REPORT

GUTENBERG TEXT CLUSTERING

GROUP NUMBER: 10

GROUP MEMBERS: DEEPANRAJ ARUMUGAM MURUGESAN KEERTHIVASAN
SURYANARAYANAN RANJITKUMAR SETHURAMAN
SRUTI SRIRAM

CHOOSING DATA:

We have taken five different samples from the Gutenberg digital books corpus. We have **randomly selected 200 partitions from each book** having 150 words and have labelled them as "a," "b," "c," etc.

BOOK ID	BOOK NAME	GENRE	AUTHOR
1342	Pride and Prejudice	FICTION	Jane Austen
2446	An Enemy of the People	DRAMA	Henrik Ibsen
308	Three men in a boat	HUMOUR	Jerome K
69916	The last space ship	SCIENCE FICTION	Murray L
69790	The Three Just Men	THRILLER	Wallace

PREPROCESSING AND DATA CLEANING:

The books are pre-selected by their book IDs and stored in the list "bi_list". The 200 random partitions of 100 words each are then selected from the processed text.

and stored in a data frame along with the label for each book. Finally, all the data frames are combined into one and saved as a CSV file.

The following python snippet uses the NTLK library to scrape text data from books on the Gutenberg project website.

```
# Creating the Function with 2 parameters
def partition(bi, la):
    url = f"http://www.gutenberg.org/files/{bi}/{bi}-0.txt" # Downloading the book from gutenberg website
    response = urllib.request.urlopen(url)
    raw_text = response.read().decode('utf-8')
    # cleaned_text = re.sub(r'\b\w*\d+\w*\b|\w*_\w*|(?<=\w)\'(?=\w)', '', raw_text).strip()
    cleaned_text = re.sub(r'\r\n?', '\n', raw_text).strip() # Cleaning the text
    wo = re.findall(r'\w+', cleaned_text) # Splitting to list of words
    random.seed(69)
    starting_indices = random.sample(range(0, len(wo) - 100), 200) # Selecting 200 random starting indices for partitions
    partitions = [] #creating empty list for partitions
    for i in starting_indices: #iterating through the starting indices
        partition = wo[i:i+150]
        partitions.append(partition) #appending each partition to the partitions list
    df = pd.DataFrame({'label': la, 'partition': partitions}) # Creating a DF to store all the partitions
    return df #the dataframe is returned
```

```
df["partition"]=df["partition"].apply(lambda x: " ".join([w for w in x.split() if len(w)>2]))
```

1. Cleaning the raw text: The newline characters "\r\n" in the text are replaced with "\n" and the text is stripped of any whitespaces at the beginning and end.
2. Splitting the cleaned text into words: The cleaned text is then split into a list of words using the re.findall() method and a regular expression pattern "\w+".
3. Also, we have removed the words which are less than 2 letters, as

FEATURE ENGINEERING:

We have performed LDA on the text documents which have been represented as BOW and TF-IDF.

```
# Extracting the book labels and text partitions
book_labels = df["label"].values
text_partitions = df["partition"].values

# Preprocessing the text data
vectorizer_bow = CountVectorizer(max_features=1000, stop_words="english")
vectorizer_tfidf = TfidfVectorizer(max_features=1000, stop_words="english")

X_bow = vectorizer_bow.fit_transform(text_partitions)
X_tfidf = vectorizer_tfidf.fit_transform(text_partitions)

# Training the LDA model
lda = LatentDirichletAllocation(n_components=10, random_state=42)
X_lda = lda.fit_transform(X_tfidf)
```

For extracting topics, the LatentDirichletAllocation is used with n_components=10 indicating that 10 topics must be extracted from the data.

CLUSTERING ALGORITHMS:

The three clustering algorithms used are:

- **K-means**
- **EM algorithm**
- **Hierarchical clustering**

The code above performs k-means clustering on the text data represented as bag-of-words (BoW), TF-IDF, and Latent Dirichlet Allocation (LDA) vectors, and calculates the Adjusted Rand Index (ARI) scores for different numbers of clusters.

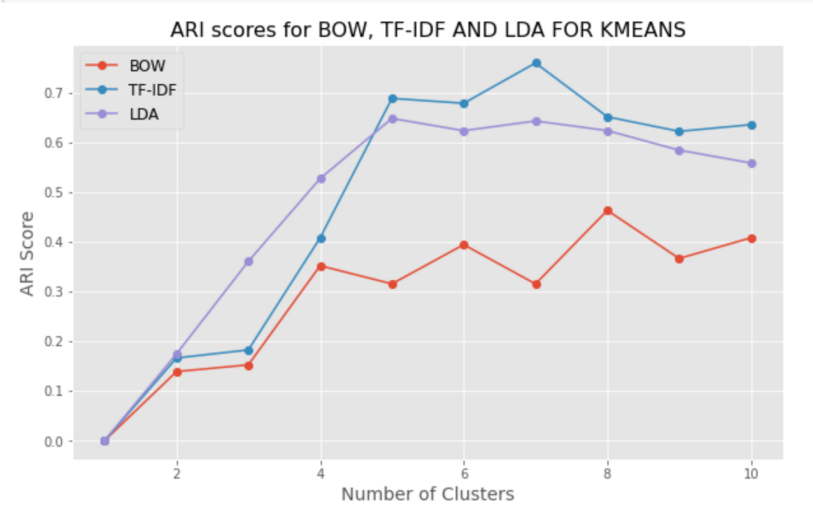
The ARI score measures the similarity between the true labels of the data and the cluster assignments generated by the k-means algorithm.

```
# Creating 10 clusters
n_clusters_range = range(1, 11)
ari_scores_bow = []
ari_scores_tfidf = []
ari_scores_lda = []

for n_clusters in n_clusters_range:

    # Using k-means clustering
    kmeans_bow = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans_bow.fit(X_bow)
    y_pred_bow = kmeans_bow.labels_

    kmeans_tfidf = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans_tfidf.fit(X_tfidf)
    y_pred_tfidf = kmeans_tfidf.labels_

    kmeans_lda = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans_lda.fit(X_lda)
    y_pred_lda = kmeans_lda.labels_
```

The purpose of using Adjusted Rand Index (ARI) scores is to evaluate the performance of a clustering algorithm in assigning data points to clusters.

PLOTTING:

Based on the plot obtained, we visually inspected it to identify the point of inflection using the elbow method.



The ARI scores for different numbers of clusters (ranging from 1 to 10) are plotted for three different vectorization methods (BoW, TF-IDF, and LDA) using k-means clustering. Based on the plot, the ARI scores for all three methods seem to increase initially and then start to level off around 5 clusters. **Therefore, 5 clusters could be considered the breaking point or optimal number of clusters according to the elbow method.**

```
# Clustering the data using K-means
n_clusters = 5
kmeans_bow = KMeans(n_clusters=n_clusters, random_state=42)
kmeans_bow.fit(X_bow)
y_pred_bow_kmeans = kmeans_bow.labels_

kmeans_tfidf = KMeans(n_clusters=n_clusters, random_state=42)
kmeans_tfidf.fit(X_tfidf)
y_pred_tfidf_kmeans = kmeans_tfidf.labels_

kmeans_lda = KMeans(n_clusters=n_clusters, random_state=42)
kmeans_lda.fit(X_lda)
y_pred_lda_kmeans = kmeans_lda.labels_
```

```
# Clustering the data using EM algorithm
em_bow = GaussianMixture(n_components=n_clusters, random_state=42)
em_bow.fit(X_bow.toarray())
y_pred_bow_em = em_bow.predict(X_bow.toarray())

em_tfidf = GaussianMixture(n_components=n_clusters, random_state=42)
em_tfidf.fit(X_tfidf.toarray())
y_pred_tfidf_em = em_tfidf.predict(X_tfidf.toarray())

em_lda = GaussianMixture(n_components=n_clusters, random_state=42)
em_lda.fit(X_lda)
y_pred_lda_em = em_lda.predict(X_lda)
```

```
# Clustering the data using hierarchical clustering algorithm
hierarchical_bow = AgglomerativeClustering(n_clusters=n_clusters)
y_pred_bow_hierarchical = hierarchical_bow.fit_predict(X_bow.toarray())

hierarchical_tfidf = AgglomerativeClustering(n_clusters=n_clusters)
y_pred_tfidf_hierarchical = hierarchical_tfidf.fit_predict(X_tfidf.toarray())

hierarchical_lda = AgglomerativeClustering(n_clusters=n_clusters)
y_pred_lda_hierarchical = hierarchical_lda.fit_predict(X_lda)
```

The code above performs clustering on the data using three different clustering algorithms: K-means, EM algorithm, and hierarchical clustering. For each algorithm, the code fits the data to the algorithm and predicts the cluster labels.

For K-means and hierarchical clustering, the code uses the scikit-learn implementations of these algorithms, while for the EM algorithm, the code uses the GaussianMixture class from scikit-learn.

After fitting and predicting the cluster labels for each algorithm, the resulting labels are stored in variables with names indicating the algorithm used and the type of feature used (BOW, TF-IDF, or LDA). Specifically, for each algorithm, there are variables storing the predicted labels using BOW features, TF-IDF features, and LDA features.

EVALUATING THE CLUSTERING PERFORMANCE:

The three clustering algorithms are represented in three different feature representations: BOW, TF-IDF and LDA.

1. ARI (Adjusted Rand Index)

The ARI score is used to evaluate the performance of these three clustering methods.

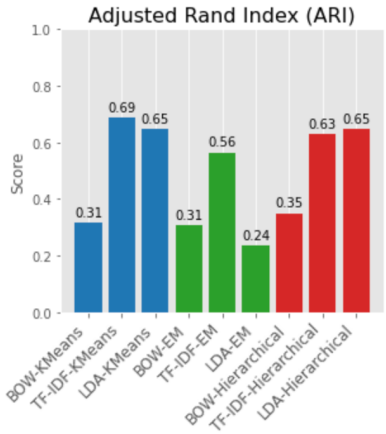
Apart from Kappa and Silhouette score, we have also chosen ARI as a measure to evaluate the model's performance. The reason why we went with ARI because it does not depend on the labeling of the clusters, making it useful for comparing different clustering algorithms or for parameter finetuning.

```
# Evaluating the clustering performance (ARI) of the above clustering algorithms
ari_bow_kmeans = adjusted_rand_score(book_labels, y_pred_bow_kmeans)
ari_tfidf_kmeans = adjusted_rand_score(book_labels, y_pred_tfidf_kmeans)
ari_lda_kmeans = adjusted_rand_score(book_labels, y_pred_lda_kmeans)

ari_bow_em = adjusted_rand_score(book_labels, y_pred_bow_em)
ari_tfidf_em = adjusted_rand_score(book_labels, y_pred_tfidf_em)
ari_lda_em = adjusted_rand_score(book_labels, y_pred_lda_em)

ari_bow_hierarchical = adjusted_rand_score(book_labels, y_pred_bow_hierarchical)
ari_tfidf_hierarchical = adjusted_rand_score(book_labels, y_pred_tfidf_hierarchical)
ari_lda_hierarchical = adjusted_rand_score(book_labels, y_pred_lda_hierarchical)
```

ARI score ranges between -1 and 1.

A score of 0 indicates that the true clustering labels and the results are independent of each other. A score of -1 indicates a complete disagreement between the two clusterings. A higher ARI score indicates better clustering performance.
The below output shows the ARI scores for the different clustering methods we have used.

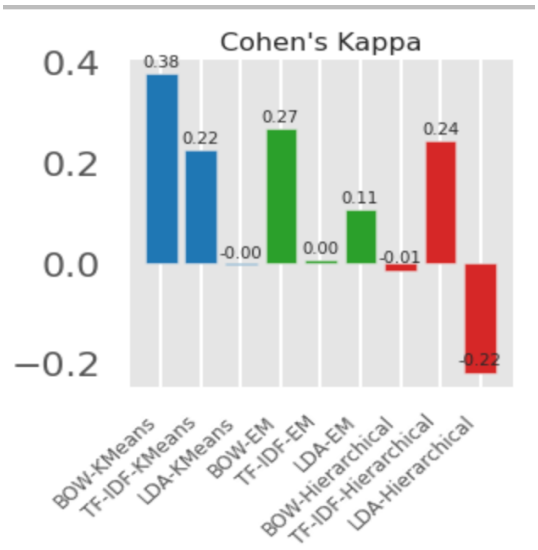
```
ARI for BOW clustering with K-means: 0.315
ARI for TF-IDF clustering with K-means: 0.688
ARI for LDA clustering with K-means: 0.648
ARI for BOW clustering with EM: 0.309
ARI for TF-IDF clustering with EM: 0.565
ARI for LDA clustering with EM: 0.237
ARI for BOW clustering with hierarchical clustering: 0.350
ARI for TF-IDF clustering with hierarchical clustering: 0.628
ARI for LDA clustering with hierarchical clustering: 0.648
```



2. Cohen Kappa Score

0s

```python
# Calculate the Cohen's kappa score
kappa_bow_kmeans = cohen_kappa_score(book_labels, y_pred_bow_kmeans)
kappa_tfidf_kmeans = cohen_kappa_score(book_labels, y_pred_tfidf_kmeans)
kappa_lda_kmeans = cohen_kappa_score(book_labels, y_pred_lda_kmeans)

kappa_bow_em = cohen_kappa_score(book_labels, y_pred_bow_em)
kappa_tfidf_em = cohen_kappa_score(book_labels, y_pred_tfidf_em)
kappa_lda_em = cohen_kappa_score(book_labels, y_pred_lda_em)

kappa_bow_hierarchical = cohen_kappa_score(book_labels, y_pred_bow_hierarchical)
kappa_tfidf_hierarchical = cohen_kappa_score(book_labels, y_pred_tfidf_hierarchical)
kappa_lda_hierarchical = cohen_kappa_score(book_labels, y_pred_lda_hierarchical)
```



From the kappa values obtained, we can see that the BOW clustering with K-means has the highest kappa value of 0.376, indicating a moderate level of agreement. The TF-IDF clustering with K-means has a lower kappa value of 0.225, indicating a fair level of agreement. The LDA clustering with K-means has a very low kappa value of -0.004, indicating worse than random agreement.

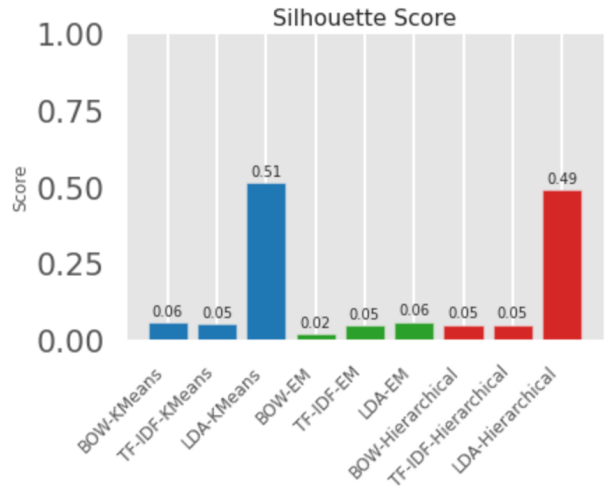
For the EM algorithm, the BOW clustering has a kappa value of 0.266, indicating a fair level of agreement. The TF-IDF clustering has a very low kappa value of 0.005, indicating almost random agreement. The LDA clustering has a kappa value of 0.106, indicating a slight level of agreement.

For hierarchical clustering, the BOW clustering has a negative kappa value of -0.015, indicating worse than random agreement. The TF-IDF clustering has a higher kappa value of 0.241, indicating a fair level of agreement. The LDA clustering has the lowest kappa value of -0.219, indicating worse than random agreement.

Overall, the BOW clustering with K-means seems to be the best performing clustering algorithm in terms of kappa values, followed by TF-IDF clustering with K-means. The LDA clustering algorithm does not perform well in comparison to the other algorithms.

3. Silhouette score:

```
# Calculate the silhouette score
silhouette_bow_kmeans = silhouette_score(X_bow, y_pred_bow_kmeans)
silhouette_tfidf_kmeans = silhouette_score(X_tfidf, y_pred_tfidf_kmeans)
silhouette_lda_kmeans = silhouette_score(X_lda, y_pred_lda_kmeans)

silhouette_bow_em = silhouette_score(X_bow, y_pred_bow_em)
silhouette_tfidf_em = silhouette_score(X_tfidf, y_pred_tfidf_em)
silhouette_lda_em = silhouette_score(X_lda, y_pred_lda_em)

silhouette_bow_hierarchical = silhouette_score(X_bow, y_pred_bow_hierarchical)
silhouette_tfidf_hierarchical = silhouette_score(X_tfidf, y_pred_tfidf_hierarchical)
silhouette_lda_hierarchical = silhouette_score(X_lda, y_pred_lda_hierarchical)
```



Silhouette score measures the compactness and separation of the clusters, with values ranging from -1 to 1. Higher values indicate better-defined clusters.

From the given silhouette scores, we can infer that the LDA clustering algorithm with K-means has the highest silhouette score of 0.512, indicating well-separated and compact clusters.

On the other hand, the BOW clustering with EM has the lowest silhouette score of 0.020, indicating that the clusters may not be well-defined or may be overlapping.

The other clustering algorithms have moderate silhouette scores, indicating relatively well-defined clusters, but not as well-defined as the LDA clustering with K-means.

4. Coherence Score:

```
# Tokenizing the partitions and creating a dictionary
partition_tokens = [partition.split() for partition in df['partition']]
dictionary = corpora.Dictionary(partition_tokens)

# Creating a bag-of-words corpus from the dictionary and partitions
corpus = [dictionary.doc2bow(partition) for partition in partition_tokens]

# Training an LDA model with 5 topics and 65 passes over the corpus
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus, id2word=dictionary, num_topics=5, passes=65)

# Calculating coherence score using C_v metric
coherence_model_lda = CoherenceModel(model=lda_model, texts=partition_tokens, dictionary=dictionary, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()

print('Coherence Score:', coherence_lda)
```

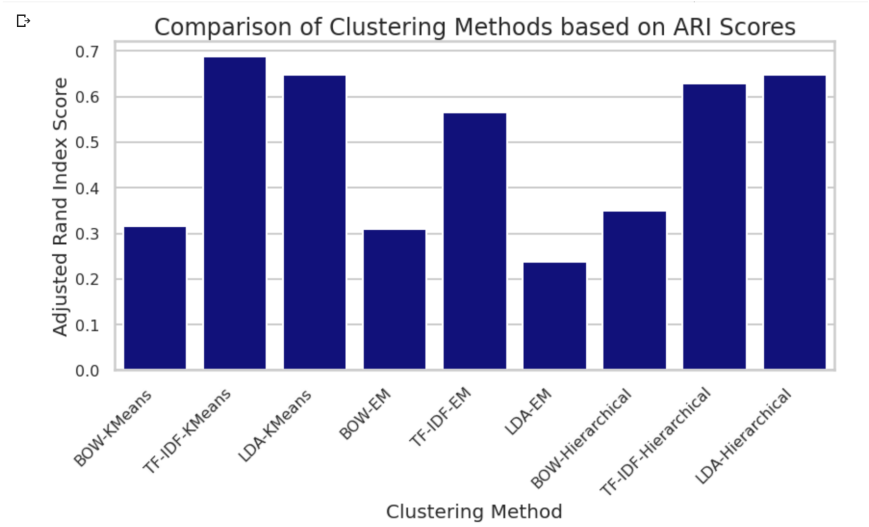
```
Coherence Score: 0.48138466122641416
```

The coherence score is a measure of how interpretable and meaningful the topics generated by the LDA model are.

The score ranges from 0 to 1, with higher values indicating more coherent and meaningful topics.

In this case, we got a coherence score of 0.48 which suggests that the topics generated by the LDA model are relatively coherent and meaningful.

COMPARISON OF CLUSTERING MODELS:



Looking at the ARI scores, we can see that the highest scores are achieved with TF-IDF clustering with K-means (0.688) and LDA clustering with K-means (0.648). The ARI score for LDA clustering with hierarchical clustering (0.648) is also high. This indicates that these clustering methods are more accurate in terms of cluster similarity with the human labels.

In terms of Kappa score, BOW clustering with K-means (0.376) and TF-IDF clustering with K-means (0.225) have the highest scores. However, LDA clustering with K-means has a negative kappa score (-0.004), indicating that the agreement between the clustering results and human labels is worse than expected by chance.

Looking at the Silhouette scores, we can see that LDA clustering with K-means has the highest score (0.512), indicating that the clusters are well-separated and compact. The Silhouette score for LDA clustering with hierarchical clustering is also high (0.490).

Finally, the coherence score for the LDA model is 0.481, indicating that the topics generated by the LDA model are meaningful and coherent.

Overall, based on the above evaluation metrics, we have found that the K-means clustering method with all the features except LDA have shown to perform closely to the human labels in all evaluation metrics.

ERROR ANALYSIS:

```
# Download stopwords
nltk.download('stopwords')

# Define the stop words
stop_words = set(stopwords.words('english'))
df["partition"] = df["partition"].str.lower()

# Remove stop words from the corpus
df["partition"] = df["partition"].apply(lambda x: ' '.join([word for word in x.split() if word not in stop_words]))

# Get the top 10 frequent words in the corpus
word_counter = Counter(df["partition"].str.split(expand=True).stack())
top_words = [word for word, count in word_counter.most_common(10)]

# Print the top 10 frequent words
print("Top 10 frequent words:")
print(top_words)
```

```
Top 10 frequent words:
['stockmann', 'said', 'would', 'one', 'could', 'man', 'know', 'mrs', 'kim', 'two']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

The top 10 frequent words in our dataset were found to be: -

`['stockmann', 'said', 'would', 'one', 'could', 'man', 'know', 'mrs', 'kim', 'two']`

We believe that the following reasons might have thrown the models off track and act as obstacles in the clustering process.

1. **Frequency:** The frequency of the words in the dataset could affect the clustering performance. If some words occur much more frequently than others, they may dominate the clustering process and cause other words to be overlooked.

2. **Ambiguity:** The ambiguity of the words could also affect clustering performance. For example, the word "man" could refer to an adult male, but it could also refer to humankind in general. This ambiguity could make it difficult for a clustering algorithm to accurately group similar concepts together.

3. **Context:** The context in which the words are used could affect clustering performance. For example, if the word "mrs" is frequently used in a particular context (e.g., as a title before a woman's name), this could affect how the clustering algorithm groups words together.

4. **Part-of-speech:** The part-of-speech (POS) of the words could also affect clustering performance. Nouns, verbs, and adjectives, for example, may have different semantic relationships with each other, and a clustering algorithm that is not sensitive to POS could group unrelated words together.

5. **Semantic similarity:** The semantic similarity between the words could affect clustering performance. If two words have similar meanings or are often used together in the same context, a clustering algorithm may group them together even if they are not the same part of speech or have different frequencies.