COMP 4432 Group project report

Data Product- A million news headlines

Group members

Name	Student ID
Yin Chenxi	19089951D
Ikram Sattar	19107373D

Choosing between machine learning methods

Decision Tree, Random Forest, SVC, K-NN, Naive Bayesian were used for choosing the model

```
dt = DecisionTreeClassifier()
rf = RandomForestClassifier()
svc = SVC(kernel='linear')
knn = KNeighborsClassifier()
nb = MultinomialNB()

dt.fit(tfidf_train, y_train)
rf.fit(tfidf_train, y_train)
svc.fit(tfidf_train, y_train)
knn.fit(tfidf_train, y_train)
nb.fit(tfidf_train, y_train)
```

```
Testing Accuracy of Decision Tree: 84.94 %
Testing Accuracy of Random Forest: 86.05 %
Testing Accuracy of SVC: 85.42 %
Testing Accuracy of K-NN: 82.97 %
Testing Accuracy of Naive Bayesian: 85.71 %
```

KMeans method

Introduction to KNN

K-Nearest Neighbor (KNN, K-NearestNeighbor) classification algorithm is one of the simplest methods in data mining classification technology. The so-called K nearest neighbors means the K nearest neighbors, which means that each sample can be represented by its nearest K neighbors. In other words, the nearest neighbor algorithm is a method of classifying each record in a data set. This approach seems to work for this problem.

Originally proposed by Cover and Hart in 1968, it is a theoretically mature method and one of the simplest machine learning algorithms. The idea of this method is very simple and intuitive: if most of the K most similar samples in the feature space (that is, the closest neighbors in the feature space) belong to a certain category, the sample also belongs to this category. In the classification decision, this method only determines the category of the sample to be classified according to the category of the nearest one or several samples.

Analysis process

To be honest, the first method that came to my mind was MapReduce, which will be described in detail below. We have to find fake news and high-frequency words in the news.
The core idea of the KNN algorithm is that if most of the K nearest neighbors of a sample in the feature space belong to a certain category, the sample also belongs to this category and has the characteristics of the samples in this category. This method only determines the category of the sample to be classified according to the category of the nearest one or several samples in determining the classification decision. The KNN method is only

related to a very small number of adjacent samples when making class decisions. Since the KNN method mainly relies on the limited surrounding samples, rather than the method of discriminating the class domain to determine the class to which it belongs, the KNN method is more efficient than other methods for the set of samples to be divided that have more intersections or overlaps in the class domain. Suitable for.

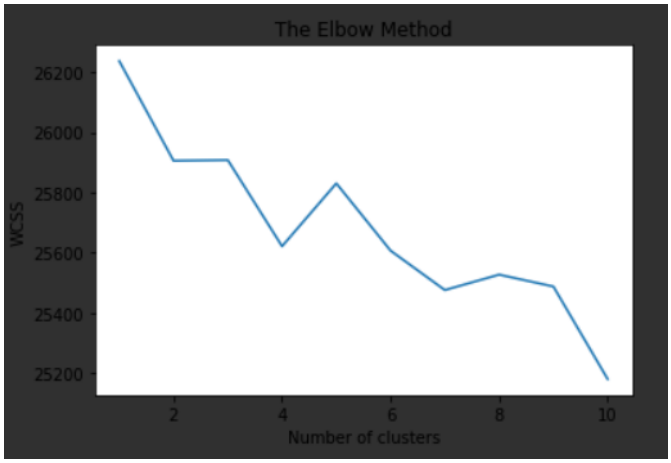
Implementation

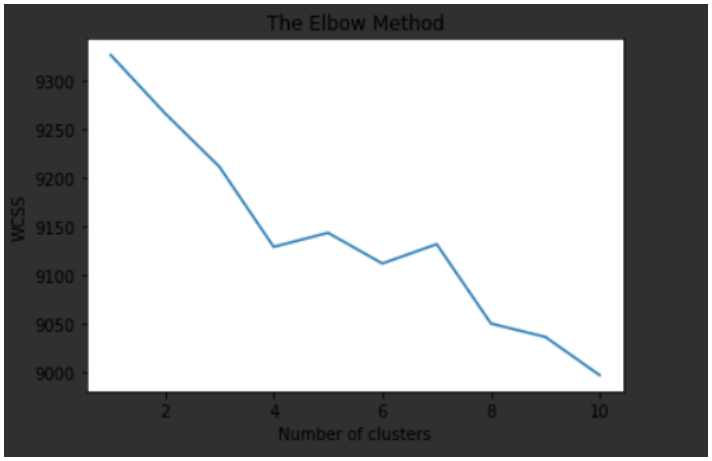
Stage 1: Simple KNN

I tried simple Kmeans first.
The dataset is too large, so I only tried 10% for the test.

According to the picture, I changed K for several times.
The result is not beautiful.
Here is an example:

kmeans = KMeans(n_clusters = 8, n_init = 20)





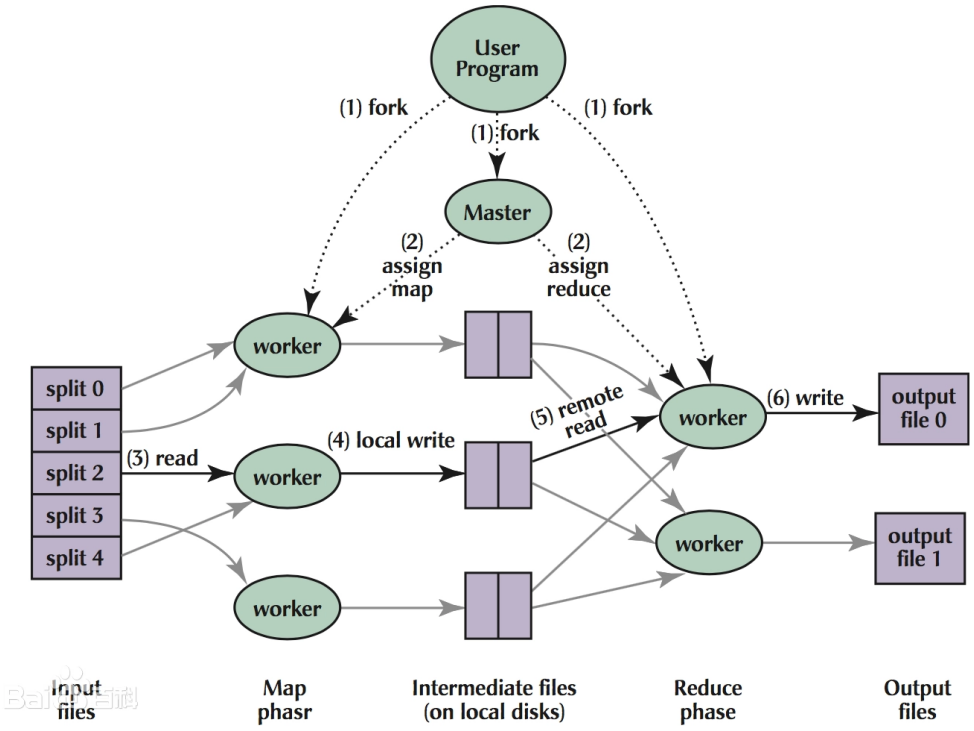
Also I tried to calculate the silhouette factor

```python
def evaluateScore(vectorized, labels):
    print("evaluateScore")
    Y = vectorized.toarray()
    calScore = metrics.calinski_harabasz_score(Y, labels)
    silScore = metrics.silhouette_score(Y, labels)
    return calScore, silScore
```

The result is only 0.07816831541512241 4.
It is too low!

Stage 2: Try MapReduce

Now I want to use Mapreduce method. MapReduce is a parallel computing model and method for large-scale data processing first proposed by Google.



What I want to do is to filter out words with certain characteristics and make a set, which is convenient for various subsequent work. The basic idea is as follows.

Map(String key, Record value) { // key: table name // value: table content for each row in value { if (row.gender = ‘Male’) emit(row.year, row.gpa); // intermediate key value pairs } }

```
Map(String key, Record value) {
    // key: News name
    // value: News content
    for each word in value {
        if (word.feature = ‘FakeNews’)
        emit(word.feature, word.beloging); // intermediate key value pairs
        }
}
```

```
Reduce(String key, Iterator values) {
    // key: News name
    // values: a list word
    fakeWordList = defaultdic{};
    for each word in values {
        fakeWordList.add(word);
    emit(key, values);
    }
}
```

The result is still bad
But we found some method that was more suitable for this problem. Word cloud and Bag-of-word

Stage 3: According to MapReduce, try TF-IDF
TF-IDF is a statistical method to assess the importance of a word to a document set or one of the documents in a corpus. The importance of a word increases proportionally to the number of times it appears in the document, but decreases inversely to the frequency it appears in the corpus. Various forms of TF-IDF weighting are often applied by search engines as a measure or rating of the degree of relevance between documents and user queries. In addition to TF-IDF, search engines on the Internet use link analysis-based ranking methods to determine the order in which documents appear in search results.

Implementation

```
tfidf = TfidfVectorizer()
tfidf_vector = tfidf.fit_transform(final_data["Clean_text"])

clusters = [2,3,4,5,6,7,8,9]
inertia = []
for i in tqdm(clusters):
    k_mean= KMeans(n_clusters=i,n_init=10)
    k_mean.fit(tfidf_vector)
    inertia.append(k_mean.inertia_)
```

I tried clusters
= [2,3,4,5,6,7,8,9]
= [2,3,4,5,6,7,8]
= [2,3,4,5,6,7]
= [2,3,4,5,6]
= [2,3,4,5]
= [2,3,4,]
= [2,3]

To find a better one.

Stage 4: clean the data first

Remove HTML tags and URL from the reviews

```
def html_tag(phrase):
    http_remove = re.sub(r"http\S+", "",phrase)
    html_remove = BeautifulSoup(http_remove, 'lxml').get_text()
    return html_remove
```

Remove the words with numbers and special character

https://stackoverflow.com/a/18082370/4084039

https://stackoverflow.com/a/5843547/4084039

```
def deleteWords(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

Removing the words from the stop words list

https://gist.github.com/sebleier/554280

```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

Stage 5: Add word cloud and bag of word

Word cloud:

The word cloud uses words as the basic unit to display the text more intuitively and artistically. The word cloud map, also called word cloud, is a visual display of the "keywords" that appear frequently in the text. The word cloud map filters out a large number of low-frequency and low-quality keywords. The text information, so that the viewer can appreciate the main idea of the text just by swiping the text.

Implementation:
https://github.com/amueller/word_cloud

```
def word_cloud(cluster_num):
    sentance = []
    num = cluster_num
    sent = final_data["Clean_text"][final_data["labels"]==num]
    for i in sent:
        sentance.append(i)
    sentance = ''.join(sentance)
    wordcloud = WordCloud(background_color="black").generate(sentance)
    print(f"Cluster Number: {num}")
    plt.figure(figsize=(12,9))
    plt.imshow(wordcloud)
    plt.axis("off")
    plt.show()
```

Bag of the world:

In information retrieval, the BOW model assumes that for a document, its word order, grammar, syntax and other elements are ignored, and it is only regarded as a collection of several words. The appearance of each word in the document is independent and does not depend on whether other words appear. (in no order)

```
final_data = data[0:100000]
bow = CountVectorizer(ngram_range=(1,2))
bow_vector = bow.fit_transform(final_data["Clean_text"])

clusters = [2,3,4,5,6,7,8,9]
inertia = []
for i in tqdm(clusters):
    k_mean= KMeans(n_clusters=i,n_init=10)
    k_mean.fit(bow_vector)
    inertia.append(k_mean.inertia_)
```

Stage 6: Using tools, data visualization

Two functions are created to see the result.

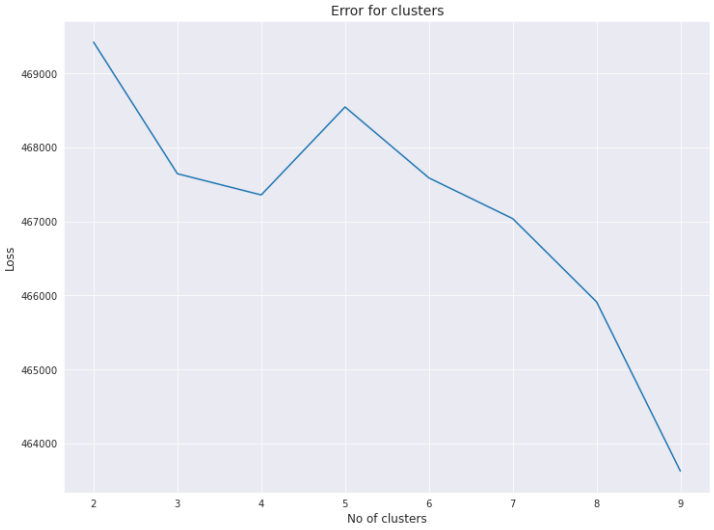
One is for clustering

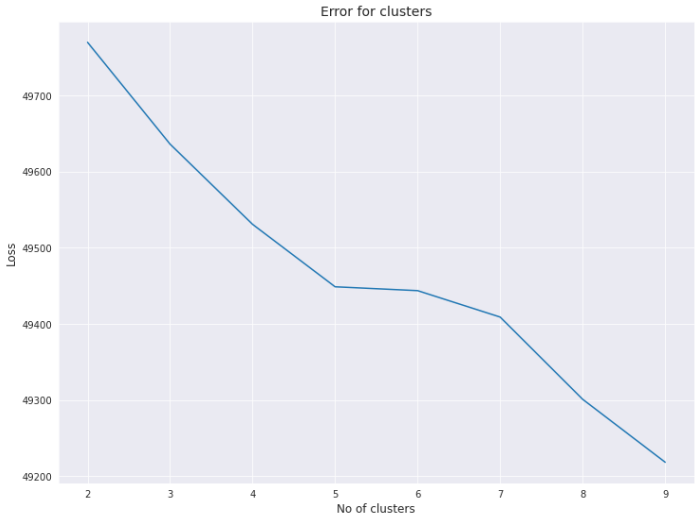
```
def NWordCloud(n_clusters):
    print("Number of clusters: {}".format(n_clusters))

    plt.figure(figsize=(15,15))

    rows = int((n_clusters/2)+1) if type(n_clusters/2)==float else n_clusters/2
    for i in range(0,n_clusters):
        sentance = []
        num = i
        sent = final_data["Clean_text"][final_data["labels"]==num]
        for j in sent:
            sentance.append(j)
        sentance = ''.join(sentance)
        wordcloud = WordCloud(background_color="black").generate(sentance)
        plt.subplot(rows, 2, i+1)
        plt.imshow(wordcloud)
    plt.axis("off")
    plt.show()
```

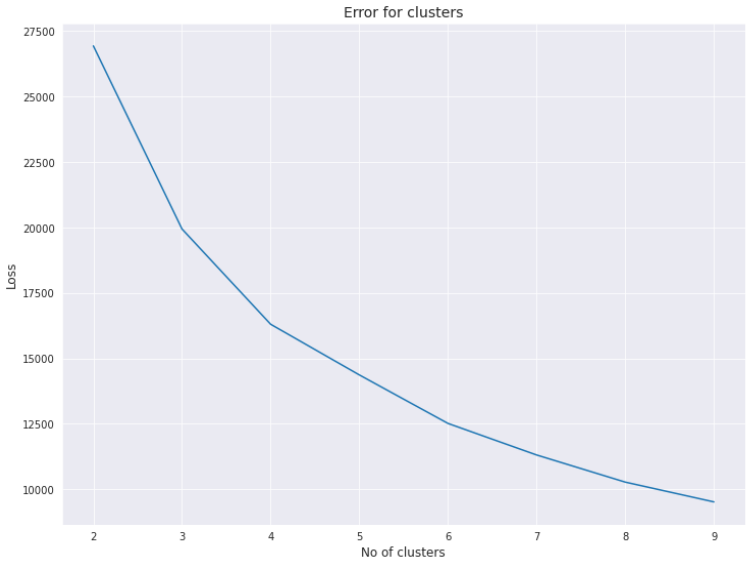
One is for the error

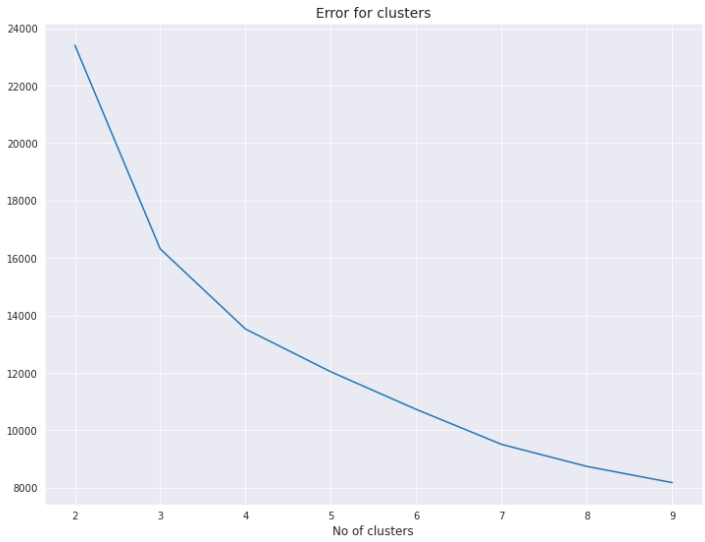
```
def printEroor(x,y,z):
    plt.figure(figsize=(x,z))
    sns.set_style(style="darkgrid")
    sns.lineplot(clusters,inertia)
    plt.xlabel("No of clusters",fontsize=x)
    plt.ylabel("Loss",fontsize=x)
    plt.title("Error for clusters",fontsize=y)
    plt.show()
```

Here are the results:
For 10% data:

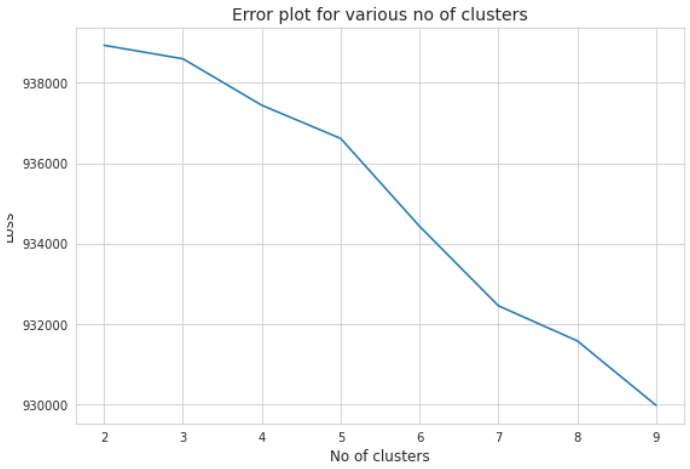


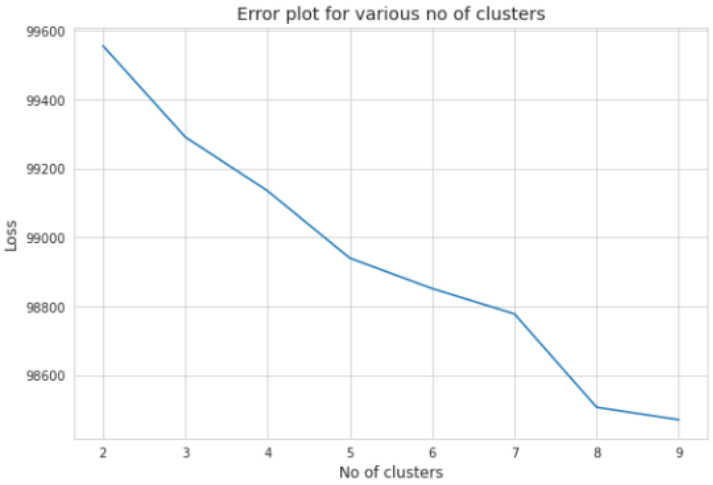


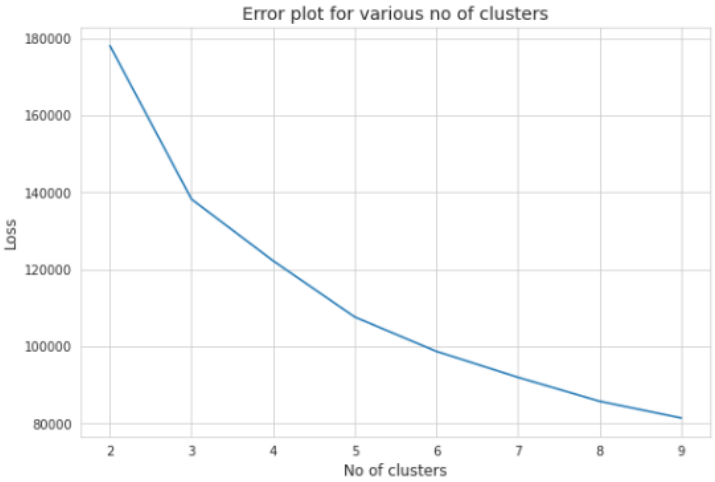


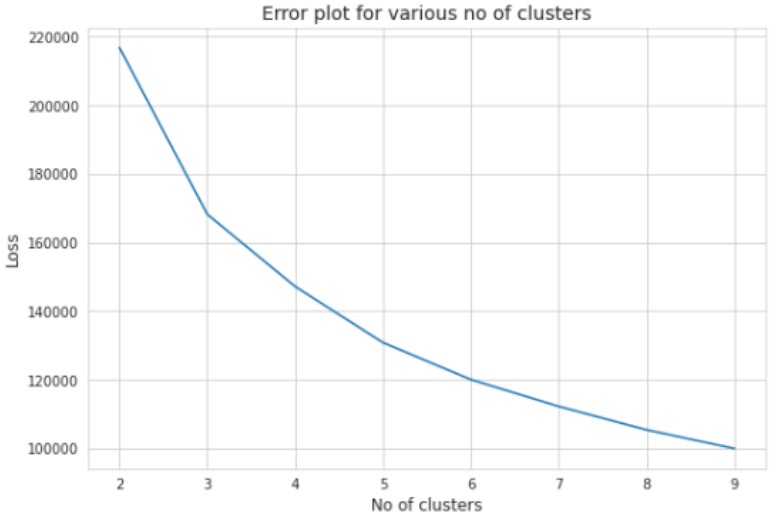


Here are the result:
For all the data:

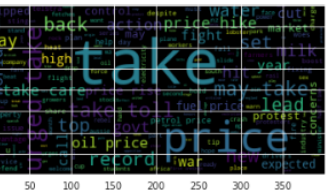


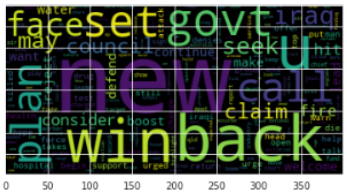


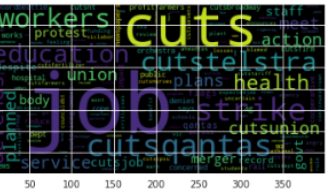




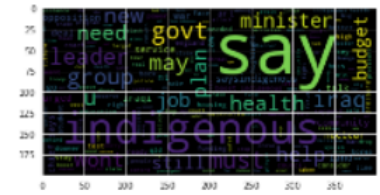
of clusters: 6



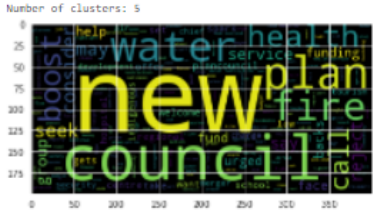




Number of clusters: 8



Number of clusters: 5



It can be seen that in the news, the words I like to use are "new, win, council, iraq, say, man , charged, wordcup, etc.".
Common words in the news are routine, so that they can attract people's attention and rank higher in search engines.

Such words include "new", "win", "charge", "set", "man", and "say". These words are generally common verbs and some nouns.
There are also some high-frequency words related to politics that can also attract attention, such as "council", "govt"
Some words related to the present will also have a higher degree of attention "iraq", "word cup".

Topic modeling using LSA and LDA

Topic modeling is an unsupervised document categorization method, similar to numeric data clustering, that finds natural groups of objects (topics) even when we don't know what we're looking for.

Latent Semantic Analysis (LSA) is a theory and method for using statistical calculations to extract and represent the contextual-usage meaning of words in a large corpus of text. The underlying premise is that the sum of knowledge about all the word contexts in which a given word appears and does not occur provides a set of mutual constraints that primarily defines how similar words and sets of words are in meaning. LSA's ability to represent human knowledge has been demonstrated in a variety of ways.

LDA (Latent Dirichlet Allocation) is a prominent statistical topic modeling technique. Documents in LDA are represented as a

collection of themes, each of which is a group of words. Those subjects are found in a hidden layer, also known as a latent layer.

The words within a document (a known factor) and the probability of words belonging to a topic (which needs to be estimated) make up LDA. The programme tries to figure out how many words in a document correspond to a specific topic. It also tries to figure out how many papers pertain to a certain topic based on a single word.
LDA, unlike LSA, does not output document similarities directly. Instead, LDA generates a z matrix with rows representing all of the words in the dataset and columns representing all of the documents. The LDA method assigns each value in the matrix to a topic to which the word represented by the row and column is assigned.

Model

After visualizing the data, we dropped the date column from the data so that only relevant data would be used in analysis. In the headlines, there were few irrelevant words found so we removed the words with less than 3 letters. This was done using libraries like “stopwords”. Such words were removed because they don’t aid in knowing the context of the topic. Then features were extracted and DTM (Document Term Matrix). A term document matrix is another way to represent text data. The text data is encoded as a matrix using this way. The columns of the matrix represent the words, and the rows of the matrix represent the sentences from the data that need to be examined. TF-IDF values were used to create the term documents. Various number of features were tried. 1000 features gave the best results, thus max_features in TF-IDF were set to 1000

```
vect =TfidfVectorizer(stop_words=stop_words,max_features=1000)
✓ 0.5s
```

After visualizing the Tfid vector values, it was found out that “police” was one of most common occurring words. TheThe lesser the value; more common is the word in the news headlines.

After the initial fine-tuning, we then went ahead with the actual implementation of LSA and LDA

We used SVD in our LSA model building. The original DTM is decomposed into three matrices via SVD (sigma). (V.T). The document-topic matrix is denoted by U, and the topic-term matrix is denoted by V.

The vector representation of the associated document is represented in each row of the matrix U(document-term matrix). The number of desirable subjects is represented by the length of these vectors. The matrix V contains a vector representation of the terms in our data (term-topic matrix).

As a result, SVD provides vectors for each document and term in our dataset. Each vector would have a length of k. The cosine similarity method can then be used to locate comparable words and documents using these vectors.

After fitting the LSA model on headlines, the topics were visualized and most frequent words in every topic were extracted.

Similarly the LDA model was designed. Model was fitted on the data by setting the number of topics as 10. The values given by the model in a particular row add to 1. This is because each value represents the percentage contribution of the document's corresponding topic. As every document consists of topics, we visualized some documents to explore most common topics. Then digging-in deeper, we extracted the most important words in each topic. For better visualization we used word cloud to see important words of each topic.

Important words of topic#2

References:

https://www.cnblogs.com/polly-ling/p/10429624.html#:~:text=word cloud%E5%BA%93%EF%BC%8C%E5%8F%AF%E4%BB%A5%E8%AF%B4%E6%98%AF,%E5%8F%AF%E9%A2%86%E7%95%A5%E6%96%87%E6%9C%AC%E7%9A%84%E4%B8%BB%E6%97%A8%E3%80%82

https://baike.baidu.com/item/tf-idf/8816134

https://zhuanlan.zhihu.com/p/29933242

https://www.kaggle.com/code/jonathanlam1014/fake-news-headlines-classification

O'Neill, D., & Harcup, T. (2019). News values and news selection. In *The handbook of journalism studies* (pp. 213-228). Routledge.

Gold, D., & Simmons, J. L. (1965). News selection patterns among Iowa dailies. *The Public Opinion Quarterly*, *29*(3), 425-430.