Table of Contents

Bianco Blanco NJITID: 31620161
Email: bmb45@njit.edu

Summary

For this final project I choose option 4 for the project in which we had to open the Reuters-21578 files, read the .sgm files provided in the .tar.gz file, extract the texts inside each .sgm file retaining where the texts originate from, using the standfordnlp to lemmatize and use stop words to better refine the keywords, taking the top 200 keywords and representing each extracted document from the .sgm file as a subset of those top 200 keywords if the keyword occurs in the document, and the finally applying the Apriori Algorithm to generate frequent item-sets and association rules. I imported standfordnlp and reused an Apriori Algorithm I have previously coded, however when it came time to do the td-idf vectorization process I coded that from scratch. I imported helpful packages to gauge how look the lemmatization process or the generate candidates process will take, since we can do text mining process on many documents. I also allowed user inputs for things such as confidence value, support value, random sampling number, and folder/file path. I applied this text mining algorithm to run on two other datasets, please see screenshots.

Text Mining Algorithm

Description – Text Mining Algorithm Python code used to extract text and generate association rules.

```
import argparse
import math
import os
import random
import re
import sys
import tarfile
import time
from tqdm import tqdm
from collections import defaultdict

import nltk
import pandas as pd
import stanza
from nltk.corpus import stopwords


def parse_arguments(argv):
    """
    Parses command-line arguments.
    """
    parser = argparse.ArgumentParser(description="Text Mining Algorithm "
                                                 "Using Apriori Algorithm and
StandfordNLP")

    parser.add_argument('-s', '--min-support', metavar='float', type=float,
default=0.1,
                        help='Minimum support ratio (must be > 0, default:
0.1).')
    parser.add_argument('-c', '--min-confidence', metavar='float', type=float,
default=0.5,
                        help='Minimum confidence (default: 0.5).')
    parser.add_argument('-k', '--sample-size', metavar='int', type=int,
default=385,
                        help='Sample size (default: 385).')
    parser.add_argument('-f', '--file', metavar='path', required=True,
                        help='Path to the dataset file.')

    return parser.parse_args(argv)


def extract_texts_from_directory(directory_path):
    """
    Extracts text files from a given directory.
    """
    texts = {}
    for root, _, files in os.walk(directory_path):
        for file in files:
            if file.endswith('.txt'):
                file_path = os.path.join(root, file)
                try:
```

```
                    with open(file_path, 'r', encoding='UTF-8') as f:
                        content = f.read()
                        texts[file] = [content]
                except IOError as e:
                    print(f"Error reading file {file_path}: {e}")
    return texts


def extract_texts_from_tar_gz(tar_gz_path):
    """
    Extracts and parses .sgm files from a tar.gz archive.
    """
    texts = {}
    try:
        with tarfile.open(tar_gz_path, "r:gz") as tar:
            for member in tar.getmembers():
                if member.name.endswith('.sgm'):
                    f = tar.extractfile(member)
                    if f:
                        content = f.read().decode('ISO-8859-1')
                        texts[member.name] = extract_body_text(content)
    except (tarfile.TarError, IOError) as e:
        print(f"Error processing tar.gz file: {e}")
    return texts


def extract_body_text(sgm_content):
    """
    Extracts <BODY> content from .sgm file format.
    """
    pattern = r'<BODY>(.*?)</BODY>'
    return re.findall(pattern, sgm_content, re.DOTALL)

def sample_texts(texts_by_filename, sample_size):
    sampled_texts = []
    # Randomly sample files if the number of files exceeds sample_size
    file_keys = list(texts_by_filename.keys())
    if len(file_keys) > sample_size:
        sampled_files = random.sample(file_keys, sample_size)
    else:
        sampled_files = file_keys

    for filename in sampled_files:
        texts = texts_by_filename[filename]
        if isinstance(texts, list):  # Handling for .sgm file contents or
multiple texts in a file
            if len(texts) > sample_size:
                sampled_texts.extend([(filename, text) for text in
random.sample(texts, sample_size)])
            else:
                sampled_texts.extend([(filename, text) for text in texts])
        else:  # Handling for individual text files (.txt files in a folder)
            # In this case, 'texts' is a single string, so we add it directly
            sampled_texts.append((filename, texts))

    return sampled_texts
```

```
def clean_text(text, stop_words, nlp):
    # Remove HTML tags and digits
    text = re.sub(r'<.*?>|\d+|[^a-zA-Z\s]', '', text)

    # Process the text with StanfordNLP for tokenization and lemmatization
    doc = nlp(text)

    # Extract lemmatized words, filtering out stopwords and None values
    lemmatized_words = [word.lemma for sent in doc.sentences for word in
sent.words if
                        word.lemma and word.lemma.strip() and word.text.lower()
not in stop_words]

    # Join the lemmatized words back into a string
    lemmatized_text = ' '.join(lemmatized_words)

    return lemmatized_text

def calculate_tf_idf(texts):
    # Step 1: Calculate TF (Term Frequency) for each word in each document
    tf = defaultdict(lambda: defaultdict(int))

    for i, (_, text) in enumerate(texts):
        words = text.split()
        for word in words:
            tf[i][word] += 1

    # Step 2: Calculate DF (Document Frequency) for each word across all
documents
    df = defaultdict(int)

    for i in range(len(texts)):
        unique_words = set(tf[i].keys())
        for word in unique_words:
            df[word] += 1

    # Step 3: Calculate IDF (Inverse Document Frequency) for each word
    N = len(texts)
    idf = {word: math.log(N / (df[word] + 1)) for word in df}

    # Step 4: Calculate TF-IDF scores for each word in each document
    tfidf = defaultdict(lambda: defaultdict(float))

    for i in range(len(texts)):
        for word, freq in tf[i].items():
            tfidf[i][word] = freq * idf[word]

    return tfidf

def replace_with_top_keywords(cleaned_texts, top_keywords):
    # Create a set of top keywords for easy lookup
    top_keywords_set = set(keyword for keyword, _ in top_keywords)

    # Initialize a list to hold the updated texts
    updated_texts = []

    # Iterate over each text in the cleaned_texts
```

```
    for filename, text in cleaned_texts:
        # Split the text into individual words
        words = text.split()

        # Create a set of unique keywords found in the text
        # This filters out any word not in the top_keywords_set and removes
empty strings
        unique_keywords = {word for word in words if word in top_keywords_set
and word.strip()}

        # Join the unique keywords with commas to create a string
        updated_text = ','.join(unique_keywords)

        # Append the tuple of filename and the updated text (string of
keywords) to the updated_texts list
        updated_texts.append((filename, updated_text))

    return updated_texts

def init_apriori(dataset, min_support):
    # Declaring the lists and sets to house the 1-itemset
    list_dataset = dataset.values.tolist()
    first_candid = set()

    for row in list_dataset:
        for element in row:
            if isinstance(element, list):
                for item in element:
                    if item and not pd.isna(item):  # Check for non-null and
non-empty
                        first_candid.add(item)
            else:
                if element and not pd.isna(element):  # Check for non-null and
non-empty
                    first_candid.add(element)

    first_candid = sorted(list(first_candid))

    # Begin to apply the apriori principle
    count_apriori_list = count_itemsets(list_dataset, first_candid)
    total_trans = len(list_dataset)
    reduce_apriori_list = reduce_itemsets(count_apriori_list, min_support,
total_trans)

    return reduce_apriori_list, count_apriori_list


def count_itemsets(list_dataset, candid_list):
    count_elem_list = {item: 0 for item in candid_list}
    for transaction in list_dataset:
        # Flatten the transaction if it's a list of lists
        if transaction and isinstance(transaction[0], list):
            flattened_transaction = [item for sublist in transaction for item
in sublist]
        else:
            flattened_transaction = transaction
```

```
        for candidate in candid_list:

            if isinstance(candidate, tuple):
                # Check if the candidate tuple is a subset of the flattened
transaction set
                if set(candidate).issubset(set(flattened_transaction)):
                    count_elem_list[candidate] += 1
            else:
                # Check if the single item candidate is in the flattened
transaction
                if candidate in flattened_transaction:
                    count_elem_list[candidate] += 1

    return count_elem_list

def reduce_itemsets(count_itemsets, min_support, total_trans):
    # Reduce the itemsets where the support value does not meet the minimum
support value
    frequent_itemsets = {}

    for items, item_count in count_itemsets.items():
        if item_count / total_trans >= min_support:
            frequent_itemsets[items] = item_count

    return frequent_itemsets

def generate_candidates(prev_frequent, k):
    candidates = set()

    # Calculate total number of iterations for the progress bar
    total_iterations = len(prev_frequent) ** 2
    progress_bar = tqdm(total=total_iterations, desc="Generating candidates")

    update_freq = 1000  # Update the progress bar every 5 iterations
    iteration_count = 0

    for prev1 in prev_frequent:
        for prev2 in prev_frequent:
            merged = sorted(list(set(prev1) | set(prev2)))
            if len(merged) == k:
                candidates.add(tuple(merged))

            iteration_count += 1
            # Update progress bar every 5 iterations or at the end
            if iteration_count % update_freq == 0 or iteration_count ==
total_iterations:
                progress_bar.update(min(update_freq, total_iterations -
progress_bar.n))

    progress_bar.close()
    return candidates

def extract_rules(frequent_itemsets, min_confidence):
    # Begin to create the association rules for the known frequent itemsets
mined
    rules = []
```

```
    for items, count in frequent_itemsets.items():
        if len(items) > 1:
            for i in range(len(items)):
                antecedent = tuple(items[:i] + items[i + 1:])
                precedent = items[i]
                conf = count / frequent_itemsets.get(antecedent, 1)

                if conf >= min_confidence:
                    # Format the rule as a string
                    rule_string = f"{set(antecedent)} => {precedent}
(Confidence: {conf * 100:.2f}%)"
                    rules.append(rule_string)
    return rules

def apriori(dataset, min_support, min_confidence):
    # First level candidates
    reduced, counts = init_apriori(dataset, min_support)

    # Hold the frequent itemsets and counts
    all_frequent = {tuple([k]): v for k, v in reduced.items()}

    k = 2

    # Enter this loop if we found initial frequent itemsets
    while True:
        candidates = generate_candidates(all_frequent.keys(), k)
        if not candidates:
            break

        count = count_itemsets(dataset.values.tolist(), candidates)

        frequent = reduce_itemsets(count, min_support, len(dataset))

        if not frequent:
            break

        all_frequent.update(frequent)
        k += 1

    # Extract association rules
    rules = extract_rules(all_frequent, min_confidence)

    return all_frequent, rules

def prepare_data_for_apriori(cleaned_texts):
    # Convert the list of tuples into a DataFrame
    df = pd.DataFrame.from_records(cleaned_texts, columns=['Filename',
'Keywords'])
    df['Keywords'] = df['Keywords'].str.split(',')

    # Drop the Filename as it's not needed for Apriori
    df = df.drop(columns=['Filename'])

    return df

def main():
    args = parse_arguments(sys.argv[1:])
```

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

    # Set of English stopwords
    stop_words = set(stopwords.words('english'))

    # Initializing Stanford CoreNLP
    stanza.download('en')
    nlp = stanza.Pipeline(lang='en', processors='tokenize,pos,lemma')

    # Extract texts by filename
    if args.file.endswith('.tar.gz'):
        print("Extracting all texts from each .sgm file in .tar.gz file.")
        texts_by_filename = extract_texts_from_tar_gz(args.file)
    else:
        print("Extracting all .txt files from folder path.")
        texts_by_filename = extract_texts_from_directory(args.file)

    # Flatten the texts_by_filename to a list of tuples (filename, text)
    all_texts = [(filename, text) for filename, texts in
texts_by_filename.items() for text in texts]

    # Decide whether to use all texts or perform random sampling
    if args.sample_size >= len(all_texts):
        print("\nUsing all available texts without sampling.")
        sampled_texts = all_texts
    else:
        print(f"\nNow commencing random sampling process, k =
{args.sample_size}.")
        sampled_texts = random.sample(all_texts, args.sample_size)


    # Cleaning and Lemmatizing words
    print("\nNow cleaning and lemmatizing words to extract relevant keywords
from document database.")

    cleaned_texts = []
    interval_time = time.time() #record the start time

    for count, (filename, text) in enumerate(sampled_texts):
        cleaned_text = clean_text(text, stop_words, nlp)
        cleaned_texts.append((filename, cleaned_text))

        # Check if 120 seconds have passed and print the message
        if time.time() - interval_time >= 120:
            print("\nProcessing dataset...")
            interval_time = time.time()  # Update the last_time to the current
time

    print("\nCalculating tf-idf scores for each keyword.")
    tfidf_scores = calculate_tf_idf(cleaned_texts)
    top_keywords = []

    print("\nRanking top keywords based on tf-idf scores and taking the top 200
keywords for the document database.")
    for i, _ in enumerate(cleaned_texts):
```

```
        top_keywords.extend([(word, score) for word, score in
tfidf_scores[i].items()])

    top_keywords = sorted(top_keywords, key=lambda x: x[1], reverse=True)[:200]

    cleaned_texts_with_keywords = replace_with_top_keywords(cleaned_texts,
top_keywords)

    print(f"\nFirst 10 documents being represented by td-idf ranked keywords:",
cleaned_texts_with_keywords[:10])

    # Prepare data for Apriori
    prepared_data = prepare_data_for_apriori(cleaned_texts_with_keywords)

    # Run Apriori algorithm
    print("\nBeginning Apriori Algorithm to generate frequent itemsets and
association rules.")
    freq_itemsets, rules = apriori(prepared_data, args.min_support,
args.min_confidence)

    # Create a single string to list all frequent itemsets
    itemsets_str = ', '.join([f"{set(itemset)}" for itemset, count in
freq_itemsets.items()])

    print("Frequent Itemsets:")
    print(itemsets_str)

    if rules:
        print(f"Rules:")
        for rule in rules:
            print(rule)
    else:
        print("No rules found for this confidence/support value.")

if __name__ == '__main__':
    main()
```

Text Files

reut2-002.sgm file

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET" OLDID="18428"
NEWID="2010">
<DATE> 5-MAR-1987 09:26:17.58</DATE>
<TOPICS><D>acq</D></TOPICS>
<PLACES><D>france</D><D>usa</D><D>west-germany</D><D>netherlands</D><D>sweden</
D></PLACES>
<PEOPLE></PEOPLE>
<ORGS></ORGS>
<EXCHANGES></EXCHANGES>
<COMPANIES></COMPANIES>
<UNKNOWN>
&#5;&#5;&#5;F
&#22;&#22;&#1;f0033&#31;reute
d f BC-FIVE-GROUPS-APPLY-TO   03-05 0118</UNKNOWN>
<TEXT>&#2;
<TITLE>FIVE GROUPS APPLY TO BUY FRENCH TELEPHONE GROUP</TITLE>
<DATELINE>    PARIS, March 5 - </DATELINE><BODY>Five consortia have applied to
buy the
French state-owned telephone equipment manufacturer &lt;Cie
Generale de Constructions Telephoniques (CGCT)>, which will
give the owners control of 16 pct of the French telephone
switching market, sources close to Finance Minister Edouard
Balladur said.
    The French government has given itself until the end of
April to decide which applicant will be accepted, they added.
    While several foreign groups have said they want to gain a
foothold in the French market, their potential stake in CGCT is
limited to 20 pct under privatisation laws passed last year,
with 80 pct to be left in French hands.
    The Finance Ministry sources gave no details of the groups
interested in CGCT, but several have publicly announced their
candidacies.
    U.S. Telecommunications giant American Telephone and
Telegraph Co &lt;T.N> which has been at the centre of the two-year
battle for CGCT, has teamed up with the Dutch-based &lt;Philips
Telecommunications Industrie B.V.>, a subsidiary of NV Philips
Gloeilampenfabriek &lt;PGLO.AS> and &lt;Societe Anonyme de
Telecommunications> (SAT) to present a joint bid, in
association with holding company Cie du Midi SA &lt;MCDP.P> and
five French investment funds.
    A second bid has come from the West German electronics
group Siemens AG &lt;SIEG.F>, which hopes to take a 20 pct stake
in CGCT, with the French telecommunications &lt;Jeumont-Schneider>
taking the remaining 80 pct.
    Sweden's &lt;AB LM Ericsson> has also submitted a bid for the
maximum 20 pct permitted, in association with French defence
electronics group &lt;Matra>, which would hold between 40 and 49
pct, and construction group &lt;Bouygues>.
    Matra has already acquired CGCT's private telephone
business.
REUTER...
```

```
&#3;</BODY></TEXT>
</REUTERS>
```

Description – Text excerpt from reut2-002.sgm file NEWID# = 2010, used for text mining

bbc/032p.txt file

```
Hunt demo at Labour meeting

Pro-hunt supporters are set to protest at Labour's spring conference.

The Countryside Alliance says it expects up to 4,000 supporters to demonstrate
against the hunting ban. They have agreed to keep to a demonstration site on
the other side of the River Tyne from the conference venue in Gateshead. A bid
to overturn the law banning hunting with dogs in England and Wales has begun in
the Court of Appeal. The ban comes into force on 18 February. The Court of
Appeal is expected to rule early next week on whether the alliance's challenge
has succeeded. Richard Dodd, regional director of the Countryside Alliance,
said he expected between 2,000 and 4,000 supporters in Tyneside to make their
protest, with hunt horns and placards.

Campaigners have been asked not to bring any animals or alcohol.

Mr Dodd said he did not believe there would be any repeat of the trouble which
marred the pro-hunt demonstration outside Parliament in September. "We are
holding a static demonstration, just to remind Labour that we are not going
away," he said. Northumbria Police said the pedestrian Millennium Bridge, by
the demonstration site, will be shut if necessary. But Assistant Chief
Constable David Warcup has liaised with several protest groups and said all
negotiations had gone well. Fathers 4 Justice, pensioners' rights activists and
Stop the War campaigners were also expected to demonstrate during the three-day
conference which starts on Friday. Pro-hunt campaigners claims the 1949
Parliament Act - which extends the right of the House of Commons to overrule
the House of Lords - was itself invalid because it was never passed by peers.
The High Court last month ruled the act was valid and the proposed hunting ban
was lawful. Pro-hunt supporters formally launched their second legal challenge
to the ban in London's High Court on Thursday. The Countryside Alliance has
lodged papers seeking a judicial review on human rights grounds. Animal welfare
groups have welcomed the ban, many of whom have campaigned for a ban for
decades saying hunting is cruel and unnecessary.
```

Description – An example of text file in the bbc folder used for text mining.

bbcsport/100r.txt

```
Dawson wins England squad recall

Wasps scrum-half Matt Dawson has been recalled to England's training squad
ahead of the RBS Six Nations and been reinstated in the Elite Player Squad.

Coach Andy Robinson dropped Dawson for the autumn Tests after he missed
training to film 'A Question of Sport.' "I always said I would consider
bringing Matt back if I felt he was playing well," Robinson said. "He merits
his return on current form." Newcastle's 18-year-old centre Mathew Tait is also
in the training squad. "It's obviously an honour to be asked to train with
England," said Tait, who has burst into contention recently. "I look forward to
going down and doing the sessions, but the most important thing at the moment
is Sunday's game against Newport, so I'm not looking any further than that."
Robinson has invited 42 players to attend a three-day session in Leeds next
week, in which his squad will train in part with the Leeds Rhinos rugby league
squad.

With Mike Tindall ruled out of the opening two matches and Will Greenwood
sidelined for the entire Six Nations, Tait is one of six or seven contenders
for the two centre berths. Stuart Abbott, Jamie Noon, Ollie Smith, Olly Barkley
and Henry Paul - who retains his place despite his early substitution against
Australia - are also in the mix. Ben Cohen could also be considered after
switching from the wing for his club Northampton recently. Prop Phil Vickery
and lock Simon Shaw both return to the squad after missing the autumn Tests
through injury, while Wasps wing Tom Voyce is recalled. The group also includes
Bath flanker Andy Beattie and Leicester hooker George Chuter. "Beattie has
matured greatly as a player these past two seasons," Robinson said. Jonny
Wilkinson, Tindall and Martin Corry have all been included despite their
unavailability for the opening two matches against Wales and France.

The revised 56-man elite squad includes Wasps hooker Phil Greening, who
replaces the retired Mark Regan, and Sale wing Mark Cueto. Cueto was selected
for the November internationals despite not being part of the group, but scored
four tries in three England appearances. Leicester scrum-half Harry Ellis has
also been promoted from the senior national academy, and will contest the
number nine jersey with Dawson and Gloucester's Andy Gomarsall. The players in
Robinson's elite squad can only play 32 matches for club and country. They can
be called up for a total of 16 training days in addition to the recognised
international weeks for each of the years leading up to the next World Cup.

Balshaw, Cohen, Cueto, Lewsey, Robinson, Simpson-Daniel, Voyce, Abbott, Noon,
Paul, Smith, Tait, Tindall, Barkley, Hodgson, King, Wilkinson, Dawson, Ellis,
Gomarsall.

Chuter, Thompson, Titterrell, Rowntree, Sheridan, Stevens, Vickery, White,
Borthwick, Brown, L Deacon, Grewcock, Kay, Shaw, Beattie, Corry, Forrester,
Hazell, Jones, Moody, Vyvyan, J Worsley.

Abbott, Balshaw, Borthwick, A Brown, Chuter, Cohen, Corry, Cueto, Dawson,
Ellis, Flatman, Gomarsall, Greening, Greenwood, Grewcock, Hazell, Hill,
Hodgson, Kay, King, Lewsey, Moody, Noon, Paul, Robinson, Rowntree, Shaw,
Simpson-Daniel, Thompson, Tindall, Titterrell, Vickery, Vyvyan, White,
Wilkinson, J Worsley, M Worsley.

Barkley, Beattie, Christophers, L Deacon, Forrester, C Jones, Palmer, Rees,
```

```
Sheridan, Skinner, Smith, Stevens, Tait, Voyce.

Dowson, Haughton, Monye, Roques, P Sanderson.
```

Description - An example of text file in the bbcsport folder used for text mining.

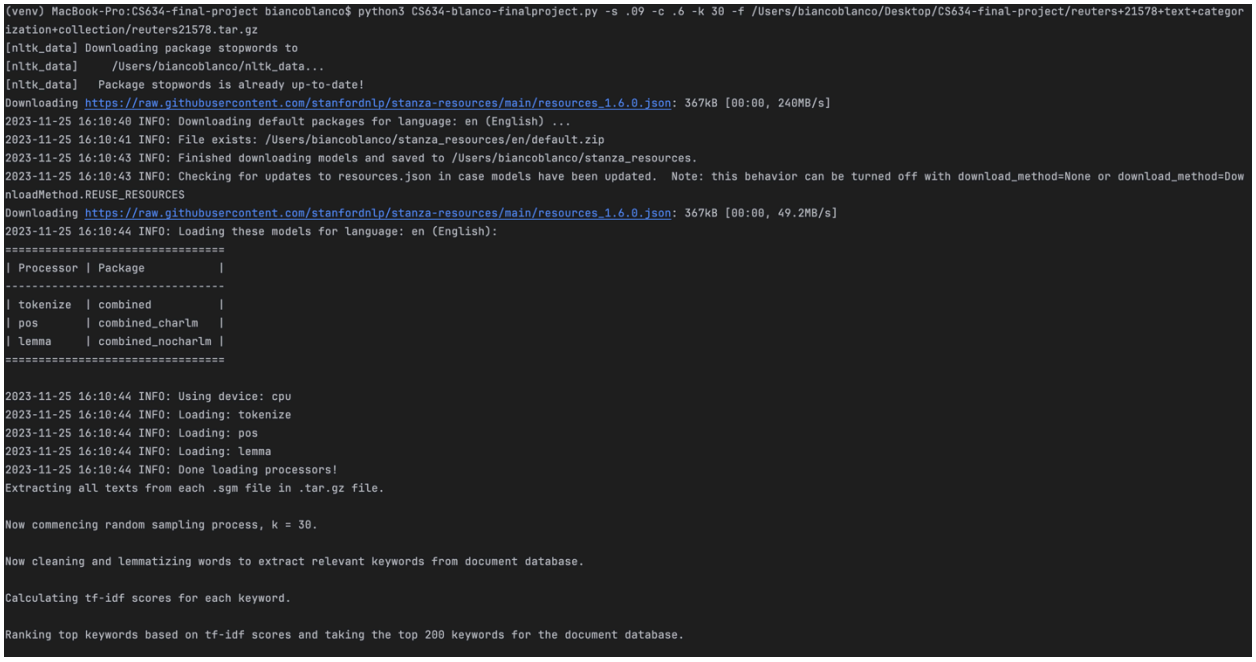
Link to all data

Link to dropbox which contains each document database correctly labeled with all files used.

Screenshots

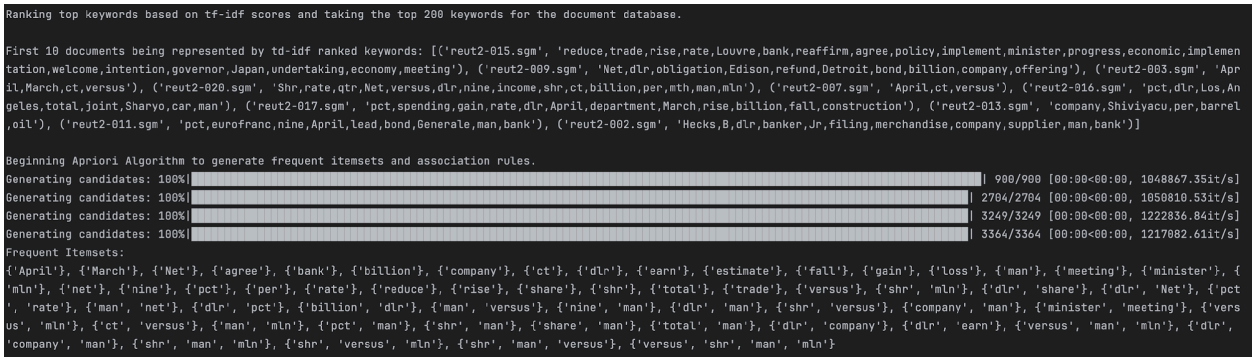
Screenshot 1

Description – Screenshot 1 showing different user inputs, and using the Reuters21578 dataset.



Screenshot 2

Description – Screenshot 2 is showing the first 10 documents replacing the text of the document of the subsequent of the top keywords. Also, we see the code executing the Apriori Algorithm, the timelapse of the generated candidates process, and the frequent item-sets.



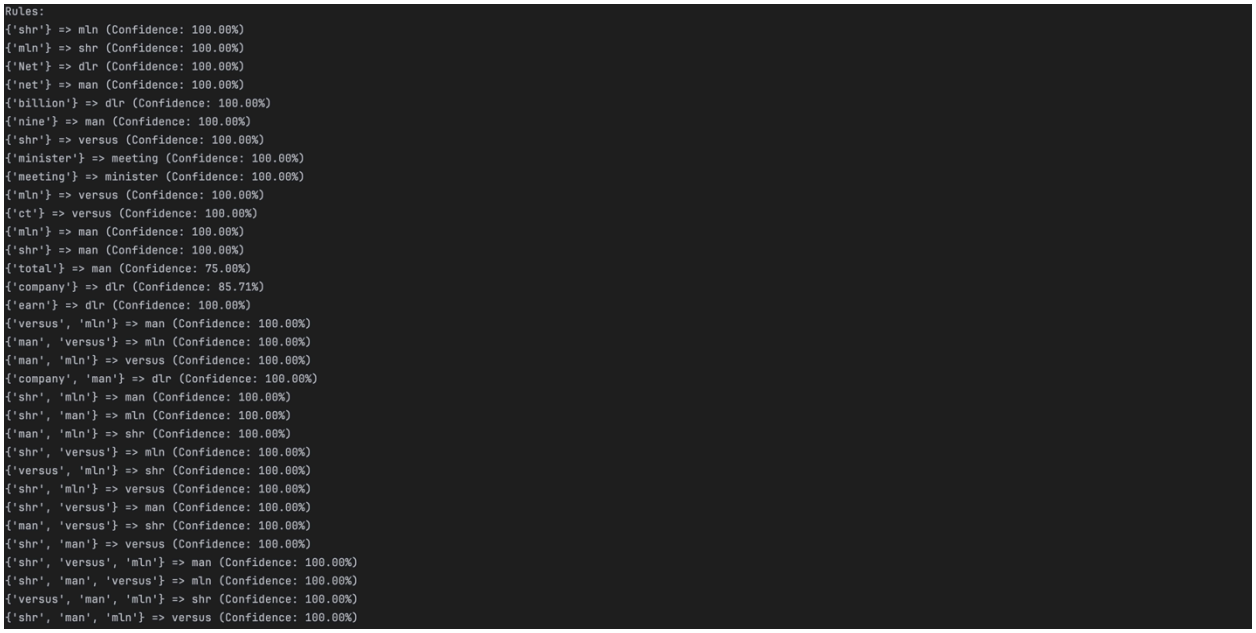
```
Ranking top keywords based on tf-idf scores and taking the top 200 keywords for the document database.

First 10 documents being represented by td-idf ranked keywords: [('reut2-015.sgm', 'reduce,trade,rise,rate,Louvre,bank,reaffirm,agree,policy,implement,minister,progress,economic,implementation,welcome,intention,governor,Japan,undertaking,economy,meeting'), ('reut2-009.sgm', 'Net,dlr,obligation,Edison,refund,Detroit,bond,billion,company,offering'), ('reut2-003.sgm', 'April,March,ct,versus'), ('reut2-020.sgm', 'Shr,rate,qtr,Net,versus,dlr,nine,income,shr,ct,billion,per,mth,man,mln'), ('reut2-007.sgm', 'April,ct,versus'), ('reut2-016.sgm', 'pct,dlr,Los,Angeles,total,joint,Sharyo,car,man'), ('reut2-017.sgm', 'pct,spending,gain,rate,dlr,April,department,March,rise,billion,fall,construction'), ('reut2-013.sgm', 'company,Shiviyacu,per,barrel,oil'), ('reut2-011.sgm', 'pct,eurofranc,nine,April,lead,bond,Generale,man,bank'), ('reut2-002.sgm', 'Hecks,B,dlr,banker,Jr,filing,merchandise,company,supplier,man,bank')]

Beginning Apriori Algorithm to generate frequent itemsets and association rules.
Generating candidates: 100%|████████████████████████████████████████████████████████████████████████████████| 900/900 [00:00<00:00, 1048867.35it/s]
Generating candidates: 100%|████████████████████████████████████████████████████████████████████████████| 2704/2704 [00:00<00:00, 1050810.53it/s]
Generating candidates: 100%|████████████████████████████████████████████████████████████████████████████| 3249/3249 [00:00<00:00, 1222836.84it/s]
Generating candidates: 100%|████████████████████████████████████████████████████████████████████████████| 3364/3364 [00:00<00:00, 1217082.61it/s]
Frequent Itemsets:
{'April'}, {'March'}, {'Net'}, {'agree'}, {'bank'}, {'billion'}, {'company'}, {'ct'}, {'dlr'}, {'earn'}, {'estimate'}, {'fall'}, {'gain'}, {'loss'}, {'man'}, {'meeting'}, {'minister'}, {'mln'}, {'net'}, {'nine'}, {'pct'}, {'per'}, {'rate'}, {'reduce'}, {'rise'}, {'share'}, {'shr'}, {'total'}, {'trade'}, {'versus'}, {'shr', 'mln'}, {'dlr', 'share'}, {'dlr', 'Net'}, {'pct', 'rate'}, {'man', 'net'}, {'dlr', 'pct'}, {'billion', 'dlr'}, {'man', 'versus'}, {'nine', 'man'}, {'dlr', 'man'}, {'shr', 'versus'}, {'company', 'man'}, {'minister', 'meeting'}, {'versus', 'mln'}, {'ct', 'versus'}, {'man', 'mln'}, {'pct', 'man'}, {'shr', 'man'}, {'share', 'man'}, {'total', 'man'}, {'dlr', 'company'}, {'dlr', 'earn'}, {'versus', 'man', 'mln'}, {'dlr', 'company', 'man'}, {'shr', 'man', 'mln'}, {'shr', 'versus', 'mln'}, {'shr', 'man', 'versus'}, {'versus', 'shr', 'man', 'mln'}
```

Screenshot 3

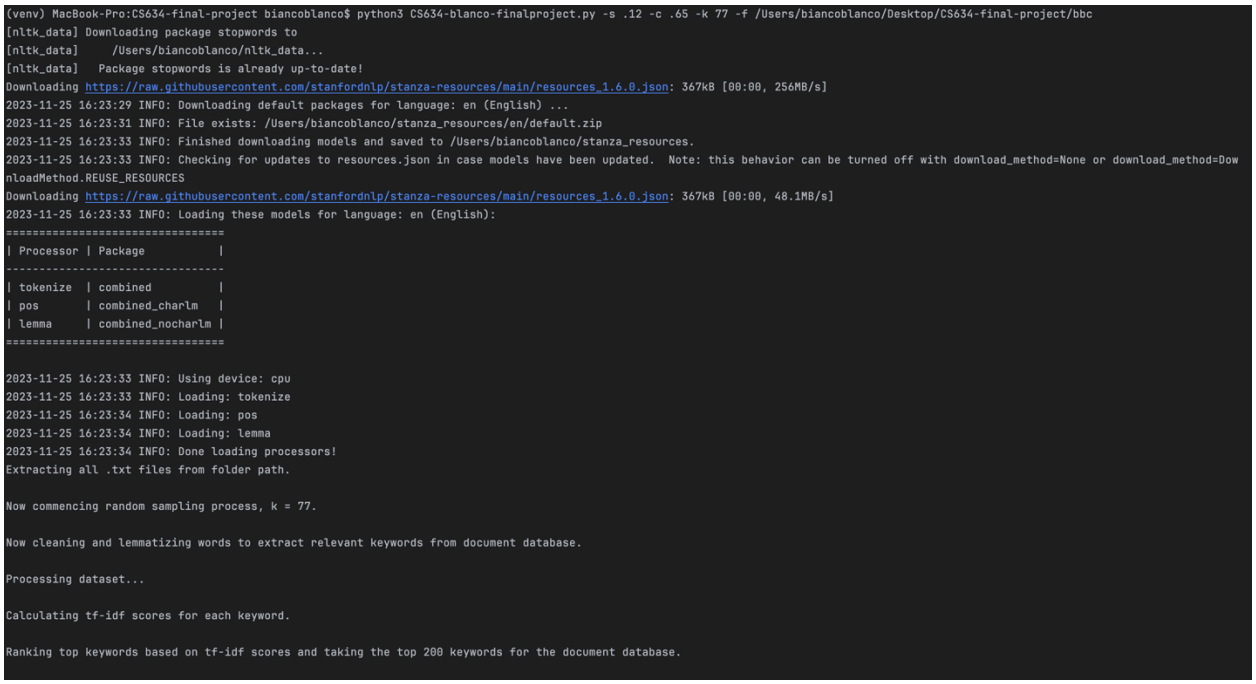
Description – Screenshot 3 is the association rules generated of confidence value of 60% and higher for the Reuters21578 dataset.



```
Rules:
{'shr'} => mln (Confidence: 100.00%)
{'mln'} => shr (Confidence: 100.00%)
{'Net'} => dlr (Confidence: 100.00%)
{'net'} => man (Confidence: 100.00%)
{'billion'} => dlr (Confidence: 100.00%)
{'nine'} => man (Confidence: 100.00%)
{'shr'} => versus (Confidence: 100.00%)
{'minister'} => meeting (Confidence: 100.00%)
{'meeting'} => minister (Confidence: 100.00%)
{'mln'} => versus (Confidence: 100.00%)
{'ct'} => versus (Confidence: 100.00%)
{'mln'} => man (Confidence: 100.00%)
{'shr'} => man (Confidence: 100.00%)
{'total'} => man (Confidence: 75.00%)
{'company'} => dlr (Confidence: 85.71%)
{'earn'} => dlr (Confidence: 100.00%)
{'versus', 'mln'} => man (Confidence: 100.00%)
{'man', 'versus'} => mln (Confidence: 100.00%)
{'man', 'mln'} => versus (Confidence: 100.00%)
{'company', 'man'} => dlr (Confidence: 100.00%)
{'shr', 'mln'} => man (Confidence: 100.00%)
{'shr', 'man'} => mln (Confidence: 100.00%)
{'man', 'mln'} => shr (Confidence: 100.00%)
{'shr', 'versus'} => mln (Confidence: 100.00%)
{'versus', 'mln'} => shr (Confidence: 100.00%)
{'shr', 'mln'} => versus (Confidence: 100.00%)
{'shr', 'versus'} => man (Confidence: 100.00%)
{'man', 'versus'} => shr (Confidence: 100.00%)
{'shr', 'man'} => versus (Confidence: 100.00%)
{'shr', 'versus', 'mln'} => man (Confidence: 100.00%)
{'shr', 'man', 'versus'} => mln (Confidence: 100.00%)
{'versus', 'man', 'mln'} => shr (Confidence: 100.00%)
{'shr', 'man', 'mln'} => versus (Confidence: 100.00%)
```

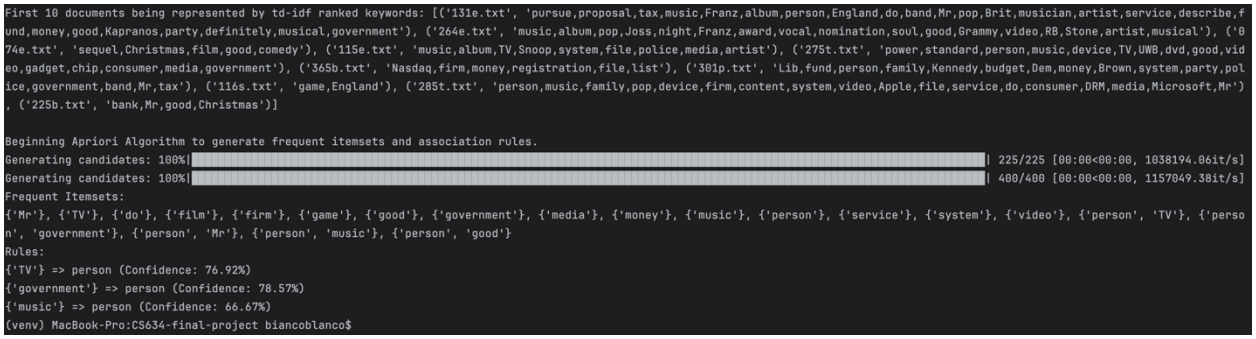
Screenshot 4

Description – Screenshot 4 is showing different user inputs and using the bbc dataset.



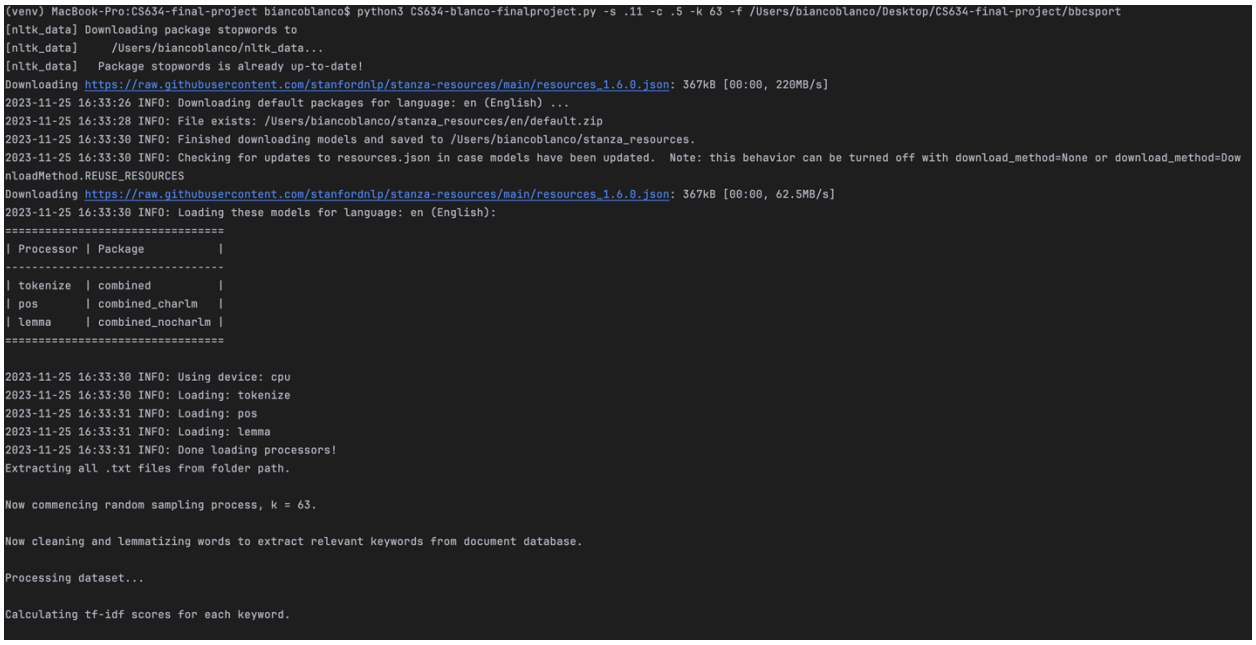
Screenshot 5

Description – Screenshot 5 is showing the first 10 documents replacing the text of the document of the subsequent of the top keywords. Also, we see the code executing the Apriori Algorithm, the timelapse of the generated candidates process, and association rules generated with confidence value 65%.



Screenshot 6

Description – Screenshot 6 showing different user inputs and using the bccsport dataset.



Screenshot 7

Description – Screenshot 7 is showing the first 10 documents replacing the text of the document of the subsequent of the top keywords. Also, we see the code executing the Apriori Algorithm, the timelapse of the generated candidates process, and association rules generated with confidence value 50%.