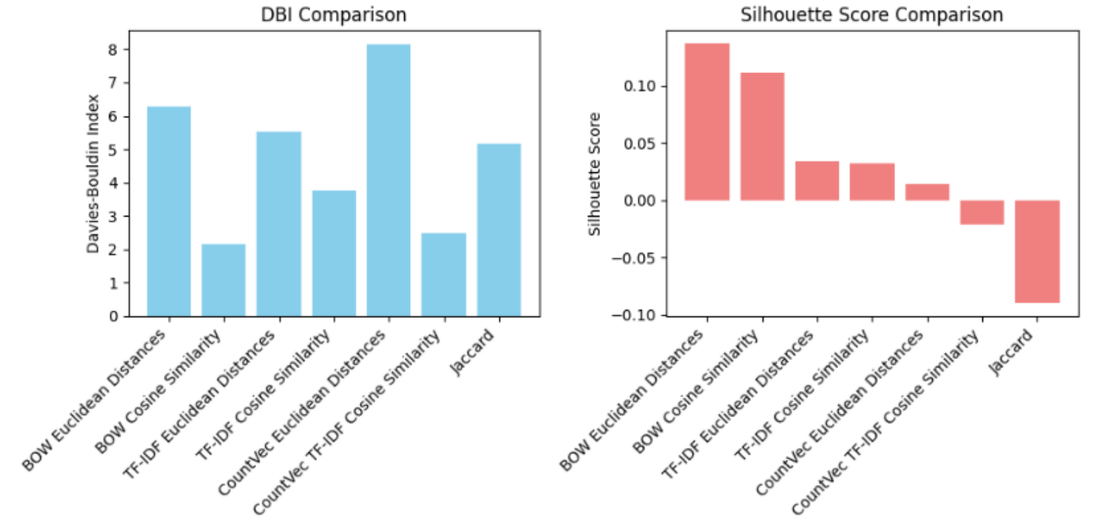
Nilai Silhouette Score -0.08983795239954451 menunjukkan bahwa klaster memiliki sejumlah tumpang tindih dan ketidakjelasan dalam penempatan objek dalam klaster.

HEATMAP JACCARD



PERBANDINGAN SCORE KESELURUHAN MODEL

```
models = ['BOW Euclidean Distances', 'BOW Cosine Similarity', 'TF-IDF Euclidean Distances', 'TF-IDF Cosine Similarity','CountVec
dbi_scores = [6.2838, 2.1591, dbi_ed, dbi_cosine, 8.1517, 2.4772, 5.1757]
ss = [0.1367, 0.111, ss_ed, ss_cosine, 0.0144, -0.0209, -0.0898]

plt.figure(figsize=(10, 5))

# DBI Plot
plt.subplot(1, 2, 1)
plt.bar(models, dbi_scores, color='skyblue')
plt.ylabel('Davies-Bouldin Index')
plt.title('DBI Comparison')
plt.xticks(rotation=45, ha='right')

# Silhouette Score Plot
plt.subplot(1, 2, 2)
plt.bar(models, ss, color='lightcoral')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score Comparison')
plt.xticks(rotation=45, ha='right')

plt.tight_layout()
plt.show()
```



Dari grafik perbandingan score disamping, dapat disimpulkan bahwa dari seluruh model yang dibuat, model yang memiliki nilai DBI score terbaik adalah model dengan vektor Bag of Word dan rumus jarak Cosine Similarity, sedangkan model dengan Silhouette score terbaik adalah model Bag of Word menggunakan rumus jarak euclidean distance. Oleh karena itu, pemilihan model tergantung pada tujuan analisinya, apakah ingin lebih memperhatikan pemisahan kluster (DBI) atau kompaknya objek dalam kluster (Silhouette Score) sesuai dengan metrik yang digunakan.

Walau begitu, kedua model masih memiliki nilai DBI tinggi dan Silhouette Score yang rendah, yang menandakan bahwa kualitas klusteringnya bisa ditingkatkan.