Programming assignment 1

Rituraj Kulshresth (B18CSE046)
Saurabh Shashikant Burewar (B18CSE050)

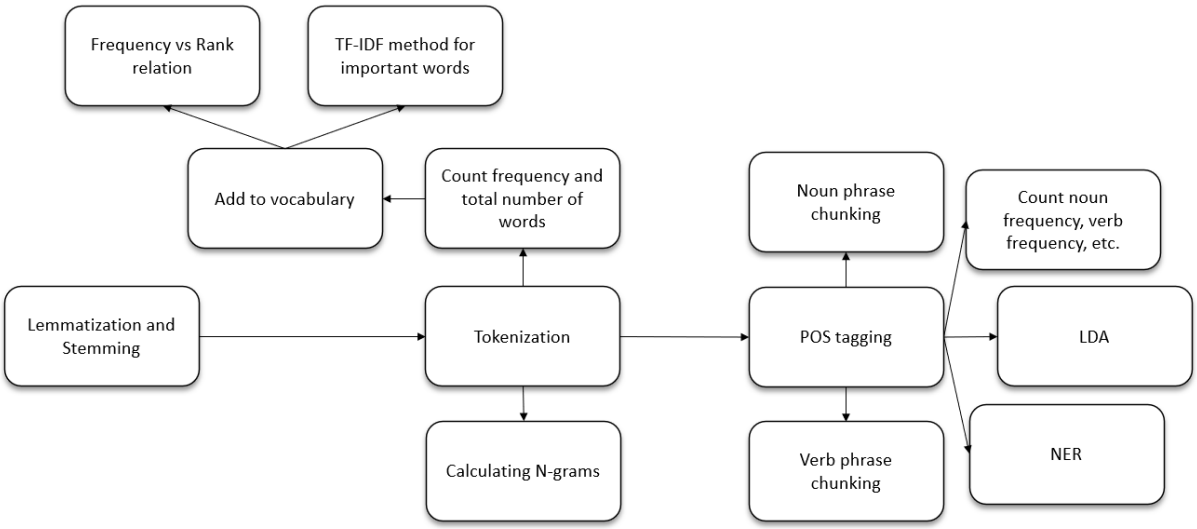
Data

The dataset used is the Movies and TV reviews dataset which is a part of the Amazon reviews dataset (http://jmcauley.ucsd.edu/data/amazon/). This dataset contains over 4 million entries and for this work 25,000 entries have been used. The data is in JSON format as a list of entries where each entry has all details of the review, including rating, reviewer details etc. However, we are interested in the review text only.

We convert the JSON into a Pandas Dataframe which is then used to perform all the subsequent experiments.

Experiments

Pipeline



Details of all the above steps in the pipeline can be found below.

Building vocabulary

To build a vocabulary of all the words in the corpus we need to process the data first. This includes -

- Lemmatization: Lemmatization removes different inflected forms of the same word and reduces it to the same word.

- Stemming: This is similar to lemmatization and reduces morphological variants to the root word.
- Tokenization: We convert the string of words into a list of tokens (words) using the NLTK library.
- Removing punctuations: We also remove all punctuations again (as a precaution) by filtering using regular expressions.
- Removing Stopwords: We remove the words that are considered stopwords by NLTK.

The above is done for every entry of review in our dataset and all the words are stored in a dictionary along with the number of times they occur in the corpus.

Our vocabulary consists of 46,140 words in total with words “movie” occurring more than 20,000 times and words like “film” and “great” more than 10,000 times.

This is a small part of the vocabulary dictionary.

{"movie": 23194, "film": 14704, "one": 12185, "great": 8842, "good": 7702, "like": 6796, "time": 5914, "story": 5575, "well": 4983, "love": 4595, "would": 4315, "really": 4149, "see": 4138, "first": 3942, "best": 3941, "also": 3909, "movies": 3905, "dvd": 3813, "even": 3783, "watch": 3688, "life": 3681, …

N-grams

The N-grams are calculated from the processed and tokenized data. We use NLTK functions to calculate N-grams which gives us a list of the words for every N-gram. We join this list to create a string of N-gram. We push this string into our list to maintain all the N-grams.

This is a part of the Tri-gram list that we got from our corpus -

["So sorry I", "sorry I did", "I didnt purchase", "didnt purchase this", "purchase this years", "this years ago", "years ago when", "ago when it", "when it first", "it first came", "first came out", …

POS Tagging

For POS tagging, we use the NLTK function which tags each word as noun, verb, etc. We can see in the output given below how the number of nouns in the text is much larger compared to other tags since it plays the varied roles of subjects, objects, adjectives, etc in a sentence.

{"NN": 455449, "JJ": 237287, "DT": 132109, "RB": 113649, "NNS": 110603, "VBP": 64914, "VBD": 59613, "VBZ": 59546, "IN": 58834, "VBG": 47717, "VB": 46313, "VBN": 38631, …

Chunking (Noun and Verb phrases)

Chunking is the process of creating chunks of words according to some rules. To create noun and verb phrases, we use chunking. Chunking consists of the following steps -

- Defining a grammar: We define a grammar which will act as our rule when we are making chunks of words. For Nouns we use the grammar, *NP: {<DT>? <JJ>* <NN.*>+}.* This means if we have an optional determinar followed by one or more adjectives followed by a noun in any form (singular, plural, etc.). Similarly, for verb phrases we use the grammar, *VP: {<VB.*> <DT>? <JJ>* <NN.*>+ <RB.?>?}.*
- Setting up Parser: We set up a regex parser using NLTK and our grammar.
- Parsing words: We parse the tokenized and tagged words through this parser.
- Extract phrases: The parser returns a tree. We go through this tree using NLTK and convert the nodes corresponding to the phrases and convert it to string. These strings are then put in a list.

Top noun phrases (along with number of occurrences in the corpus)

```
"movie": 4539, "the film": 2287, "the movie": 2255, "film": 2230, "this movie": 1966
```

Top verb phrases (along with number of occurrences in the corpus)

```
"linked class link": 301, "love movie": 263, "seen movie": 177, "loved movie": 124, "linked class": 118
```

NER

For NER, we use the NLTK library provided method. This method recognizes a word as person, organization or GPE (location). From the results we have seen this method is not as accurate since a word can have different meanings depending on the context which this method doesn't take into account. For example, us is recognized as a location (as in United States) but is actually the pronoun.

```
{"Believe": "PERSON", "Cathedral": "ORGANIZATION", "VHS": "ORGANIZATION", "LA":
"ORGANIZATION", "Tony": "PERSON", "Billy": "PERSON", "DVD": "ORGANIZATION", "AMAZING":
"ORGANIZATION", "Us": "GPE", …
```

Another observation here is seen after lemmatization and clearing stopwords where most of these words are not recognized at all. Therefore, we decided to not process the corpus extensively before NER.

Important words to describe corpus

To understand how important a word is in a corpus, we have used the TF-IDF method. Here,

- TF means term frequency, which is - number of times a word occurs/total number of words
- IDF means inverse document frequency, which is - log(total number of sentences/number of times a word occurs)

- Then, we define the importance of a word as TF*IDF.

This method decreases the weight of common words and increases that of uncommon words that are not used as much in a document.

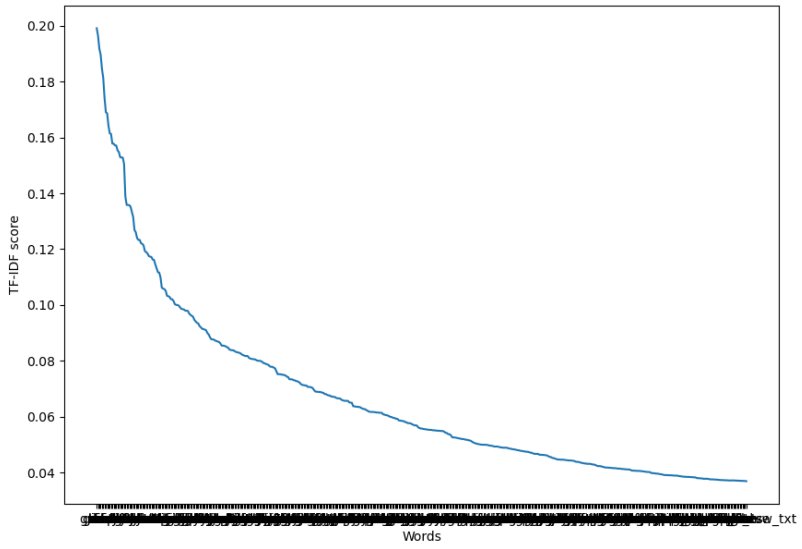
From this, we get the most important words to define our corpus as -

{"great": 0.19910839967106325, "good": 0.19647874190140358, "like": 0.19179943338160274, "one": 0.1896967542090629, "time": 0.18472512797166515, "story": 0.1812688706781215, "well": 0.1741440394857841, "film": 0.1690278639582935, "love": 0.1686573143585988, "would": 0.1642597648300162, "really": 0.16146825507573126, "see": 0.16127825231594747, "first": 0.15778487662140833, "best": 0.15776652043585968, "also": 0.15717621116606634, "movies": 0.15710202425773054, "dvd": 0.1553710297811602, …

Some of these are still common words like “would” or “also” which shows that this method is not completely accurate. However, we do have words to describe our corpus as follows -

“great”, “story”, “film”, “like”, “movies”, “dvd” etc.

If we plot this importance score for the top 500 words, we get something like this -

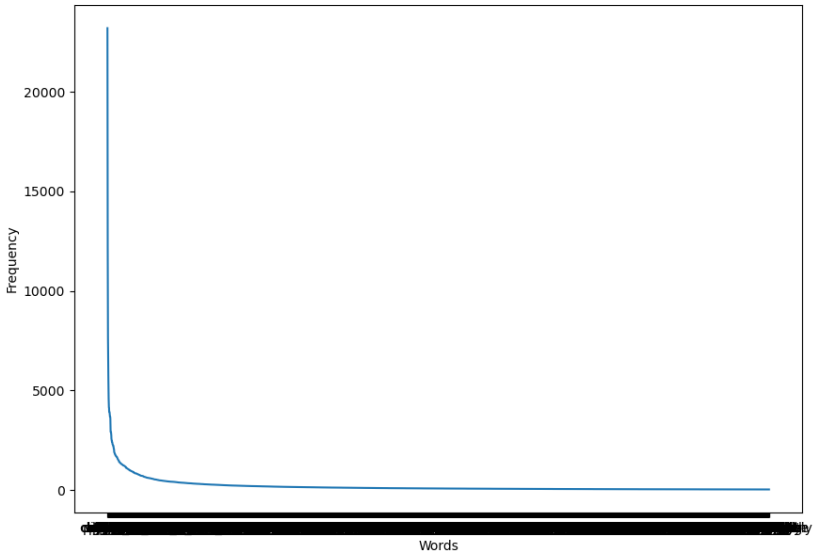


This plot shows how there is a steady decrease in the score and how the top 100 words have a lot higher score than the rest. Even in the top 100, the score difference is very significant. So, we can justify the importance of these words to describe the corpus, since the words at the top are much more important than the ones that follow.

Frequency vs Rank of words

We were already counting the number of occurrences of a word while making our vocabulary. So, we just sort this dictionary and plot the frequency of each word according to the rank of the word.

This is a plot of 5000 words and we can see how frequency decreases sharply (since we have a lot of words). Increasing the number of words will make this drop sharper. This shows that the amount of very high rank words is much less as compared to low rank words. We only have a few words with very high frequency. For a high number of words, the plot resembles a hyperbola which also satisfies the relationship of frequency being inversely proportional to rank which is said by Zipf's law.



Extracting the Topics

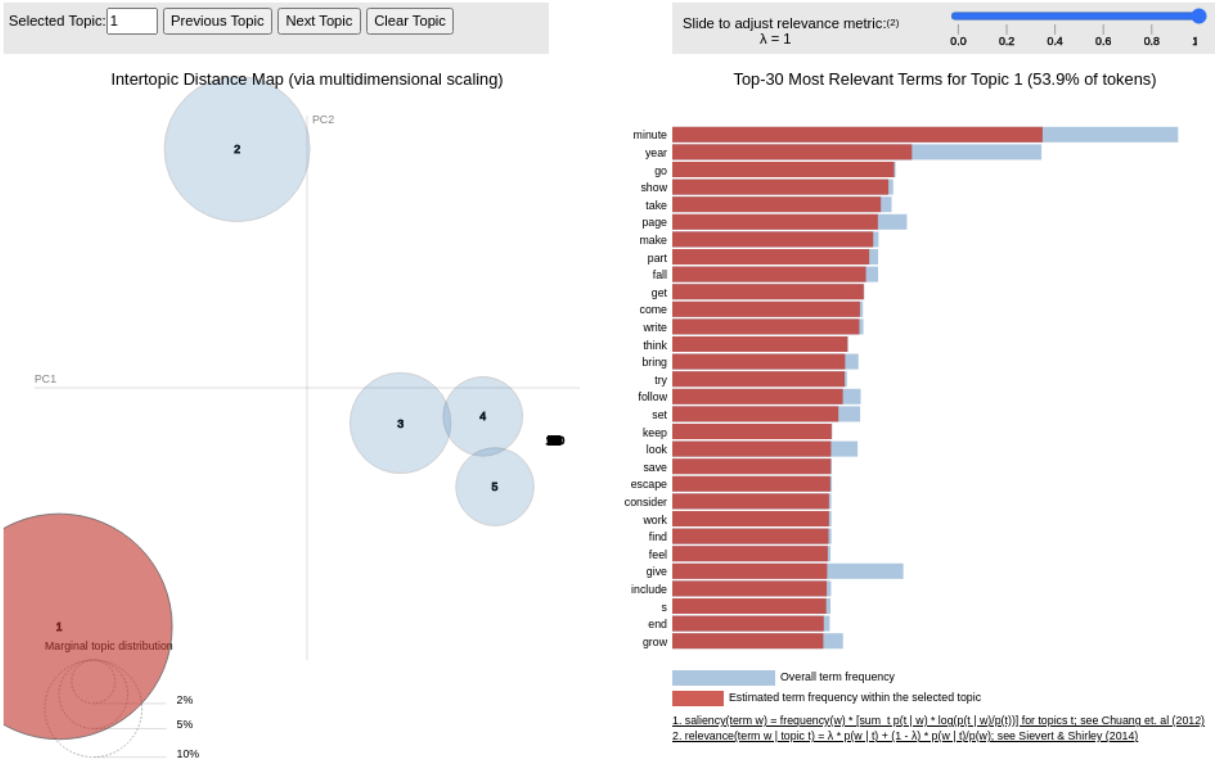
We used LDA to extract the topics from the corpus. In order to start with the extraction of the topic first the words were extracted from the text. We used gensim for LDA. Next the stop words were removed. We used standard English stop words provided by NLTK. Next we Lemmatized the words with the allowed posttags to be 'NOUN', 'ADJ', 'VERB', 'ADV' only. Next we create a dictionary from this lemmatized data. Next we create a corpus in the bow format as this is needed by the LDA function of gensim.

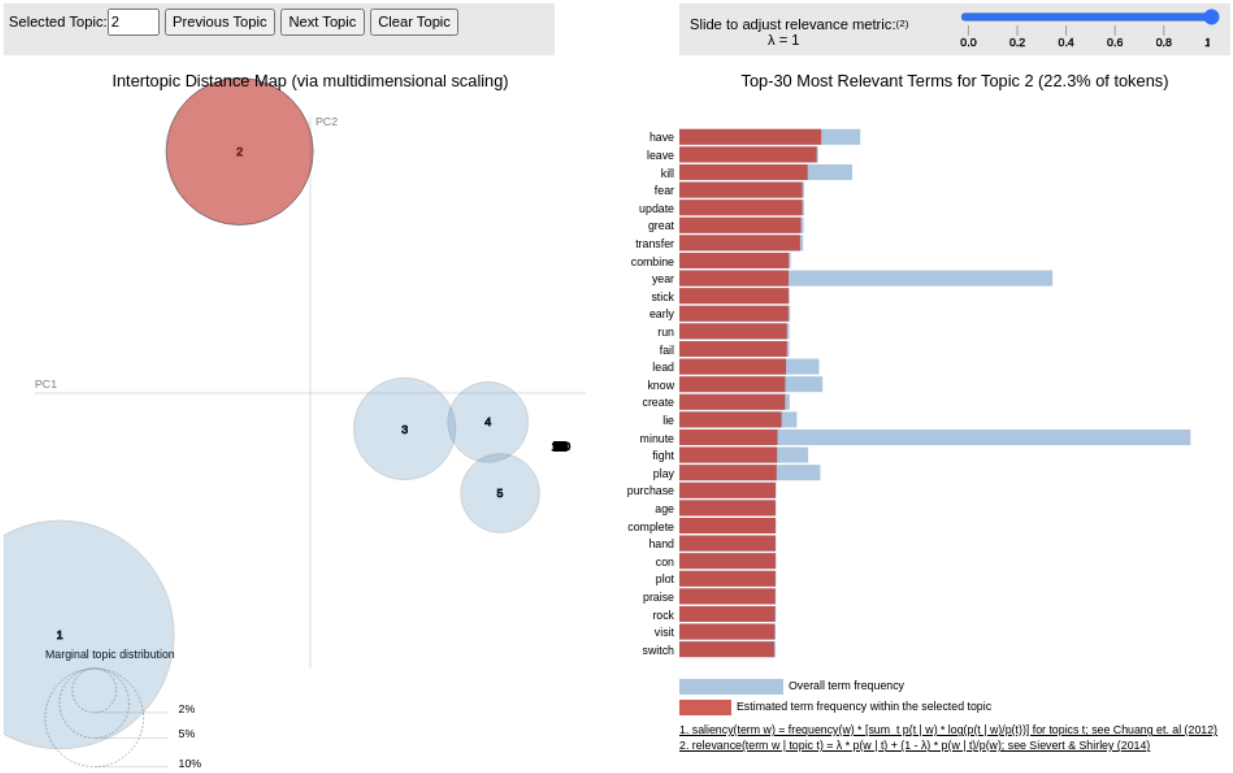
The LDA model gives us the option to tweak the model by changing the minimum probability of the topic that is allowed. It also gives us the option to limit the minimum phi value i.e. each term probability. We can change the number of topics that are extracted and the number of epochs for which the model will run over the dataset. Some of the observations for the topic extraction are:

- Most of the topics generated had a very low probability.
- Most of the topics generated had repetitive words in multiple topics.

- Minimum_probability and minimum_phi_value did not have any major effect on the probability of the topics.
- This may be due to the large number of words present in the dictionary, making individual probabilities very low.

Visualization for some of the topics is as follows:





Some of the topics generated by LDA are:

```
(94,
 '0.000*"minute" + 0.000*"part" + 0.000*"page" + 0.000*"transfer" + '
 '0.000*"love" + 0.000*"get" + 0.000*"have" + 0.000*"come" + 0.000*"laugh" + '
 '0.000*"year"'),
(73,
 '0.000*"minute" + 0.000*"make" + 0.000*"fall" + 0.000*"come" + 0.000*"set" + '
 '0.000*"have" + 0.000*"part" + 0.000*"love" + 0.000*"add" + 0.000*"year"'),
(67,
 '0.000*"tell" + 0.000*"try" + 0.000*"write" + 0.000*"year" + 0.000*"minute" '
 '+ 0.000*"die" + 0.000*"give" + 0.000*"go" + 0.000*"show" + 0.000*"see"'),
(56,
 '0.000*"add" + 0.000*"page" + 0.000*"make" + 0.000*"part" + 0.000*"bring" + '
 '0.000*"end" + 0.000*"fear" + 0.000*"set" + 0.000*"go" + 0.000*"kill"'),
(37,
 '0.000*"try" + 0.000*"close" + 0.000*"show" + 0.000*"year" + 0.000*"hour" + '
 '0.000*"come" + 0.000*"drive" + 0.000*"die" + 0.000*"take" + 0.000*"film"'),
(15,
 '0.000*"man" + 0.000*"show" + 0.000*"year" + 0.000*"give" + 0.000*"see" + '
 '0.000*"minute" + 0.000*"die" + 0.000*"take" + 0.000*"dream" + '
 '0.000*"priviledge"'),
(4,
 '0.000*"minute" + 0.000*"like" + 0.000*"love" + 0.000*"blow" + '
 '0.000*"remember" + 0.000*"film" + 0.000*"steal" + 0.000*"face" + '
 '0.000*"drive" + 0.000*"make"'),
(58,
 '0.000*"part" + 0.000*"find" + 0.000*"minute" + 0.000*"save" + 0.000*"go" + '
 '0.000*"year" + 0.000*"travel" + 0.000*"feel" + 0.000*"look" + 0.000*"fall"'),
(9,
 '0.000*"minute" + 0.000*"end" + 0.000*"bring" + 0.000*"gross" + 0.000*"s" + '
 '0.000*"subtitle" + 0.000*"run" + 0.000*"thank" + 0.000*"set" + '
 '0.000*"view"'),
(82,
 '0.000*"learn" + 0.000*"save" + 0.000*"tell" + 0.000*"part" + 0.000*"minute" '
 '+ 0.000*"lose" + 0.000*"rise" + 0.000*"talk" + 0.000*"become" + '
 '0.000*"close"'),
(91,
 '0.000*"old" + 0.000*"rip" + 0.000*"bar" + 0.000*"story" + 0.000*"touch" + '
 '0.000*"year" + 0.000*"love" + 0.000*"clear" + 0.000*"early" + '
 '0.000*"stick"'),
(17,
 '0.017*"side" + 0.017*"record" + 0.017*"gun" + 0.017*"story" + 0.015*"refer" '
 '+ 0.015*"free" + 0.015*"child" + 0.013*"print" + 0.013*"push" + '
 '0.013*"chance"'),
(72,
 '0.016*"enter" + 0.014*"toy" + 0.014*"care" + 0.014*"surround" + '
 '0.014*"cheer" + 0.012*"stage" + 0.012*"represent" + 0.012*"pull" + '
 '0.012*"explore" + 0.012*"assume"'),
(32,
 '0.016*"use" + 0.013*"word" + 0.013*"equal" + 0.013*"fill" + 0.013*"bar" + '
 '0.013*"contain" + 0.011*"version" + 0.011*"murder" + 0.011*"region" + '
 '0.011*"survive"'),
(69,
 '0.011*"have" + 0.010*"walk" + 0.010*"know" + 0.009*"leave" + 0.009*"lead" + '
 '0.009*"call" + 0.008*"fear" + 0.008*"great" + 0.008*"transfer" + '
 '0.008*"update"'),
(3,
 '0.012*"minute" + 0.009*"year" + 0.006*"see" + 0.006*"give" + 0.005*"page" + '
 '0.005*"show" + 0.005*"go" + 0.005*"take" + 0.005*"fall" + 0.005*"make"')]
```

Analysis

The dataset that we chose was for movies and TV reviews. Hence the reviews were bound to be on various topics each related to the story of the individual movie. Moreover the opinions of the people would also be different. However there was very low frequency of the words that we expected to have a high frequency, such as movie and film. The collection of words was very distributed with low repetitive frequency hence the LDA did not identify the topics correctly. During a search for a possible solution to this we found that the LDA was working properly and it was the dataset which had issues. LDA doesn't work well with large datasets with too much noise or words with low frequency. LDA also missed some words that were high in frequency due to them being misclassified. LDA gave many repetitive topics over its runtime due to the use of Dirichlet distribution. A possible solution to this could be to use logistic normal distribution or any other algebraic distribution.

Future work

- The corpus can be further processed using more stopwords. This will require creating a better set of stopwords.
- We can use more complicated grammar with multiple different rules to make chunking of noun and verb phrases as accurate as possible.
- Since the method for NER provided by NLTK is not as accurate, we can look for a better solution or write our own, if data for it is available.
- The data can be further cleaned to remove unnecessary words with low frequency so that the probability of the topics generated could be better.

Contribution

Rituraj Kulshresth (B18CSE046) - Module 2
Saurabh Shashikant Burewar (B18CSE050) - Module 1

References

- He, Ruining and Julian McAuley. “Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering.” *Proceedings of the 25th International Conference on World Wide Web* (2016): n. pag.
- McAuley, Julian & Targett, Christopher & Shi, Qinfeng & Hengel, Anton. (2015). Image-Based Recommendations on Styles and Substitutes. 10.1145/2766462.2767755.