Project Report: Finding Similar Clinical Trials

A project report for CSD361: Introduction to Machine Learning

Guided by
Harish Chandra Karnick
(Professor at Shiv Nadar University)

Submitted By:
Shruti Bansal(2110110492)

Date: 3rd December, 2023

1 Project Description

1.1 Background

The ability to search for clinical trials that are similar to an index trial can be very useful for trial participants, investigators, and researchers interested in synthesizing medical evidence. This project focuses on clustering Chimeric Antigen Receptor (CAR) T Cell clinical trials based on attributes available in their respective records on ClinicalTrials.gov.

1.2 Goals

The main goals of the project are to cluster CAR T cell clinical trials using relevant attributes and visualize/describe the identified clusters using enrichment analysis or similar methods.

1.3 Data Availability

The data for this project is sourced from ClinicalTrials.gov, a publicly accessible registry of clinical trials. A local instance of the database is available for students working on the project. Additionally, NCT ids for CAR T cell trials are provided in a CSV file. Eligibility criteria are identified, and medical concepts are mapped to Unified Medical Language System (UMLS) identifiers. This database is then provided to us in the form of smaller tables.

1.4 Additional Resources

Eligibility criteria for patients in each clinical trial are defined and mapped to medical concepts in UMLS. A 'bag of medical concepts' is associated with each CAR T cell clinical trial.

1.5 Evaluation

The clustering performance will be evaluated based on Silhouette score and Callinski-Harabasz Index. Cluster stability will be assessed through resampling and comparing clusters from the original and resampled data.

2 Methodology

2.1 Data Preprocessing

The initial steps involve handling missing values and preparing the data for clustering. Numeric columns are extracted from the dataset, and missing values are filled with zeros. The data is then scaled using StandardScaler to ensure uniformity across numeric features.

Text columns are processed separately by filling missing values with empty strings. The text data is further transformed using the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer. This vectorizer converts the raw text data into numerical features, representing the importance of each word in the context of the entire dataset. The TF-IDF features capture both the frequency of a term in a document and its rarity across all documents.

The TF-IDF formula is given by:

$$TF - IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

Where:

- $TF(t, d)$ is the Term Frequency of term t in document d, representing the frequency of t in d.
- $IDF(t, D)$ is the Inverse Document Frequency of term t in the entire dataset D, given by $\log\left(\frac{Total\ number\ of\ documents\ in\ D}{Number\ of\ documents\ containing\ term\ t}\right)$.

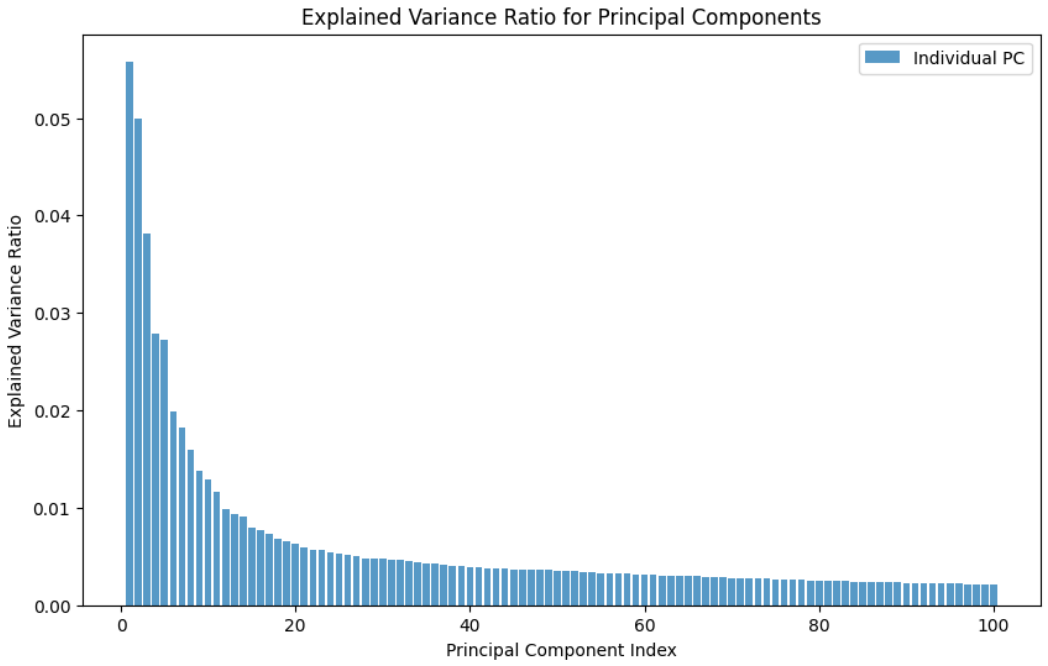
```
numeric_df = df[float_columns]
numeric_df.fillna(0, inplace=True)
scaler = StandardScaler()
numeric_df_scaled = scaler.fit_transform(numeric_df)

text_df = df[str_columns]
text_df.fillna("", inplace=True)

vectorizer = TfidfVectorizer(stop_words='english', max_features=500)
text_features = vectorizer.fit_transform(text_df.apply(lambda x: ' '.join(x), axis=1))

combined_features = pd.concat([pd.DataFrame(numeric_df_scaled, columns=numeric_df.columns),
                    pd.DataFrame(text_features.toarray(), columns=vectorizer.get_feature_names_out())],
                    axis=1)

scaler = StandardScaler()
combined_features_standardized = scaler.fit_transform(combined_features)
```

2.2 Dimensionality Reduction

Principal Component Analysis (PCA) is applied to reduce the dimensionality of the combined features.

```
from sklearn.decomposition import PCA

pca = PCA()
pca_result = pca.fit_transform(combined_features_standardized)
explained_variance_ratio = pca.explained_variance_ratio_

```

```
plt.figure(figsize=(10, 6))
plt.bar(range(1, 101), explained_variance_ratio[:100], alpha=0.75,
                    align='center', label='Individual PC')
plt.xlabel('Principal Component Index')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio for Principal Components')
plt.legend()
plt.show()
```

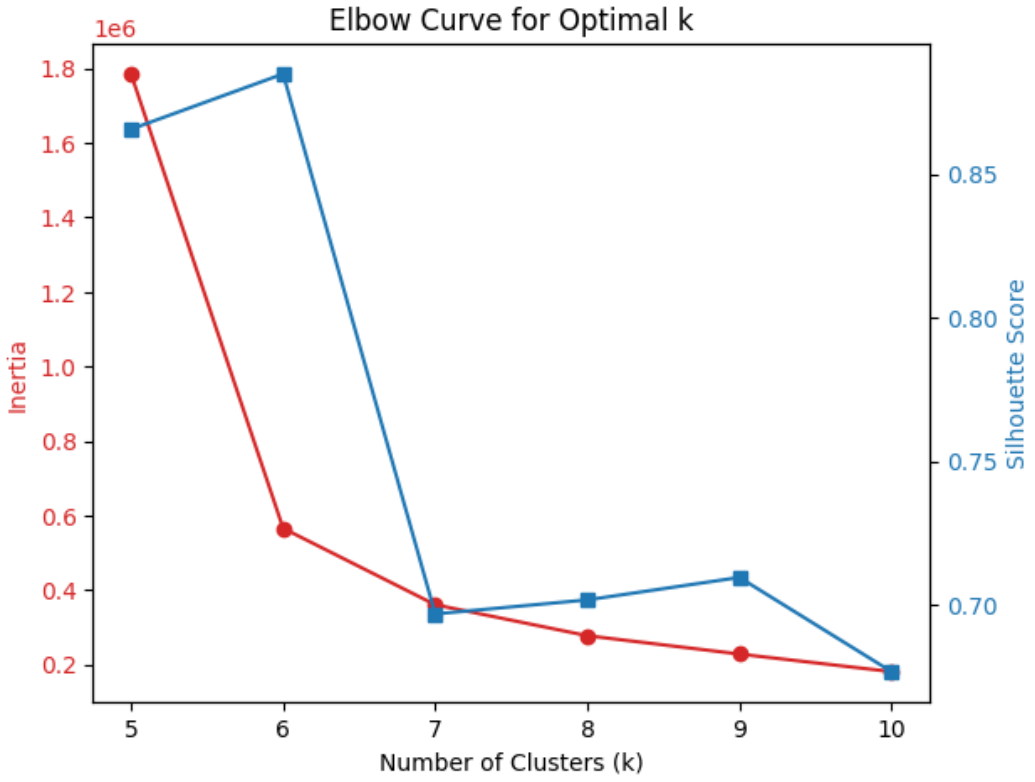


2.3 Clustering and Evaluation

The K-means clustering algorithm is applied to the selected PCA features. The optimal number of clusters is determined based on Silhouette Score and Calinski-Harabasz Index. The results are visualized to aid in choosing the appropriate cluster size.

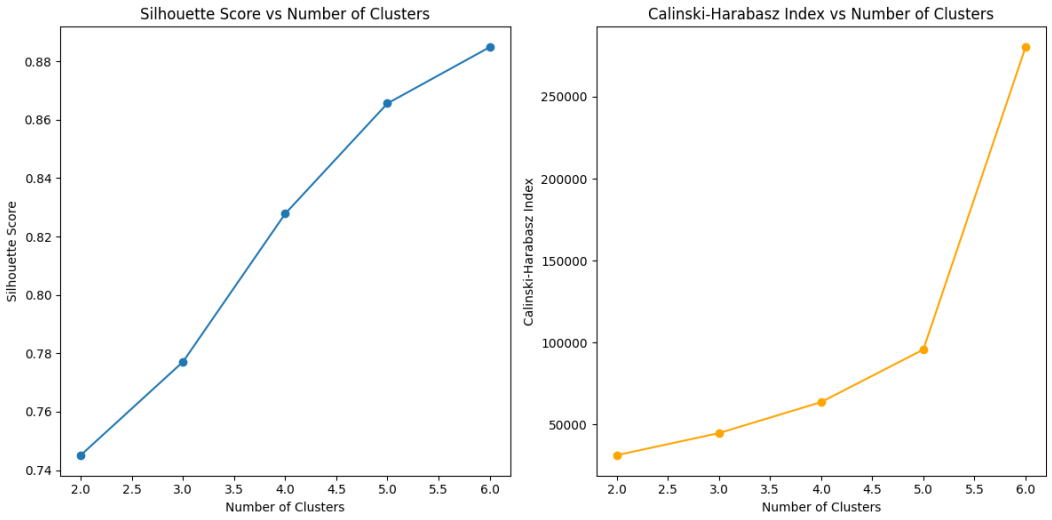
```
num_components_to_select = 5
selected_pca_features = pca_result[:, :num_components_to_select]

inertia = []
silhouette_scores = []
k_values = range(5, 11)  # You can adjust the range of k values

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(selected_pca_features)
    inertia.append(kmeans.inertia_)
    silhouette_scores.append(silhouette_score(selected_pca_features, kmeans.labels_))

# Plotting the elbow curve
```

```
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('Number of Clusters (k)')
ax1.set_ylabel('Inertia', color=color)
ax1.plot(k_values, inertia, 'o-', color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()
color = 'tab:blue'
ax2.set_ylabel('Silhouette Score', color=color)
ax2.plot(k_values, silhouette_scores, 's-', color=color)
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()
plt.title('Elbow Curve for Optimal k')
plt.show()
```




```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, calinski_harabasz_score
import matplotlib.pyplot as plt

num_clusters_range = range(2, 7)
silhouette_scores = []
calinski_harabasz_indices = []
```

```
for num_clusters in num_clusters_range:
    kmeans = KMeans(n_clusters=num_clusters, random_state=42)
    labels = kmeans.fit_predict(selected_pca_features)

    silhouette_scores.append(silhouette_score(selected_pca_features, labels))
    calinski_harabasz_indices.append(calinski_harabasz_score(selected_pca_features, labels))

# Plotting the results
plt.figure(figsize=(12, 6))

# Silhouette Score
plt.subplot(1, 2, 1)
plt.plot(num_clusters_range, silhouette_scores, marker='o')
plt.title('Silhouette Score vs Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')

# Calinski-Harabasz Index
plt.subplot(1, 2, 2)
plt.plot(num_clusters_range, calinski_harabasz_indices, marker='o', color='orange')
plt.title('Calinski-Harabasz Index vs Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Calinski-Harabasz Index')

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



2.4 Cluster Analysis

The optimal cluster size is determined to be 6 based on the evaluation metrics. The final step involves analyzing the clusters and extracting key features for each cluster.

```
original_feature_indices = range(num_components_to_select)
feature_names = [f'PC_{i}' for i in original_feature_indices]

ordered_centroids = kmeans.cluster_centers_.argsort()[:, ::-1]

for cluster_num in range(6):
    key_features = [feature_names[index] for index in ordered_centroids[cluster_num, :]]

    print(f'Cluster #{cluster_num + 1}')
    print('Key Features:', key_features)
    print('-' * 80)
```

```
Cluster #1
Key Features: ['PC_3', 'PC_4', 'PC_1', 'PC_2', 'PC_0']
--------------------------------------------------------------------------------
Cluster #2
Key Features: ['PC_1', 'PC_3', 'PC_0', 'PC_4', 'PC_2']
--------------------------------------------------------------------------------
Cluster #3
Key Features: ['PC_4', 'PC_3', 'PC_1', 'PC_0', 'PC_2']
--------------------------------------------------------------------------------
Cluster #4
Key Features: ['PC_2', 'PC_3', 'PC_1', 'PC_0', 'PC_4']
--------------------------------------------------------------------------------
Cluster #5
Key Features: ['PC_4', 'PC_2', 'PC_1', 'PC_0', 'PC_3']
--------------------------------------------------------------------------------
Cluster #6
Key Features: ['PC_0', 'PC_3', 'PC_4', 'PC_2', 'PC_1']
--------------------------------------------------------------------------------
```

```
for pc_index in range(num_components_to_select):
    loadings = pca.components_[pc_index]
    loading_df = pd.DataFrame({'Feature': combined_features.columns, 'Loading': loadings})
    loading_df = loading_df.reindex(loading_df['Loading'].abs().sort_values(ascending=False).index)
    top_features = loading_df.head(10)
    print(f'PC_{pc_index}')
    print(top_features)
    print('-' * 80)
```

```
PC_0
          Feature   Loading
277      included  0.176921
459           set  0.176414
159       cohorts  0.174524
179        cutoff  0.173422
84             61  0.172650
85             64  0.171291
76             35  0.169822
79             47  0.169044
444  respectively  0.167595
112           apr  0.166366
-------------------------------------------------
PC_1
            Feature   Loading
442         reports  0.186166
166        concepts  0.186049
488  systematically  0.185795
135        captured  0.185343
177              cu  0.185146
118     attribution  0.182574
212              dw  0.179919
301          jensen  0.178443
46              124  0.177844
```