Text Clustering

Ricardo Alamo
Fengyuexin Huang
Charles Ohiri

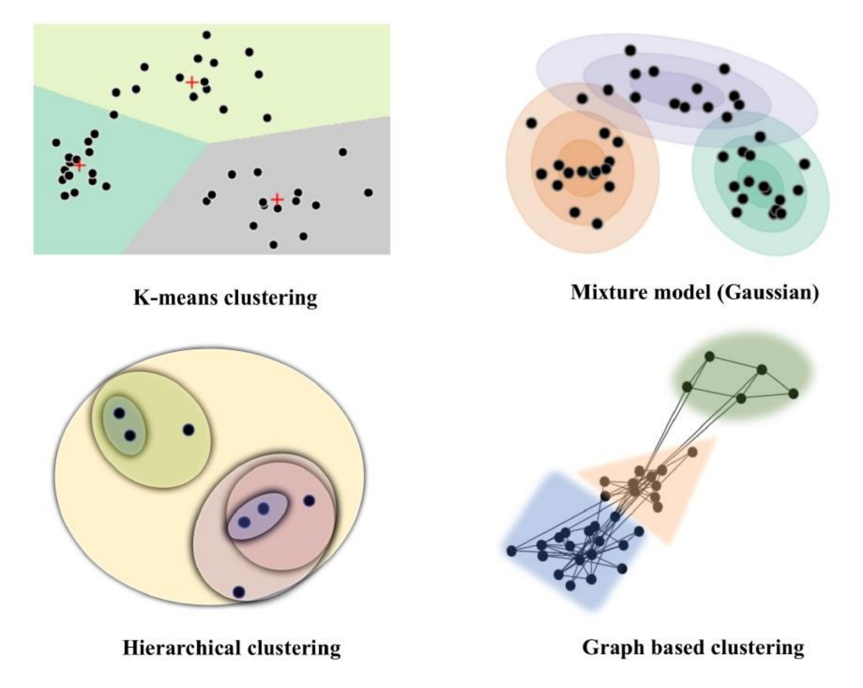
Table of Contents

- Introduction
  - Chosen ML Algorithm and its Industry Application
  - Popular Alternatives
- ML Algorithm Implementation
  - Advantages
- Disadvantages
- Code Breakdown
- Conclusion

Introduction

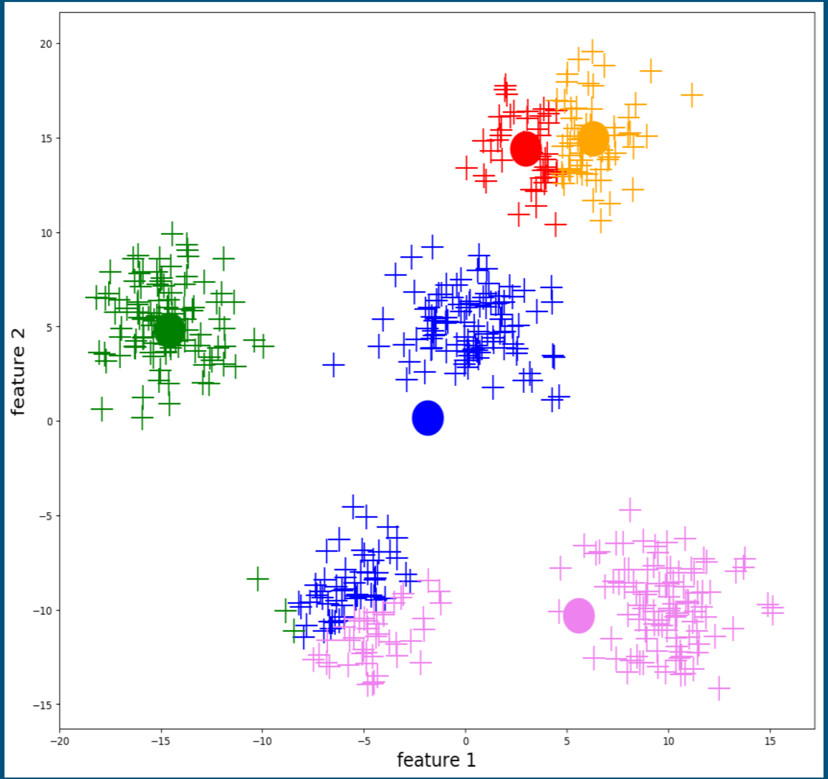
Text Clustering

- Text clustering is a subfield of unsupervised machine learning that involves grouping together similar textual data into clusters
- The aim of text clustering is to automatically discover patterns or structure in unstructured textual data, without any prior knowledge or labeling of the data.
- Common uses include:
  - Fraud Prevention
  - Theme Identification
  - HealthCare



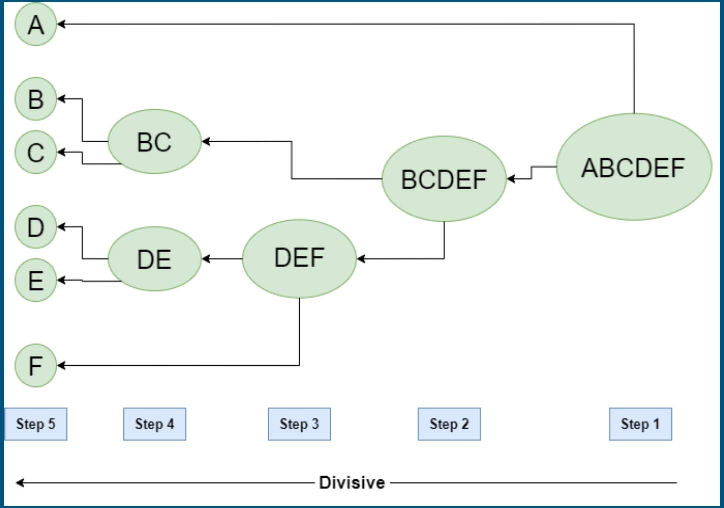
K-Means

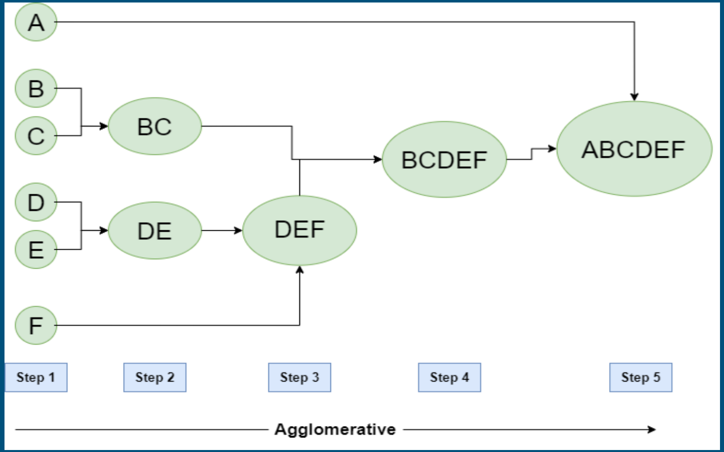
- This is a form of hard clustering where every object belongs to exactly one cluster.
    - it is an efficient, effective, and simple clustering algorithm.
- It is centroid-based clustering where data is organized into non-hierarchical clusters.
- For this method to work perfectly, the number of clusters, k, must be predefined and the data cannot be sparse.
    - Truncated Singular Value Decomposition (SVD) is a matrix factorization technique used for dimensionality reduction of high-dimensional data.
    - The elbow method is a graphical technique used to determine the optimal number of clusters



Alternative Algorithms

- Density-Based Clustering
  - Density-based clustering connects areas of high example density into clusters.
  - This allows for arbitrary-shaped distributions as long as dense areas can be connected
  - These algorithms have difficulty with data of varying densities and high dimensions.
    - DBSCAN
    - Spectral Clustering
- Hierarchical Clustering
  - This creates a tree of clusters
  - The algorithm starts with each data point as its own cluster and then iteratively merges clusters together until a stopping criterion is met

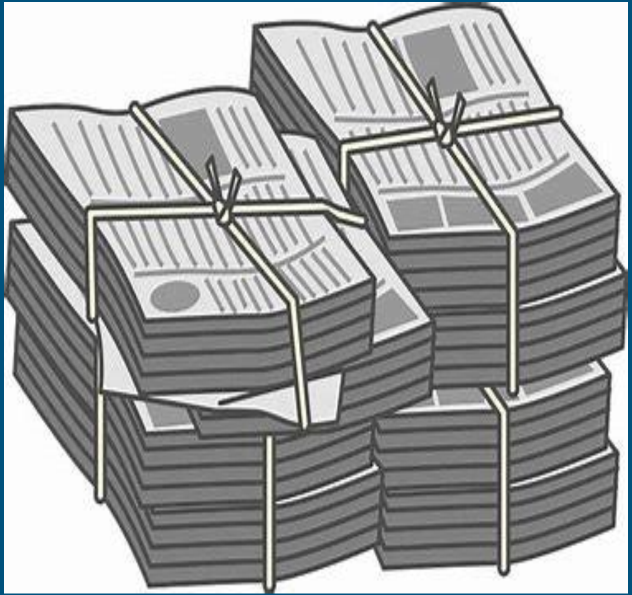




ML Algorithm Implementation

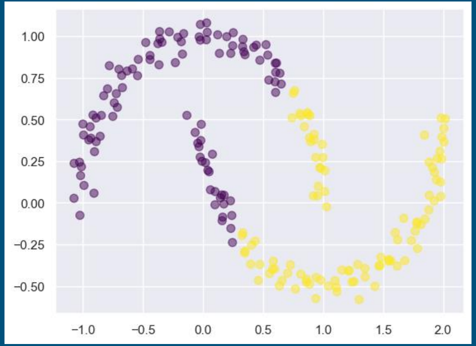
T-news Dataset

- This is a famous dataset that contains documents by 20 different newsgroups
    - It contains 4000 different documents
    - For our project, we chose the first 100 documents
- News data often has distinct topics or themes that can be easily separated into clusters
- We chose K-means for:
    - its simplicity and and ease of understanding
    - Its interpretability
    - Its ability to handle continuous data effectively
        - News data after vectorization becomes continuous data



Disadvantages:

1. Requires a predefined number of clusters (k) to be specified.
2. Sensitive to initial cluster centroids, which can result in different solutions.
3. Can converge to a local minimum, rather than the global minimum.
4. Assumes that the clusters are spherical and equally sized, which may not be the case in all datasets.
5. Can be biased towards the mean of the data and may not work well with non-linear data.



Not the best for non-spherical shapes.

CODE BREAKDOWN

```
# Define your list of stop words
my_additional_stop_words = ['com', 'wa', 'ha','did']
stop_words = text.ENGLISH_STOP_WORDS.union(my_additional_stop_words)

# Define a function to tokenize and lemmatize a document
def tokenize_and_lemmatize(art):
    artmod1 = re.sub(r'\d+', ' ', art)
    art_mod2 = re.sub(r"[^a-zA-Z0-9]+", ' ', artmod1)
    # Tokenize the document
    tokens = word_tokenize(art_mod2)
    # Lemmatize each token
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
    # Remove stop words
    filtered_tokens = [token for token in lemmatized_tokens if token not in stop_words]
    # Return the filtered tokens as a string
    return " ".join(filtered_tokens)

# Tokenize, lemmatize, and remove stop words from the documents
preprocessed_docs = [tokenize_and_lemmatize(art) for art in book_corpus]

# Define the vectorizer with stop words removed
vectorizer = TfidfVectorizer(stop_words=stop_words)

# Fit and transform the vectorizer on the preprocessed documents
tfidf_matrix = vectorizer.fit_transform(preprocessed_docs)
news = pd.DataFrame(tfidf_matrix.toarray(), columns= vectorizer.get_feature_names_out())
```

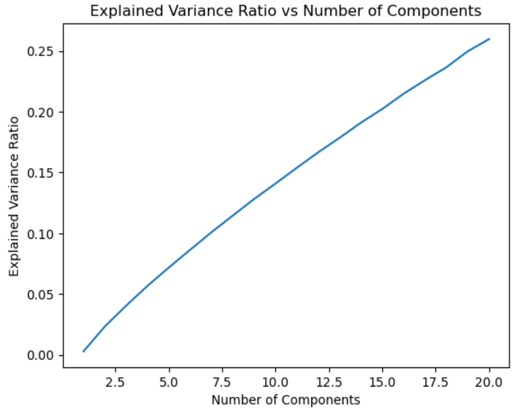
Before running any model or dimension reduction the text needs to be cleaned.

- Creating a personalized stop word list
- Use regex to remove all numbers and symbols.
- Tokenize to get every words.
- Lemmatization of all the words.
- Using TFID to vectorize the words.

CODE BREAKDOWN

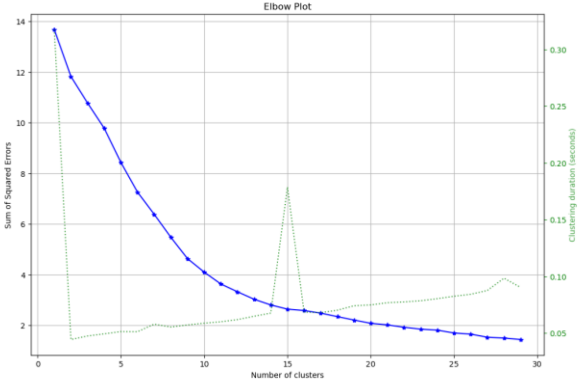
4. KMeans

```
In [15]: # Define the range of number of components to test
n_components_range = range(1, 21)

# Define an empty list to store the explained variance ratios for each number of components
explained_variances = []

# Loop over the range of number of components
for n_components in n_components_range:

    # Initialize TruncatedSVD with the current number of components
    svd = TruncatedSVD(n_components=n_components, random_state=42)

    # Fit and transform the Tfidf matrix using the current TruncatedSVD instance
    X_svd = svd.fit_transform(tfidf_matrix)

    # Calculate the explained variance ratio for the current TruncatedSVD instance
    explained_variance = np.sum(svd.explained_variance_ratio_)

    # Append the explained variance ratio to the list
    explained_variances.append(explained_variance)

    # Print the current number of components and the corresponding explained variance ratio
    print(f"Number of components: {n_components}, Explained variance ratio: {explained_variance:.2f}")

# Plot the explained variance ratios for each number of components
plt.plot(n_components_range, explained_variances)
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio vs Number of Components')
plt.show()

# Find the optimal number of components based on the highest explained variance ratio
optimal_n_components = n_components_range[np.argmax(explained_variances)]
print(f"Optimal number of components: {optimal_n_components}")
```

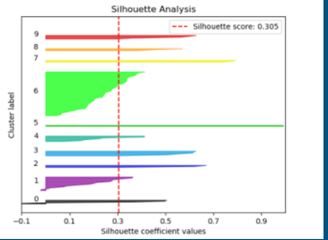



```
In [19]: svd = TruncatedSVD(n_components=10, random_state=42)
X_svd = svd.fit_transform(tfidf_matrix)
print(f"Total variance explained: {np.sum(svd.explained_variance_ratio_):.2f}")

Total variance explained: 0.14
```

```
In [20]: model= KMeans()
plot_elbow_curve(model,X_svd,cluster_ranges= range(1,30),figsize= (12,8));
```




```
In [40]: ## Kmeans = 10  stable and have a high score
model= KMeans(10)
model.fit(X_svd)
plot_silhouette(X_svd,model.labels_);
```



After using TF-IDF vectorizer, the transformation results in a large sparse matrix (in our case 4619 features) Clustering with high-dimensional data can be difficult, so dimension reduction is necessary.

The model must find a balance between Total Variance explained and a reasonable clustering number.

We decided to choose 10, it has the disadvantage of only explaining 14% of variance, but is a good selection on the elbow curve.

Also, it is a reasonable clustering number since our dataset contained 100 news-articles now we are trying to separate them into 10 topics.

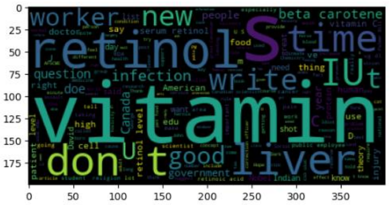
CODE BREAKDOWN

```
In [24]: ### Getting the labels of the clusters
model.labels_

Out[24]: array([ 3, 10,  4,  0,  1, 10, 11, 10,  7,  3,  6,  3, 10,  3,  3,  0,  3,
         0,  3, 10,  9,  3,  3,  3,  3,  8,  3,  0,  0, 10, 10,  8,  3,  0,
        10,  5, 10,  8, 10,  0,  0,  3,  4, 10,  0, 10,  1, 11,  9, 10, 10,
         4,  6,  9,  4,  2,  0,  3, 10, 10, 10, 10,  3,  0, 10,  7, 11,  1,
        10,  3, 10,  1,  1, 10,  6,  3,  0,  0,  4,  3,  3, 10,  3,  7, 10,
         3, 10,  9,  5, 10, 10, 10,  2,  3,  4, 10,  3,  5,  0, 10])

In [25]: ### Returning Lables to original dataframe
news['cluster']=pd.Series(model.labels_)

In [61]: news
Out[61]:
```

	aaron	ab	abbott	abi	ability	able	absolute	absolutely	absorption	abuse	...	zinn	zl	zodiacal	zoloft	zone	zoo	zoology	zv	zyue	cluster
0	0.0	0.0	0.0	0.0	0.0	0.053022	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	3
1	0.0	0.0	0.0	0.0	0.0	0.000000	0.164031	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	10
2	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	4
3	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0
4	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	10
96	0.0	0.0	0.0	0.0	0.0	0.183369	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.264062	0.0	0.0	0.0	0.0	0.0	3
97	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	5
98	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0
99	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	10

100 rows × 4620 columns

```
In [30]: cluster3_index= news[news['cluster'] == 3].index.tolist()
cluster3_wordcloud=[]
for i in cluster3_index:
    cluster3_wordcloud.append(preprocessed_docs[i])
cluster3_wordcloud_string = ' '.join(cluster3_wordcloud)

In [31]: wordcloud = WordCloud().generate(cluster3_wordcloud_string)

plt.imshow(wordcloud);
```

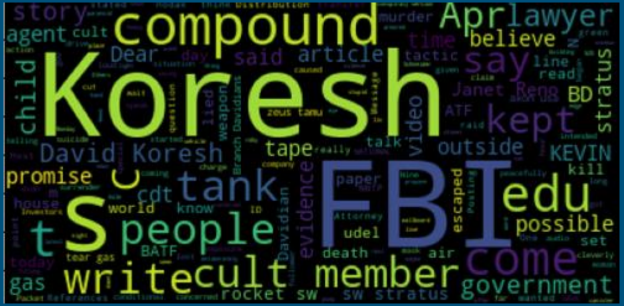


- On the left we are returning the labels to the original dataframe. Let's remember than each index on the dataframe relates to an article to a total of 100 articles.
- The code on the right loops over the index and gets the text of all the articles within the same cluster.
- Finally we create a wordcloud to see if the cluster has a topic or not.

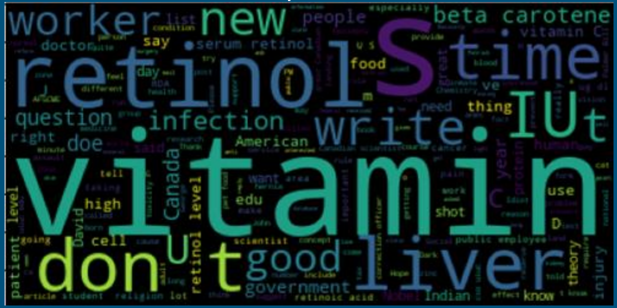
CODE BREAKDOWN

- 4 Examples of the 10 clusters.

Nasa - space



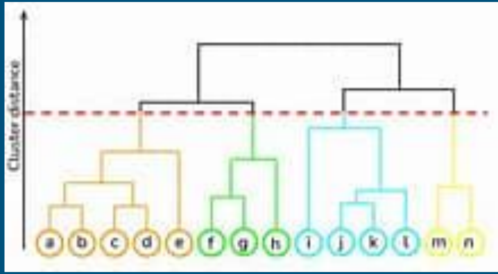
Crime. (David Koresh) Cult leader



Health

Sports.

CONCLUSION



- K-means requires a balance between the total variance explained due to dimension reduction and a reasonable number of clusters that align with the goal in this case the number of topics required.
- Other ML algorithms could be more successful in clustering text data. Hierarchical Clustering is also popular because it lets the user not only select the last cluster but it is possible to cut the dendrogram at a specific level, which will give you the clusters that are formed at that level.
- For this exercise we decided to to TFID with n_grams=1, but it is possible to use a range or a fixed bigger number to identify group of words that could have more meaning. It is also possible to use other vectorization techniques like Word Embeddings in particular Word2Vec and others.