Task Report

-Anjali Ragupathi
Github Upload Date: 10th January 2022

Objective:

To create a supervised topic classifier which has been trained on / learned from a set of raw, unlabelled documents.

Dataset Details:

- CSV file : Each line represents a document.
- Text file : Each point contains a topic name and its description. These topics make a list of Provided Topics, based on which we can get the topic number.

Major Steps of Implementation:

1. Data cleaning
2. Exploratory Data Analysis
3. Compiling vocabulary
4. Generating unsupervised topic model using Latent Dirichlet Allocation, GSDMM, and Contextualized Topic Models
5. Visualizing topics
6. Fine-tuning and hyperparameter tuning
7. Finding out topic distribution for each document
8. Tagging each document with topics above certain threshold
9. Training classifier based on split of training and testing

Data Cleaning Steps:

The initial phase of data cleaning is the most important one as this determines what data is ultimately used for the generation of topics. The major pre-processing and cleaning steps used in this implementation are the following:

1. Lowercasing
   This is a common text normalization step to ensure that words at the beginning of a sentence (starting with a capital letter) and the same words elsewhere are treated in the same manner. This process also helps avoid redundant phrases and words when building a vocabulary.
2. Removing spaces, bad punctuation, and converting to ASCII
   This step is done to ensure there are no special characters like emojis being used here, and that no punctuation (except % which is helpful for discounts) will be used.
3. Part-of-Speech Tagging
   This step is used to refine the words that go into the topic model. The most descriptive words for a topic often turn out to be nouns, verbs, and adjectives, as they are part of an

open class. Open class categories of speech are generally the most unique as they are not fixed or limited by existing grammar rules, and can be extended to describe a situation better. In our case, since we want words that describe a topic, we can choose nouns (which include words like 'garage', 'service', 'fitter', 'tyre'), verbs (like 'book', 'service', 'deliver'), and adjectives (like 'expensive', 'cheap', 'poor', 'great'). Having explored using other categories like adverbs and cardinal numbers, it was found that these categories did not significantly contribute to the final result, and so these were omitted. For the sake of a study, I looked at three separate cases of Verbs, Nouns, and Adjectives each being kept, while the other POS tags were discarded. This was done to make the topic allocation easier in the later steps, as will be explained. Finally, I decided on keeping two cases: one with only nouns, and one with nouns, verbs, and adjectives together. The dictionary was adjusted accordingly.

4. Replacing contractions
   In order to simplify tokenization and to separate negations from their contracted form, it was necessary to replace certain common contractions like "didn't", "couldn't", and so on. Some grammatical variations of words like "tire" were standardized to one form, say "tyre".
5. Removing stopwords
   Stopwords include words that typically do not add any specific information beyond grammar, to the text. These may be conjunctions, modals, frequently used terms, articles, and so on. In our case, the term 'REDACTED' could also be considered a stopword as it mainly referred to the company, but did not add any extrinsic meaning. However, we have chosen to keep the word 'REDACTED'.
6. Lemmatization
   To avoid redundancy in terms of tense, subject-verb agreement, and so on, lemmatization was found to be necessary to reduce the remaining words into their lemma forms. This approach is not perfect and can be tuned to ignore proper nouns, so that proper nouns can remain the same without being subject to this step. Stemming was not preferred because of its tendency to simply chop off the endings of words.
7. Replacing multi-words
   Some reviews had multiple words that were actually single words, but had been split up due to fast typing or grammatical issues. I have manually replaced those as needed.
8. Bigrams and trigrams
   This step was an important part for generating phrase clusters. Bigrams and trigrams contribute to collecting frequently co-occurring words, for example, "great service" or "local tyre fitters". The complexity for this step would have been O(nl) where n is the number of reviews, and l is the number of words in each review. I used collocation to generate bigrams and trigrams. This method used a Pointwise Mutual Information (PMI) score, which calculates the log likelihood of a set of words occurring together in a joint

distribution over them occurring independently. I filtered the list of bigrams and trigrams and set a threshold at the point where the n-grams stopped making sense.

Choice of Topic Modeling Approach:

The reasons I selected Latent Dirichlet Allocation over other topic modeling approaches like Latent Semantic Analysis and Non-Negative Matrix Factorization were that firstly, LDA could be visualized easily in terms of inter-topic distance and per-topic keywords, and secondly, my reading about other forms of topic modeling showed that LDA was the most consistent with a higher number of topics. Additionally, for NMF and LSA, a document*feature matrix would have to be constructed explicitly, which would add to memory consumption for large datasets. Since our dataset has over 10000 records, it did not make sense to use these methods.

Also, it was pre-determined that there were to be 12 topics; therefore, I thought it would be better to use simple LDA and not its hierarchical, non-parametric counterpart.

Another reason why I chose LDA was because the end goal of the task was to build a multi-label topic classifier. This would be easier to construct if the topic model could give a probabilistic representation of each document, such that it was possible to accommodate multiple topics. Thus, since each document was a mixture of topics, and each topic was a mixture of words, the LDA model could predict multiple topics per document, which would have been slightly trickier with LSA and NMF.

Based on this paper, I also used a Gibbs Sampling Dirichlet Mixture Model, as it was cited to be effective for short texts, such as these reviews. I found an implementation here and trained the GSDMM model on the set of texts. However, the issue with this method was that it was difficult to determine multiple useful topics per document with the inbuilt method. Custom methods would have to be designed for this.

I also experimented with using contextualized topic models which could include context into the generation of the bag-of-words using contextual embeddings. However, this approach proved to be difficult with the final topic refinement.

Parameter selection:

A major point was to choose the number of iterations (maximum number of iterations on a document till convergence) and the number of passes (through the whole dataset). Using a simple for-loop, a grid search based approach was implemented, with a complexity of O(nip), where n is the length of the dataset, i is the length of the iteration parameter array, and p is the length of the passes parameter array. This could be a constant value in the case of this experiment, but would vary to quadratic complexity with the number of parameter values being passed. This high complexity is also because the model would have to be trained with the entire dataset but different parameters each time.

Metric of choice:

The choice of a suitable coherence measure while evaluating the coherence score was a challenging task. Initially, I used the c_umass coherence measure as it was the most cited and was based on co-occurrence of words. With this measure, I found that the best parameters were 100 iterations and 40 passes, with a chunk size of 5000 for memory optimization. I also tested the c_v measure as it was touted to be better than u_mass, but this gave different results, and the combination obtained here was highly variable every time the code was run. Finally, I settled on using c_uci because it looked at external semantic viability of the keywords in a topic, i,e. It looked at the log likelihood of two words in a topic occurring together overall within a joint distribution, as opposed to independently occurring. With this metric, I used a hyperparameter search method similar to GridSearch to find the best iterations and passes.

For GSDMM, I opted for 16 iterations over the dataset with a filtered dictionary having words that were in more than 20 documents and in less than 12% of the documents, so as to create pure topics. I used the major part of speech tags for preprocessing this data instead of simply nouns.

Labeling topics:

Given the word and phrase clusters for each topic, it was necessary to match each of the generated topics to the list of provided topics. This was a challenge because of the high number of overlapping and unnecessary keywords that had not been caught during the cleaning stage of the pipeline. I attempted to create a dictionary mapping of the topics to the keywords; however, these changed each time the models were trained, and the topic labels were shuffled.

Annotating documents:

This was the stage where the dataset had to be augmented with the labels that would further be used to train a supervised classifier. This was done by setting a threshold for which the most prominent topics per document would be selected. Then, each document was processed iteratively, and each of the 12 topics was given a probability score according to the topic mixture associated with it. The topics that had a probability greater than the threshold were added as class labels to the document. I made sure that the topics were either assigned with LDA or with GSDMM, and dropped the remaining docs.

Topic classifier approach:

For a multi-label classification problem, I reviewed multiple approaches including the binary all-vs-one classification system, the multi-class conversion approach, and the ensemble method with voting. Out of these, I found that the multi-class method would create a very high space and time complexity, because the final layer would have to have all possible combinations of selected classes, i.e. each possible combination of categories/topics would be treated as a class label and then the final layer of the model would simply be treated the same as a multi-class, single output classification. Ensemble methods would also be costly to implement based on the classifiers

involved, and may have some degree of randomness. Thus, for simplicity, I decided to use the binary all-vs-one method.
For vectorization, I used the TF-IDF Vectorizer supplied by scikit-learn, as it showed better results over simple bag-of-words models. However, it is to be noted that word embeddings may show better results in terms of semantic correctness.
I created 12 binary classifiers of the same kind. I experimented with using Multinomial Naive Bayes (which was documented to be useful for text classification problems), SVM, and Logistic Regression, and found that Logistics Regression (the simplest classification choice) had the highest F1 score and accuracy with classification.
Multinomial Naive Bayes proved to have better results than Gaussian Naive Bayes. Additionally, SVM took a much longer time to run and required standard scaling to be applied.

Limitations and Future Improvements:

In my approach, I have refrained from using deep learning methods such as an artificial neural network, as I wanted to stick to the basics. However, for much larger datasets, it would be prudent to use a deep neural net with 12 neurons in the output layer.
Additionally, refined POS tagging, parsing, and further preprocessing like spelling correction, et al. can be done to make the data cleaner. Finding an optimal minimum and maximum frequency of words to be used in the final dictionary would also help in generating well-separated topic clusters.
Finally, a pipeline ought to be created to connect an interface (where the review is entered) to the backend, where the topic classifier would receive input, along with intermediate preprocessing (using the create_pipeline function), scaling, and vectorization steps.
For the topic modeling approach itself, while LDA and GSDMM have certainly brought about fairly good results, it may be worth looking into versions of non-negative matrix factorization, hierarchical LDA, and dense topic models that use Transformers (such as BERT) to see improvements in the results.

Key References

https://medium.com/analytics-vidhya/an-nlp-approach-to-mining-online-reviews-using-topic-modeling-with-python-codes-9d766e783003

https://towardsdatascience.com/topic-modeling-articles-with-nmf-8c6b2a227a45

https://towardsdatascience.com/topic-modeling-for-the-new-york-times-news-dataset-1f643e15caac

https://medium.com/analytics-vidhya/topic-modeling-using-lda-and-gibbs-sampling-explained-49d49b3d1045

https://www.nltk.org/_modules/nltk/tokenize/mwe.html

https://www.machinelearningplus.com/nlp/lemmatization-examples-python/#wordnetlemmatizer

https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/#13viewthetopicsinldamodel

https://towardsdatascience.com/6-tips-to-optimize-an-nlp-topic-model-for-interpretability-20742f3047e2

https://people.cs.umass.edu/~mccallum/papers/tng-icdm07.pdf

https://towardsdatascience.com/evaluate-topic-model-in-python-latent-dirichlet-allocation-lda-7d57484bb5d0

https://stackoverflow.com/questions/50805556/understanding-parameters-in-gensim-lda-model

https://www.os3.nl/_media/2017-2018/courses/rp2/p76_report.pdf

https://towardsdatascience.com/short-text-topic-modeling-70e50a57c883

https://towardsdatascience.com/feature-engineering-with-nltk-for-nlp-and-python-82f493a937a0

http://ikee.lib.auth.gr/record/324006/files/Dimitriadis-2158.pdf