15CSE358 Natural Language Processing

Case Study Report

Group: 20

Anjali Ragupathi

CB.EN.U4CSE17307

Clustering Books Based on the Similarity of their Contents Using Topic Modelling Techniques

Department of Computer Science and Engineering

Amrita School of Engineering

Amrita Vishwa Vidyapeetham

Coimbatore 641 112

Introduction

Reading is one of the most popular hobbies in the world, as well as one of the oldest. Books have been around for a very long time, most of them containing a wealth of information and entertainment. Of course, the style of writing has changed significantly over the years, and one cannot simply compare Jane Austen to JK Rowling, or Shakespeare to Amish Tripathi. Nevertheless, the actual content of a book written in the 40s may not be that different from a similar book in the 2010s.

If we take a simple approach to analysing books, by tokenizing the text, counting the words and then matching books with similar words in a vector space, we would likely end up with recommendations that fall more within the range of the time period that a particular book exists in. In such cases, purely word-count-based, extrinsic approaches are not advisable.

Topic modelling is an unsupervised learning technique which trains itself on a corpus of multiple documents and extracts a set of topics from them. Each document belongs to a mixture of topics, and each topic is described by a set of words. This technique is not restricted to a simple comparison of two documents; in fact, it searches the latent space of the training data to find topics that can then be used to cluster books into. This is exceptionally helpful for discovering literary themes that transcend the lexical or syntactic form of a text.

In this case study, I will be training a Latent Dirichlet Allocation model on texts from the Brown and Gutenberg corpora. I will also be examining the differences in the results based on application of specific pre-processing steps like lemmatization and noun-only Part-of-Speech tagging. Additionally, I will be implementing a web scraped categorization of a particular book, to match it to a particular theme.

Objective

+ To train a topic modelling algorithm – LDA – on a corpus of books and determine the latent themes
+ To perform hyperparameter tuning and data cleaning for optimization
+ To predict similar books based on the plot of a user-entered book, by looking at books with similar themes.

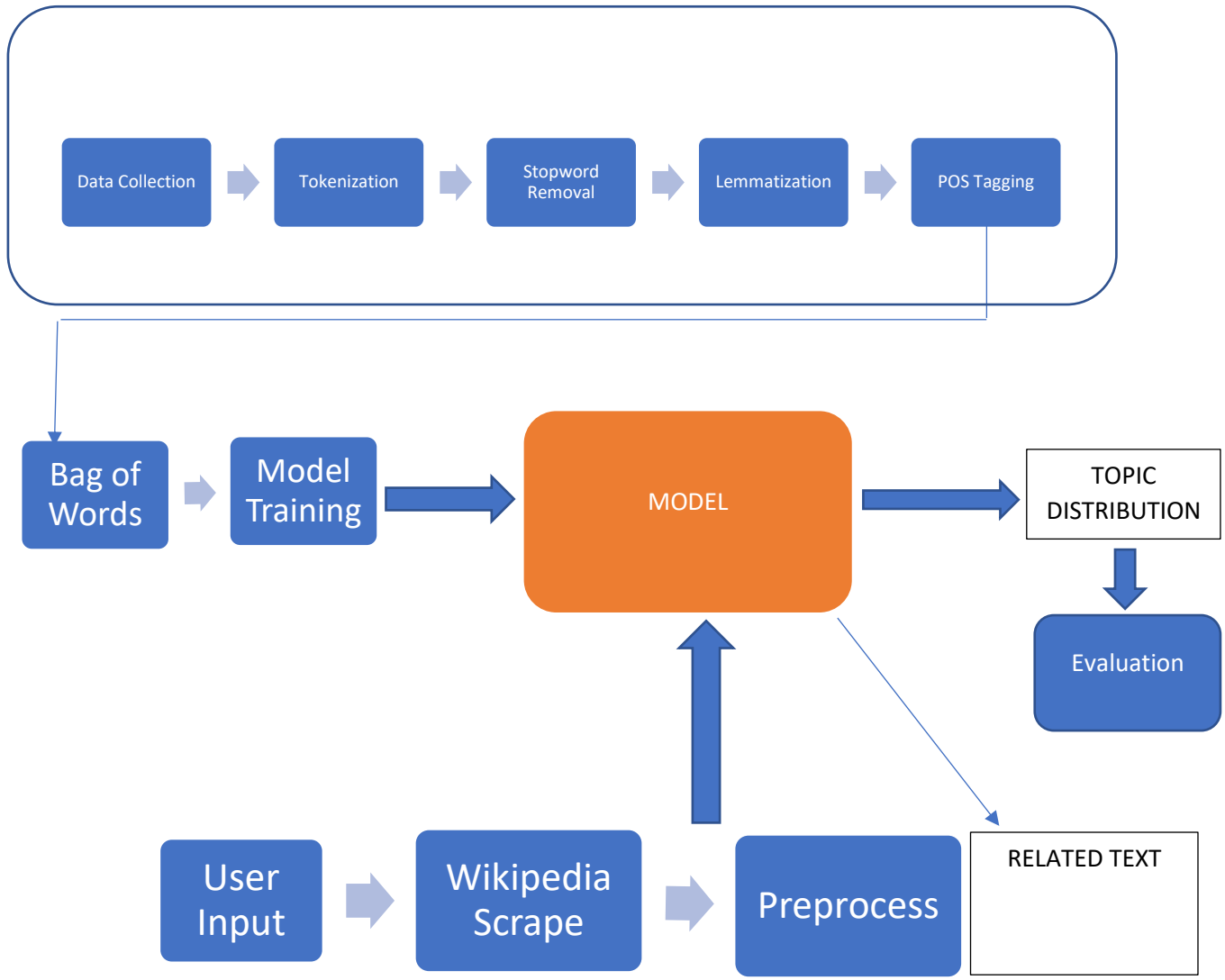
Problem Statement

There is a need to build more functional and relevant recommendation engines and algorithms not based only on explicitly mentioned features, but on hidden themes that run through a number of related documents. Thus, alternative solutions like topic modelling must be explored.

Statement: To find themes present in document corpus and use them to group and classify books.

Architecture

Pre-processing



Related Work

Topic modelling is a continuously evolving methodology with different models being created every few years. While the field is relatively sparse in research, it has given rise to some of the most prominent models in NLP. These are Latent Dirichlet Allocation (LDA) and Latent Semantic Analysis (LSA). The latter makes use of the TF-IDF (Term Frequency – Inverse Document Frequency) matrix of words and documents. It performs Singular Value Decomposition on this matrix and retains a certain number of components. Because of this decomposition, LSA is often found to have lower accuracy. The resultant matrix with lower resolution is the topic matrix, and from here, the most prominent topic of a document can be selected.

In contrast, LDA – which is employed in this case study – considers documents as probability distributions over the latent topic space, and topics to be probability distributions over words. That is, documents have probabilities of being associated with specific topics, and topics have probabilities of being associated with specific words. Thus, for a given text, a topic mixture is chosen, and then the relevant words for the topic are picked based on the multinomial distribution of the same.

One feature that has been employed in this case study is the use of POS Tagging to isolate noun words in the text, so as to make the topic distribution more coherent. This was found to be a very useful step because it contributed to human interpretability about the actual content of the topics. This method was discussed in a paper by **Martin and Johnson (10).**

There have been papers discussing hyperparameter tuning for topic modelling algorithms, because the number of passes or epochs a model runs for is in some way related to the size of the corpus. Here, because a very large corpus is used, it is important to strike the right balance.

Additionally, the References include links to implementations of recommender systems using text reviews and clustering, as well as LDA implementation in gensim and scikit-learn. These were inspirations for the implementation of my model.

Novelty of the Work

The novelty in this case study is use of web scraping to get the plot from Wikipedia for testing purposes; this has not been done in a vast majority of book recommendation implementations.

```
[106] import wikipedia
```

```
[107] query="Alice in Wonderland"
      text=""
```

```
[108] possible_titles = ['Plot','Synopsis','Plot synopsis','Plot summary',
                         'Story','Plotline','The Beginning','Summary',
                        'Content','Premise']
      flag=0
```

```
[109] try:
        page=wikipedia.WikipediaPage(query)
        print("Success")
        flag=1
      except:
        page=np.nan
        print("No such page exists")
```

```
      Success
```

```
[110] if flag:
        try:
          for title in possible_titles:
            if page.section(title) != None:
                text = [page.section(title).replace('\n','').replace("\'","")]
                print("Found the section")
        except:
          text=np.nan
          print("No section for the same. Cannot process.")
```

After scraping the plot, an index of the documents in the corpus is built with their most likely topics. Then, for the queried/scraped book, the most similar topic is found. Within that topic, the most similar document is obtained by using cosine similarity. Cosine similarity is a popular metric used in word embeddings and vector representations of text.

```
def get_most_similar(text,topic,dictionary, corpus):
  document_ids=[id for id in dictionary.keys() if dictionary[id]==topic]
  documents=[d for d in corpus]
  overall_similarities=[]
  text_string=' '.join(text)
  print("Topic:", topic)
  if len(document_ids)==0:
    for doc in documents:
      processed=nlp(doc)
      overall_similarities.append(nlp(text_string).similarity(processed))
  else:
    for id in document_ids:
      #print(documents[id])
      t=nlp(text_string)
      processed=nlp(documents[id])
      #print(type(documents[id]))
      overall_similarities.append(t.similarity(processed))
  ind=np.argmax(overall_similarities)
  return documents[ind]
```

Data and Pre-processing

In the case study, I have used two corpora – Brown and Gutenberg, both found in the NLTK library of corpora. These texts were chosen because they exhibited a wide variety of genres and hence were more likely to produce various themes.

The case study proceeded in 3 stages, with each adding on a pre-processing step so that its effect on the model could be observed.

In the first stage, simple tokenization is done using NLTK's word_tokenize module, as follows.

```
def tokenization(corpus):
  return nltk.word_tokenize(corpus.lower())
```

Additional cleaning is performed by eliminating stopwords and conforming strictly to the pattern of all alphabets.

```
def cleaning(tokens, stop):
  #print(re.match(regex,tokens[0].lower()))
  text=[word for word in tokens if word not in stop and re.match(regex,
word)]
  #text=[word for word in text if len(word)>=4]
  return text
```

In the second stage, lemmatization is performed using WordNetLemmatizer, which converts a word to its lemma.

```
from nltk.stem.wordnet import WordNetLemmatizer

def lemma(word):
  return WordNetLemmatizer().lemmatize(word)

def cleaning_lemma(tokens, stop):
  text=[word for word in tokens]
  text=[word for word in text if len(word)>4]
  text=[lemma(word) for word in text]
  return text
```

In the third stage, prior to lemmatization, the tokens are passed through a Part-of-Speech Tagger and only those tokens that are nouns but not proper nouns, are allowed to remain in the dictionary.

```
def get_pos_tags(text):
  return nltk.pos_tag(text)
```

```
def create_pipeline_pos(corpus, stopwords):
  cleaned_data=[]
  corpus_lengths=[]
  for document in corpus:
    print("\n----\nDocument Name:", document)
    step1=tokenization_new(document)
    stepx=get_pos_tags(step1)
    t=[]
    for i in stepx:
      if i[1].startswith('NN') and not i[1].startswith('NNP'):
        t.append(i[0].lower())
    #print("\nTokens:", t)
    step2=cleaning_lemma_pos(t,stopwords)
    print("\nCleaned:", step2)
    cleaned_data.append(step2)
  return cleaned_data
```

This is the final pipeline that is created. Using this, the corpus of files is flattened into a single array and all the texts are cleaned up, as well as put into a dictionary. From this dictionary, those words which are very frequent are removed, so as to not pollute the results of the model. Finally, the tokens are converted into a bag of words, mapping the ID of the word to its frequency. The data is now ready to be fed into the model.

```
dictionary_pos = corpora.Dictionary(flattened_pos)
dictionary_pos.filter_extremes(no_below=5,no_above=0.15)
corpus_lemma_pos = [dictionary_pos.doc2bow(text) for text in flattened_
pos]
```

Implementation

The case study uses a Latent Dirichlet Allocation model, which is a probabilistic model relying on Dirichlet priors. It assumes that a document comes from a mixture of topics, and a topic comes from a mixture of words. In essence, it is a mixture model.

The implementation used is gensim's LDA, as opposed to scikit-learn. This choice has mainly been due to the possibility of model evaluation using gensim's inbuilt Coherence Model, as well as because scikit-learn required an additional preprocessing step of TF-IDF vectorization, so that a document-word matrix could be fed into the model.

The results obtained differed significantly for the raw corpus, the lemmatized corpus and the POS-tagged corpus. In fact, the lemmatized corpus seemed to perform the worst. The metric used for evaluation here was **coherence score.** A coherence score measures how semantically similar the highest-probability words in a topic are. This would correspond to human interpretability of the topic.

Initial training experiments on the model with 15 passes and 15 topics, without filtering the dictionary beforehand, gave coherence scores between 30 and 35. After filtering the dictionary and choosing a random state for reproducibility, the coherence score increased to a range of the 50s, which shows that frequent words across multiple documents definitely played a role in biasing the topics that the probability distribution selected.

```
[54] from gensim.models import CoherenceModel
     # Compute Coherence Score
     coherence_model_lda_raw = CoherenceModel(model=lda, texts=flattened_raw, dictionary=dictionary_raw, coherence='c_v')
     coherence_lda_raw = coherence_model_lda_raw.get_coherence()
     print('\nCoherence Score for Raw Texts: ', coherence_lda_raw)

     Coherence Score for Raw Texts:  0.5099580316187408

[55] coherence_model_lda_lemma = CoherenceModel(model=ldamodel_lemma, texts=flattened_lemma, dictionary=dictionary_lemma, coherence='c_v')
     coherence_lda_lemma = coherence_model_lda_lemma.get_coherence()
     print('\nCoherence Score for Lemmatized Texts: ', coherence_lda_lemma)

     Coherence Score for Lemmatized Texts:  0.4827243845634896

[58] coherence_model_lda_pos = CoherenceModel(model=ldamodel_lemma_pos, texts=flattened_pos, dictionary=dictionary_pos, coherence='c_v')
     coherence_lda_pos = coherence_model_lda_pos.get_coherence()
     print('\nCoherence Score for POS Texts: ', coherence_lda_pos)

     Coherence Score for POS Texts:  0.5139277778857234
```

Conclusion and Future Enhancements

The base topic model proved to be very susceptible to the hyperparameters that were passed. Therefore, it could be concluded that significant understanding of how the model generates topics is necessary, and what values of hyperparameters are optimal for large corpora. It could also be concluded that annotations in the form of POS tags were very helpful in ensuring that the model did not get side-tracked by irrelevant vocabulary.

For future improvements to the model, I would like to explore the possibility of training it with a much larger corpus of books and creating a recommendation system which is more dynamic and flexible. Additionally, I would like to experiment with the scikit-learn implementation and see how the interpretability varies. Finally, I plan to make a web app in the future, which can visualize the results in a neat and presentable format.

Current Link to the IPython Notebook:

1. Martin, Fiona, and Mark Johnson. "More efficient topic modelling through a noun only approach." *Proceedings of the Australasian Language Technology Association Workshop 2015*. 2015.
2. https://www.garysieling.com/blog/part-of-speech-tagging-nltk-vs-stanford-nlp/
3. https://humboldt-wi.github.io/blog/research/information_systems_1819/is_lda_final/
4. https://towardsdatascience.com/topic-modelling-in-python-with-nltk-and-gensim-4ef03213cd21
5. https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0
6. https://spacy.io/usage/vectors-similarity
7. https://towardsdatascience.com/an-nlp-view-on-holiday-movies-part-i-topic-modeling-using-gensim-and-sklearn-5f7c15c8a65a