

**School of InfoComm Technology**

**Applied Analytics Assignment**

Diploma in Cybersecurity & Digital Forensics

Diploma in Financial Informatics

Diploma in Information Technology

Year 2/3 (20212/2023), Semester 3/5

**TEAM/INDIVIDUAL ASSIGNMENT**

(40% of AA Module)

**Deadline for Submission:**

**Presentation: 31th July 2022 (Sunday),23:59hrs**

**Report & Code: 14th August 2022 (Sunday),23:59hrs**

|  |  |  |  |
| --- | --- | --- | --- |
| Tutorial Group | : | P01 | |
| Team Number | : | Grp 3 | |
| Tutor | : | Toh Ser Chye | |
| Members | : | Student No. | Student Name |
|  |  | S10219150 | Leo Yun Tao |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Penalty for late submission:**

10% of the marks will be deducted every day after the deadline.

**NO** submission will be accepted after 21st August 2022, 23:59 hrs.

Table of Contents

[0. Introduction 3](#_Toc111382512)

[1. Text Data Preprocessing 5](#_Toc111382513)

[1.1 Importing libaries 5](#_Toc111382514)

[1.2 Loading Data 5](#_Toc111382515)

[1.2 Cleanse the Text Data 7](#_Toc111382516)

[1.3 Bag-of-Word 8](#_Toc111382517)

[1.4 TF-IDF 9](#_Toc111382518)

[2. Text Data Understanding 11](#_Toc111382519)

[2.1 Extract Keywords using TF-IDF matrix 11](#_Toc111382520)

[2.2 Association Rules Mining on Keywords 13](#_Toc111382521)

[3. Summary & Improvement 20](#_Toc111382522)

[3.1 Summary 20](#_Toc111382523)

[3.2 Possible further improvements 22](#_Toc111382524)

[References 23](#_Toc111382525)

# Introduction

The dataset (bbc-text.csv) contains 2 attributes, called text and category. This dataset contains 2225 documents from the BBC news website corresponding to stories in five topical areas from 2004-2005. Each document belongs to one of five categories: Business, Entertainment, Politics, Sports and Tech. More information can be found here: ML Resources - BBC Datasets (ucd.ie)

The purpose, and henceforth my approach to this problem is to perform text analytics on the BBC dataset to find meaningful keywords in the corpus using text analysis techniques. In this case, we will be using association rule mining for this problem. In order to better understand this report, you need to have a basic understanding of Python, association rules mining concepts, such as support, lift and confidence and a bit of natural language processing techniques such as stemming and lemming.

Association rule mining is one of the techniques in unsupervised machine learning to find an interesting association between the itemsets. Using the Apriori algorithm, one of the algorithms used in association rule mining, we will find which keywords are highly related to another keyword for us to interpret and provide an explanation as to what causes these keywords to be related to each other. Association rule mining is commonly used by retailers to increase sales by understanding customer purchasing patterns where we use customer to purchase items history to find out which products are likely to be purchased together. This is known as market basket analysis. However, in this assignment, we are going to find out the interesting keywords that are grouped together in the BBC news dataset.

Similar to the previous assignment, I will show the process step by step using python to find useful insights into the BBC news data. I will also copy and paste the code inside instead of taking a screenshot so that you can copy the section's code to try it out for yourself. I will also justify my reasons behind implementing the code so that you can understand my thoughts and rationale behind the decision that I made.

This report will set as a stepping stone for me to better understand what is going on in the data. In this context, it will be a better understanding of what is going on between the years 2004-2005 before moving on to building a classification model to predict the category of a document in the next problem. I will also try to explain the reason why these itemsets are related to each other by searching up the events using the keywords generated within the time frame 2004-2005.

As a result of the apriori algorithm, we will find out a list of rules generated with the support, confidence and lift values. Support is a probability of an itemset occurrence in the document. Confidence is the probability of the consequent occurring if the antecedent occurs. Lift is the ratio between the confidence and the expected confidence of the antecedent and consequent. All of these values are essential for us to integrate the rules generated.

Using the generated rules, I have derived some of the key terms that are trending in the news in each category between 2004-2005. I have also found the keywords that will be frequently grouped together in the document. For example, if the antecedent is “google”, there will be a high chance of the consequent “search” occurring in the document. Furthermore, I have split the keywords into 2 different types of groups, event-specific keywords and general keywords. Popular keywords are words that are popular from 2004-2005 and general keywords are keywords that are naturally similar to each other. I will explain this further in the summary section.

Overall, this report will showcase how I load and explore the BBC text data. Followed by cleaning the data using several natural language processing techniques such as Lemming, removing stopwords etc. This is done to reduce unimportant words and to convert the data into a format a computer can understand so that we can use it for association rule mining. Lastly, I will summarize all my findings and interpret the computer-generated findings to give insight into the patterns found in the data. I will also provide suggestions of what I could have improved during this assignment so that I will work on those suggestions if I need to apply text analytics in the future.

# Text Data Preprocessing

This section will showcase how I process the BBC text data

## 1.1 Importing libaries

I imported all the libaries needed for applying association rule mining.

import pandas as pd

import numpy as np

import nltk

import string

import re

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import cross\_val\_score, GridSearchCV, train\_test\_split

from sklearn.linear\_model import LogisticRegression, SGDClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_recall\_fscore\_support, confusion\_matrix, plot\_confusion\_matrix

import matplotlib.pyplot as plt

import seaborn as sn

## 1.2 Loading Data

Loaded the CSV file into data frame format and displayed 5 results to understand the data loaded.

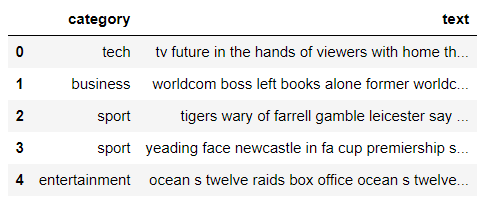
pd.read\_csv() will read a comma-seperated values (csv) into DataFrame format.

df.head() will display the first 5 results if the n value is not set.

The bbc-text.csv is given to us for this assignment to download.

df = pd.read\_csv("bbc-text.csv")

df.head()



Use df.info() to find if there’s any null value in the data, if there is any null value, we would have to clean the null value.

df.info()

Turns out there are no null values in the dataset which means the dataset is quite clean. The data frame contains 2 text columns.

Text, letter

Description automatically generated

Finding out how many documents are there in each category by counting the number of categories and plot a bar chart to visualize it.

df.category.value\_counts().plot.bar()

As you can see the distributions of the documents in each category are slightly unbalanced. This is fine in association rule mining, so no changes are needed to the dataset.

Icon

Description automatically generated

## 1.2 Cleanse the Text Data

In this section, we are going to clean the dataset so that we can prepare for the apriori algorithm, and more importantly, the classification model in the next problem.

I created 3 different methods to clean the text data. The first method get\_stop\_words(stop\_file\_path) is used to load all the stopwords in stopwords.txt given to use. It removes duplicate words by converting to a set datatype. Later on, we are going to remove stopwords from the text because stopwords serve no purpose in the text.

A set datatype is a Data Structure in Python with an unordered and unindexed collection of elements. Every element in a Set is always unique. The Set Data Structure does not allow any duplication of elements (Bharath K, 2021).

This function returns the value in a frozen set format. frozenset is a fixed version of set. In other words, you cannot add or remove items to the frozen set.

def get\_stop\_words(stop\_file\_path):

    """load stop words """

    with open(stop\_file\_path, 'r', encoding="utf-8") as f:

        stopwords = f.readlines()

        stop\_set = set(m.strip() for m in stopwords)

        return frozenset(stop\_set)

#load a set of stop words

stopwords=get\_stop\_words("stopwords.txt")

The second method preprocessor(word) is used to replace any characters that are not alphanumeric with a single space using regex.

A regex also known as regular expression is a sequence of characters that specifies a search pattern in the text. Usually, such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. Regular expression techniques are developed in theoretical computer science and formal language theory (Wikipedia Contributors, 2022).

def preprocessor(word):

    return re.sub("(\\d|\\W|\_)+"," ",word)

The last method LemmaTokenizer(text) is used to Lemmatize, tokenize and remove the stop words. Lemmatization is a form of text normalization where we try to group the words with the same meaning to a common root word. For example “running” will become “run” when we use lemmatization. This helps to reduce the number of words the computer needs to process, thus improving the accuracy of the model.

I chose to use Lemmatization instead of stemming because it uses vocabulary and morphological analysis of words, usually aiming to remove inflectional endings only and return the base or dictionary form of a word. Whereas stemming is a crude heuristic process that chops off the ends of words and frequently includes the removal of derivational affixes (Stemming and Lemmatization, 2022).

Removing stopwords is especially important when building a classification model later on in problem 2 to improve its performance.

Lastly, tokenizing helps to split a text into a list to be used for further analysis. Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either word, characters, or subwords (What Is Tokenization | Tokenization in NLP, 2020).

def LemmaTokenizer(text):

    tokens = re.split('\\W+', text)

    return [wn.lemmatize(word) for word in tokens if word not in stopwords]

Before processing the data by extracting the first 100 characters in the first text

df.text[0][:100]

'tv future in the hands of viewers with home theatre systems plasma high-definition tvs and digital'

After processing the data

[i for i in LemmaTokenizer(preprocessor(df.text[0][:100]))]

['tv', 'future', 'hand', 'viewer', 'home', 'theatre', 'system', 'plasma', 'definition', 'tv', 'digital']

Now the data looks cleaner and we can begin to use bag-of-words and tfidf on the preprocessed data.

## 1.3 Bag-of-Word

In this section, we are going to convert our preprocessed data into a bag-of-word format, so that we can convert it into a TF-IDF matrix later on.

Bag of words is a Natural Language Processing technique of text modelling. In technical terms, we can say that it is a method of feature extraction with text data. This approach is a simple and flexible way of extracting features from documents (Great Learning Team, 2020).

A bag of words is a representation of text that describes the occurrence of words within a document. We just keep track of word counts and disregard the grammatical details and the word order. It is called a “bag” of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document (Great Learning Team, 2020).

Used max\_df=0.15 to remove terms/words that come out in > 15% of the corpus to remove any corpus-specific stopwords. Used a customer preprocessor and tokenizer created in the last section as it is better than the default preprocessor and tokenizer. Also used max\_features=5000 to keep 5000 terms to speed up computing time.

Lastly, I used fit\_transform to convert df.text into bag-of-word format.

count\_vect = CountVectorizer(max\_df=0.15, max\_features=5000, lowercase=True,

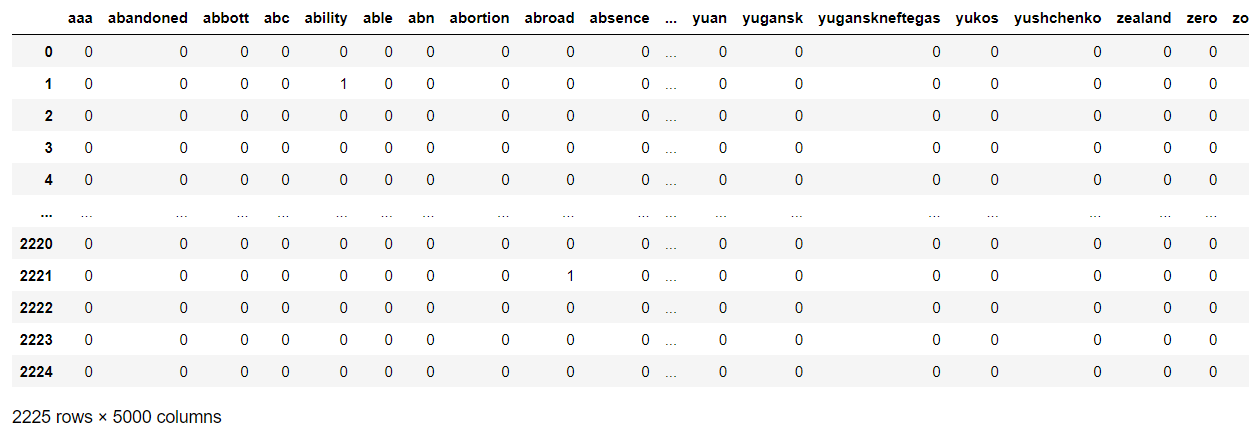
                             preprocessor=preprocessor, tokenizer=LemmaTokenizer)

X\_counts = count\_vect.fit\_transform(df.text)

Visualizing bag-of-words:

pd.DataFrame(X\_counts.toarray(),

columns=count\_vect.get\_feature\_names\_out())



## 1.4 TF-IDF

Calculating frequency-inverse document frequency score for each word in tokens. TF-IDF is an algorithm that is used to represent how important a word is in a document. TF-IDF score is between 0 – 1. The higher the score is, the more important the word is in a document.

TF-IDF for a word in a document is calculated by multiplying two different metrics:

* The term frequency of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are ways to adjust the frequency, by the length of a document, or by the raw frequency of the most frequent word in a document.
* The inverse document frequency of the word across a set of documents. This means, how common or rare a word is in the entire document set. The closer it is to 0, the more common a word is. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm.

(Understanding TF-ID: A Simple Introduction, 2019)

Now we instantiate the TfidfTransformer class and used fit\_transform from the bag of words we have created previously to convert it into a TF-IDF score format.

tfidf\_transformer=TfidfTransformer()

X\_tfidf = tfidf\_transformer.fit\_transform(X\_counts)

Visualizing TF-IDF score:

pd.DataFrame(X\_tfidf.toarray(),

columns=count\_vect.get\_feature\_names\_out())

Table

Description automatically generated with medium confidence

With the TF-IDF score, we are ready to extract the keywords in the next section to prepare for the apriori algorithm.

# Text Data Understanding

This section will showcase how I extract the keywords using association rule mining.

## 2.1 Extract Keywords using TF-IDF matrix

This code will sort the TF-IDF score from the lowest to highest value and will take out the top 500 terms with the highest TF-IDF score. The reason why we do this is that the apriori algorithm is very inefficient with a runtime complexity of O(2^n). By reducing the terms to the top 500, we can reduce the computing time needed to run the apriori algorithm.

# find maximum value for each of the features over dataset:

max\_value = X\_tfidf.max(axis=0).toarray().ravel()

sorted\_by\_tfidf = max\_value.argsort()

highestTfidf = np.array(count\_vect.get\_feature\_names\_out())[sorted\_by\_tfidf[-500:]]

highestTfidf

The function extract\_keywords(paragraph) will extract the top 500 tfidf keywords found in the paragraph and return in a list format so that we can encode it using TransactionEncoder later on.

def extract\_keywords(paragraph):

    paragraph = LemmaTokenizer(preprocessor(paragraph.lower()))

    keywords = [word for word in paragraph if word in highestTfidf and word not in stopwords]

    return keywords

Applying the function:

cleaned\_array = df.text.apply(lambda x: extract\_keywords(x))

cleaned\_array

Output:

Text, letter

Description automatically generated

Code to Visualize the top 50 most frequent keywords to get a sense of which words are popular.

full\_list = []

for row in cleaned\_array:

    for value in row:

        full\_list.append(value)

full\_list = pd.Series(full\_list)

# looking at the frequency of most popular items

plt.figure(figsize=(18,7))

full\_list.value\_counts().head(50).plot.bar()

plt.title('frequency of most popular items', fontsize = 20)

plt.xticks(rotation = 90 )

plt.grid()

plt.show()

As you can see film is the most frequent keyword, followed by bn, music …

Chart, bar chart, histogram

Description automatically generated

Next using the list format that we have generated; we will convert the list into a one-hot boolean array format to be used for Apriori Algorithm. This is similar to one hot encoding method in scikit learn. In other words, TransactionEncoder encodes database transaction data in form of a Python list of lists into a NumPy array (Raschka, 2014).

te = TransactionEncoder()

data\_encoded = te.fit\_transform(cleaned\_array)

data\_encoded = pd.DataFrame(data\_encoded, columns = te.columns\_)

data\_encoded

Output:

Table

Description automatically generated with medium confidence

## 2.2 Association Rules Mining on Keywords

We will use the encoded data to generate the support value of each itemsets using the apriori algorithm.

Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects. It means how two or more objects are related to one another (Apriori Algorithm - Javatpoint, 2021).

The primary objective of the apriori algorithm is to create the association rule between different objects. The association rule describes how two or more objects are related to one another. Apriori algorithm is also called frequent pattern mining. Generally, you operate the Apriori algorithm on a database that consists of a huge number of transactions (*Apriori Algorithm - Javatpoint*, 2021).

I set the min\_support parameter = 0.01 to remove itemset that appears in < 1% of the corpus to speed up computation, this is also known as the Apriori principle where if an itemset is frequent, all its supersets will also be frequent, so there is no point in finding infrequent itemsets as the support value will be low also. The apriori principle is similar to an algorithm known as Alpha-beta pruning I also sorted the support value by descending order and display the top 10 highest support values.

Support value is calculated by the fraction of the total number of transactions in which the item set occurs. The higher the support value means that an itemset appears more frequently in the document (Garg, 2018).

frequent\_itemsets=apriori(data\_encoded, min\_support = 0.01, use\_colnames = True)

frequent\_itemsets.sort\_values("support", ascending = False, inplace=True)

frequent\_itemsets.head(10)

As you can see, (bn) occurs in the most corpus followed by (sale), (business) …

What is suprising is that film does not have the highest support value. This could be because film may have appeared in one document many times but not throughout all the documents.

Table

Description automatically generated

Used pd.describe() to find out how the statistics of the support value.

As the support value is numeric, pd.describe will return a result index that will include count, mean, std, min, and max as well as lower, 50 and upper percentiles. By default, the lower percentile is 25 and the upper percentile is 75. The 50 percentile is the same as the median (Pandas.DataFrame.describe — Pandas 1.4.3 Documentation, 2022).

frequent\_itemsets[['support']].describe()

From the statistics generated, 451 itemsets has >= 1% support value.

Table

Description automatically generated

In this block of code, the association\_rules() function will form an association rule for all itemsets.

An association rule is an implication expression of the form X→Y, where X and Y are disjoint itemsets. A more concrete example based on consumer behaviour would be {Diapers}→{Beer} suggesting that people who buy diapers are also likely to buy beer. To evaluate the "interest" of such an association rule, different metrics have been developed. Our implementation makes use of the confidence and lifts metrics (Raschka, 2014).

It will remove all itemsets with < 1.5 lift and <= 0.5 confidence so that only those interesting rules will be left. Furthermore, I added a new column “antecedent\_lent” to find the length of itemsets in the antecedent for further exploration. I will also sort and display the top 15 lift values to find the interesting rules.

Confidence defines the likeliness of occurrence of consequent on the cart given that the cart already has the antecedents, whereas, Lift controls for the support (frequency) of consequent while calculating the conditional probability of occurrence of {Y} given {X} (Garg, 2018).

Typically a good lift and confidence value are > 1.5 and > 0.5 respectively.

rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1.5)

rules.sort\_values("lift", ascending=False, inplace=True)

rules = rules[rules.confidence > 0.5]

rules["antecedent\_len"] = rules["antecedents"].apply(lambda x: len(x))

rules.head(15)

As you can see from the output, (yukos) -> (bn, tax, oil) has the highest lift value of 55.7 this means that you are more 55.7 times more likely to see (yukos, bn, tax, oil) together than with (yukos) alone.

From this result, different combinations of yukos, bn, tax and oil yield a really high lift value.

Also, (yukos) -> (bn, tax, oil) have a confidence value of ~0.85, this means that if the word “yukos” apperas in the document, bn, tax and oil will have an 85% chance of apperaing in the document too.

Table

Description automatically generated

Plotting boxplot for the lift values

A boxplot is a standardized way of displaying the distribution of data based on a five-number summary (“minimum”, first quartile [Q1], median, third quartile [Q3] and “maximum”) (Understanding Boxplots, 2022).

rules[['lift']].boxplot()

From the boxplot, most lift values fall between 6-12 and there are some outliers with very high lift values.

Chart, box and whisker chart

Description automatically generated

Filtering out the result by antecedent length to find more interesting rules.

rules[rules.antecedent\_len == 1].head(10)

Table

Description automatically generated

rules[rules.antecedent\_len == 2].head(10)

Table

Description automatically generated

rules[rules.antecedent\_len == 3].head(10)

Table

Description automatically generated

Filtering out the results by common itemsets found previously

rules[rules['antecedents'] == {'yukos'}]

Table

Description automatically generated with low confidence

rules[rules['consequents'] == {'film'}]

Graphical user interface

Description automatically generated with medium confidence

rules[rules['consequents'] == {'election'}].head(10)

Table

Description automatically generated

rules[rules['consequents'] == {'mobile'}].head(10)

Table

Description automatically generated

Finding the categories of the popular itemsets using regex. These code will find all the documents that contains a substring and group them into different categories.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

After gathering all these information, we will move on to the summary and improvement section.

# Summary & Improvement

## 3.1 Summary

Based on the previous section, I have found and summarized some of the keywords that are highly related to each other and have grouped them based on the category. In a document, you will have a high chance of seeing these keywords together in a document. I have also split the keywords into 2 different types of groups called event-specific keywords and popular keywords.

Since this dataset belongs to BBC news from 2004-2005, I have searched up some of the keywords together to get a clue of what is happening between 2004-2005 that causes these keywords to be highly related to each other.

**Popular Keywords**

Popular keywords are keywords that appear due to a famous situation occurring between 2004-2005. These keywords normally do not have a high lift value if the event does not exist. Because of these events happening, the keywords have now been associated with each other more freqently.

|  |  |
| --- | --- |
| **Keywords** | **Category** |
| yukos, oil, bn, tax | Business |
| party, blair, brown, election | Politics |

**Yukos, oil, bn, tax**

Yukos is a Russian oil company owned by a Russian oligarch Mikhail Khodorkovsky, believed to be the richest person in Russia got sentenced to nine years in jail for fraud and tax evasion between 2004-2005. Since his arrest, his wealth has shrunk from $15.2bn to $2.2bn (Nick Paton Walsh, 2005).

This explains why between 2004-2005, Yukos, oil, bn (short form for billions) and tax are closely related to each other due to this news.

**Party, blair, brown, election**

From 2004-2005, Prime Minister Tony Blair's supports for war in Iraq lead to a heavy defeat in local elections. Furthermore, more than 460 Labour party members were voted out of local government. Because of this, more Labour lawmakers fear for their own jobs and wanted Gordon Brown to replace Tony Blair to fight for the election (NBC Universal, 2004).

This explains why between 2004-2005, Party, Blair (Tony Blair), Brown (Gordon Brown) and election keywords are closely related to each other due to the election and Blair’s support for the war in Iraq.

**General Keywords**

General keywords are the keywords you can commonly find relating to the category for the news. These keywords are related to each other because they are similar to each other by nature. You can find these keywords together in the most time frame, not limited to 2004-2005.

|  |  |
| --- | --- |
| **Keywords** | **Category** |
| google, search | Tech |
| phone, gadget, camera, mobile, call, text | Tech |
| currency, dollar | Business |
| film, theatre, festival, studio, dvd, award | Entertainment |
| arsenal, chelsea | Sport |
| zealand, rugby | Sport |

**Google, search**

Google search is a common and widely used term. Therefore, it is not surprising to see these keywords together.

**Phone, gadget, camera, mobile, call, text**

Same as google search, these keywords often come together in a tech news article.

**Currency, dollar**

Once again another common keyword often used together in a business article. These keywords are often found in articles featuring different exchange rates between different currencies.

**Film, theatre, festival, studio, dvd, award**

These keywords are related to several news articles relating to a film festival award. A film festival award is an organized event at which many films are shown. They also give many awards in different categories.

**Zealand, rugby**

New Zealand is a country that is popular for its rugby. Because of that, you can see a lot of articles with the keywords New Zealand and rugby together.

**arsenal, chelsea**

There was a lot of match build up between a football match between Arsenal and Chelsea on 12 December 2004, which means that many news will have to write an article about Arsenal vs Chelsea which makes these keywords to grouped together frequently. Both keywords belong to a team in soccer.

## 3.2 Possible further improvements

One possible improvement for this project is to use leverage and conviction to understand the association rules more and find more interesting rules. Leverage can be used to find the difference between the observed frequency between 2 itemsets and the expected frequency if the 2 itemsets are independent. Conviction can be used to find how dependent is the consequent on the antecedent (Raschka, 2014).

Another possible improvement that I can make is tweaking the ngram\_range in the CountVectorizer.

For example, setting the ngram\_range=(1,1) would give us unigrams or 1-grams such as “google” and “search”, while (2,2) would give us bigrams or 2-grams, such as “google search” (Matt Clarke, 2021). This will probably help in identifying keywords that are together to find a more interesting rule in association rule mining.

# References

Nick Paton Walsh. (2005, June). *Russian oligarch jailed for nine years*. The Guardian; The Guardian. https://www.theguardian.com/world/2005/jun/01/business.oil

‌ NBC Universal. (2004, June 11). *Blair faces party anguish*. NBC News; NBC News. <https://www.nbcnews.com/id/wbna5188843>

Raschka, S. (2014). *Association rules - mlxtend*. Github.io. <http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/>

*Stemming and lemmatization*. (2022). Stanford.edu. <https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>

*What is Tokenization | Tokenization In NLP*. (2020, May 25). Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/>

Bharath K. (2021, January 16). *Mastering Dictionaries And Sets In Python! - Towards Data Science*. Medium; Towards Data Science. <https://towardsdatascience.com/mastering-dictionaries-and-sets-in-python-6e30b0e2011f#:~:text=Set%20is%20a%20Data%20Structure,when%20it%20comes%20to%20mutability>.

Great Learning Team. (2020, September 11). *An Introduction to Bag of Words (BoW) | What is Bag of Words?* GreatLearning Blog: Free Resources What Matters to Shape Your Career! <https://www.mygreatlearning.com/blog/bag-of-words/#sh1>

*Understanding TF-ID: A Simple Introduction*. (2019, May 10). MonkeyLearn Blog. <https://monkeylearn.com/blog/what-is-tf-idf/#:~:text=TF%2DIDF%20(term%20frequency%2D,across%20a%20set%20of%20documents>.

*Matt Clarke*. (2021, December 24). Practicaldatascience.co.uk. <https://practicaldatascience.co.uk/machine-learning/how-to-use-count-vectorization-for-n-gram-analysis#:~:text=CountVectorizer%20will%20tokenize%20the%20data,such%20as%20%E2%80%9Cwhey%20protein%E2%80%9D>.

Wikipedia Contributors. (2022, August 2). *Regular expression*. Wikipedia; Wikimedia Foundation. <https://en.wikipedia.org/wiki/Regular_expression>

Raschka, S. (2014). *TransactionEncoder - mlxtend*. Github.io. <http://rasbt.github.io/mlxtend/user_guide/preprocessing/TransactionEncoder/>

*Apriori Algorithm - Javatpoint*. (2021). Www.javatpoint.com. <https://www.javatpoint.com/apriori-algorithm>

Raschka, S. (2014). *Association rules - mlxtend*. Github.io. <http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/#example-2-rule-generation-and-selection-criteria>

*Understanding Boxplots*. (2022). Built In. <https://builtin.com/data-science/boxplot>

*pandas.DataFrame.describe — pandas 1.4.3 documentation*. (2022). Pydata.org. <https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html>

Garg, A. (2018, September 3). *Complete guide to Association Rules (1/2) - Towards Data Science*. Medium; Towards Data Science. <https://towardsdatascience.com/association-rules-2-aa9a77241654>