Topic Modeling on BBC News

Arvind Krishna, Jayesh Panchal,
Keshav Sharma, Sahil Ahuja
Data Science trainees
AlmaBetter, Bangalore

Abstract:

BBC stands for British Broadcasting Corporation. It is an operational business division of the British Broadcasting Corporation (BBC) responsible for the gathering and broadcasting of news and current affairs in the UK and around the world. The department is the world's largest broadcast news organization and generates about 120 hours of radio and television output each day, as well as online news coverage.

The service maintains 50 foreign news bureaus with more than 250 correspondents around the world. BBC News Online is the BBC's news website. It is one of the most popular news websites in the UK, reaching over a quarter of the UK's internet users, and worldwide, with around 14 million global readers every month.

The website contains international news coverage as well as articles based on entertainment, sport, science, and political news.

Keywords: BBC news, Unsupervised Machine Learning, Topic Modeling, LDA, NLP

1. Problem Statement

The Data provided is divided into 5 categories, that are, Sports, Tech, Business, Entertainment and Politics. Across these folders the data is stored in form of text files such that each text file contains the complete transcript of the article. The dataset in this case isn't collective, it has been stored in form of numerous text files sub-categorized in 5 different domains. The first task is to segregate the data which can be achieved by traversing all these folders and storing the data in the files to a data-frame.

For loading the data we'll visit all text files individually and copy all articles to a data-frame along with their category.

In this project the main objective is to identify major themes/topics across a collection of BBC news articles.

- Index: Entry index.
- Filename: Destination File name/number.
- Contents: Complete transcript of a article, this contains all the textual data present in the destination file for a particular entry.
- Category: Theme/domain of a article.

2. Introduction

We'll start by segregating the data from all sources into one data-frame and then we'll apply some vectorization over the data such that it could be passed on to a Machine learning model. In order to use textual data for predictive modeling, the text must be parsed to remove certain words – this process is called **Tokenization**.

These words then need to be encoded as integers, or floating-point values, such that they can be used as inputs in machine learning algorithms. This process is called **Feature Extraction (or Vectorization)**.

After finishing up with Vectorization we can move forward for model development. In this project we'll use LDA (Latent Dirichlet Allocation) algorithm for topic modeling. We'll represent our findings using Word-cloud and LDAVis.

3. Steps Involved

- Loading and Discovery (Exploratory Data Analysis)
- Stemming (Lemmatization)
- Feature Extraction
- Model Development
- Learning outcomes
- Conclusions

4.1. Loading and Discovery (Exploratory Data Analysis)

The dataset contains a set of news articles for each major segment consisting of business, entertainment, politics, sports and technology.

There are over 2000 news article available in these categories. The dataset in this case isn't collective, it has been stored in form of numerous text files sub-categorized in 5 different domains. The first task is to segregate the data which can be achieved by traversing all these folders and storing the data in the files to a data-frame.

Note:- There was a particular file in this dataset which was formatted differently from the rest of the files, hence while reading the data, the text formatting was throwing an exception. For such case we have neglected the file altogether, since the case was relevant to only a single file.

Treating Duplicates

Now, we checked if a particular article is being stored more than once into the data-frame. Having duplicates in our data would cause inconsistencies in results.

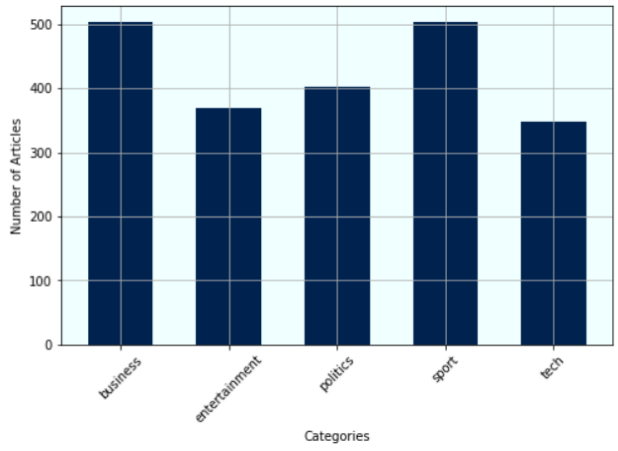
We found out that there are around 99 duplicate rows in our dataset, it resembles that these articles are present in more than one category, we will be dropping such rows before we proceed further with the model development. We also dealt with outliers like having a empty string as contents etc.

Categories

In total we had 5 categories in which the data was distributed. Each of these categories represents a particular domain on which the articles were based on. Following are the categories for BBC news articles:

- Sports
- Business
- Tech
- Entertainment
- Politics

The distribution of data along these categories (Number of articles) is portrayed by the following graph:-



We found that Business and Sports category have the highest number of articles, while for Tech category the number is quite low.

The number of articles will play a major role in determining the topics along with a particular category. Larger the number of articles for a category, higher will be the result accuracy for the model. This is obvious because having more data helps the model to learn better and provide stellar results.

Contents

Contents were the most important feature for our model since it contains the whole transcript of the news article. First we extracted the attribute from the data and then applied CountVectorizer to get the top words from the data based on their frequencies.

- **Removing Stop words (NLTK):**

  Before applying CountVectorizer to get the most recurring words we had to get rid of all stop words from our data, such that the results would not contain filler words or helping words.

  Stop Words: A stop word is a commonly used word (such as “the”, “a”, “an”, “in”). Since these words occur a lot, a search engine has to be programmed to ignore them. We can easily remove them by storing a list of words that we consider to be stop words.

  For removing these stop words, we implemented **NLTK** (Natural Language Toolkit). NLTK in python has a list of stop words stored in 16 different languages. We've use it to extract our data and getting rid of such words. We know that all of our articles were in English language;

hence we only required stop words in English.

Now that the stop words were removed from the data, we implemented our CountVectorizer to extract top words with highest frequencies.

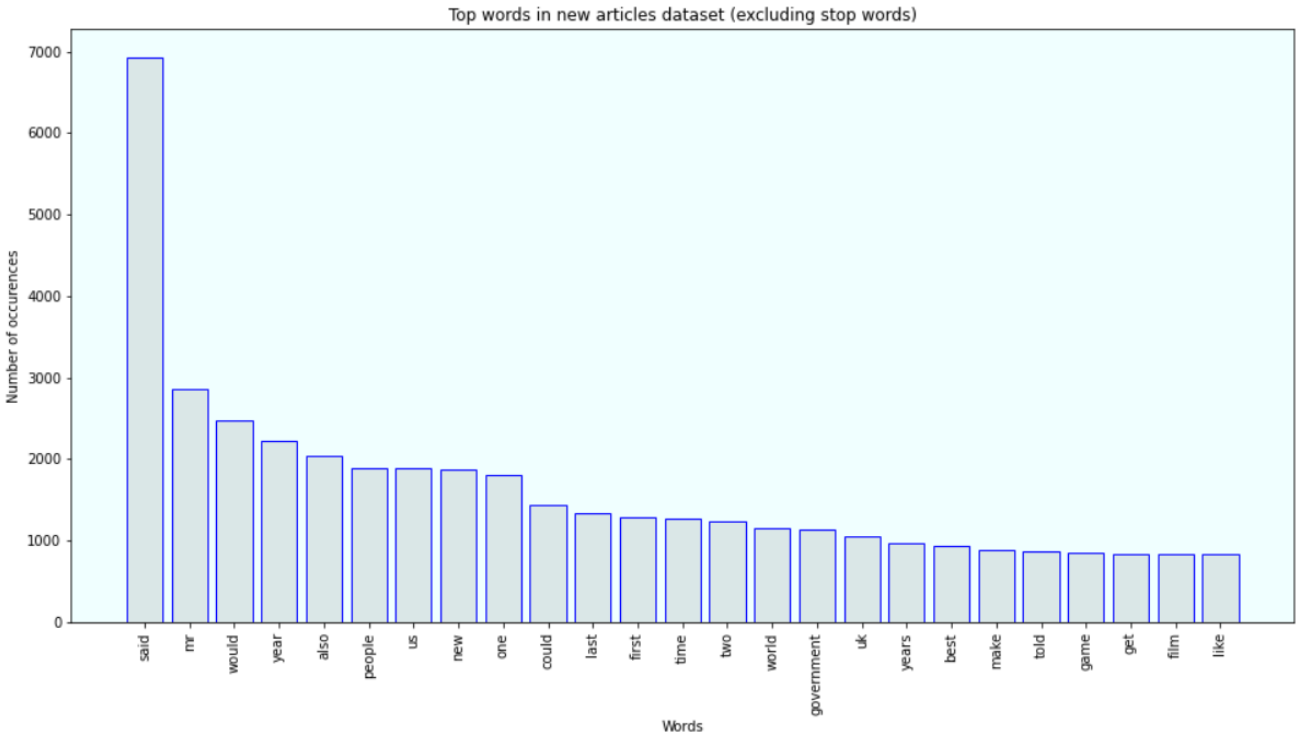
- **CountVectorizer:** In order to use textual data for predictive modeling, the text must be parsed to remove certain words – this process is called **Tokenization**.

  These words then need to be encoded as integers, or floating-point values, such that they can be used as inputs in machine learning algorithms. This process is called **Feature Extraction (or Vectorization)**.

  Scikit-learn's **CountVectorizer** is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

  CountVectorizer creates a matrix in which each unique word is represented by a column of the matrix, and each text sample from the document is a row in the matrix. The value of each cell is nothing but the count of the word in that particular text sample.

Below is the bar-plot representing the top 25 words found throughout the data, based on their frequencies.



From the above graph, we saw that some commonly used words like "said", "Mr", "would" etc, are on top, which makes sense because no matter what the subject of a article is, these words help build up a sentence and hence the excessive use is inevitable.

However, we can also found a few words relevant to the categories of our articles. For example, '**UK**' and '**US**' (since we converted all words to lower case, 'us' represents US too) both are country names which could reflect to any of the categories. Also words like '**government**' could reflect in articles related to Politics, Business, etc.

Words like '**film**' and '**game**' sounds relevant to the entertainment category.

4.2. Stemming (Lemmatization)

Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Stemming is important in natural language understanding (NLU) and natural language processing (NLP).

Lemmatization: This algorithm collects all inflected forms of a word in order to break them down to their root dictionary form or lemma. Words are broken down into a part of speech (the categories of word types) by way of the rules of grammar.

First we used the **WordNet** Lemmatizer on our data. The lemmatizer seemed to be working fine; the words in the response were segregated into a list. But the issue was for some words the lemmatizer returned ambiguous results.

For example, the first line of an article said "A US government", here the lemmatizer reduced the text "US" to just "u". This takes away the whole meaning of that word.

Since the results for WordNet lemmatizer weren't satisfactory. We implemented the **Snowball** lemmatizer. Again the lemmatizer does its job and we got the list of words but similar to WordNet, some words lost their meaning completely in the response.

For example, "government" is returned as "govern", this is acceptable to some extent but we also have words like 'accus' and 'countri' derived from accusing and companies respectively. These words do not make any sense at all. This lemmatizer worked much more aggressively and hence we encountered more such words. Still the overall performance was slightly better than WordNet.

We also used **TextBlob** lemmatizer and similar issued were encountered. The results are mostly similar to that of WordNet lemmatizer.

Even after implementing all these lemmatizers, the results for any of them weren't conclusive and hence we didn't use any lemmatization on data.

This part remained experimental since the results were ambiguous and non-conclusive.

4.3. Vectorization (Feature Extraction)

Vectorization is jargon for a classic approach of converting input data from its raw format (i.e. Text) into vectors of real numbers which is the format that ML models support.

The idea is to get some distinct features out of the text for the model to train on, by converting text to numerical vectors. We applied CountVectorizer to convert words of articles into numeric form such that we can train our model on them.

4.4. Model Development

Now that all the necessary preprocessing was completed, we implemented the ML model for topic modeling on the BBC news articles. Here we have used the LDA algorithm for the purpose. Let's discuss more about the algorithm and its specifics.

- **Latent Dirichlet Allocation (LDA)**:

  It is one of most popular topic modeling technique to extract topics from a given corpus. The term latent conveys something that exists but is not yet developed. In other words, latent means hidden or concealed. The Dirichlet model describes the pattern of the words that are repeating together, occurring frequently, and these words are similar to each other.

This stochastic process uses Bayesian inferences for explaining "the prior knowledge about the distribution of random variables". In the case of topic modeling, the process helps in estimating what are the chances of the words, which are spread over the document, will occur again? This enables the model to build data points, estimate probabilities, that's why LDA is a breed of generative probabilistic model. LDA generates probabilities for the words using which the topics are formed and eventually the topics are classified into documents.

The LDA makes two key assumptions:

- Documents are a mixture of topics, and
- Topics are a mixture of tokens (or words)

The end goal of LDA is to find the most optimal representation of the Document-Topic matrix and the Topic-Word matrix to find the most optimized Document-Topic distribution and Topic-Word distribution.

As LDA assumes that documents are a mixture of topics and topics is a mixture of words so LDA backtracks from the document level to identify which topics would have generated these documents and which words would have generated those topics.

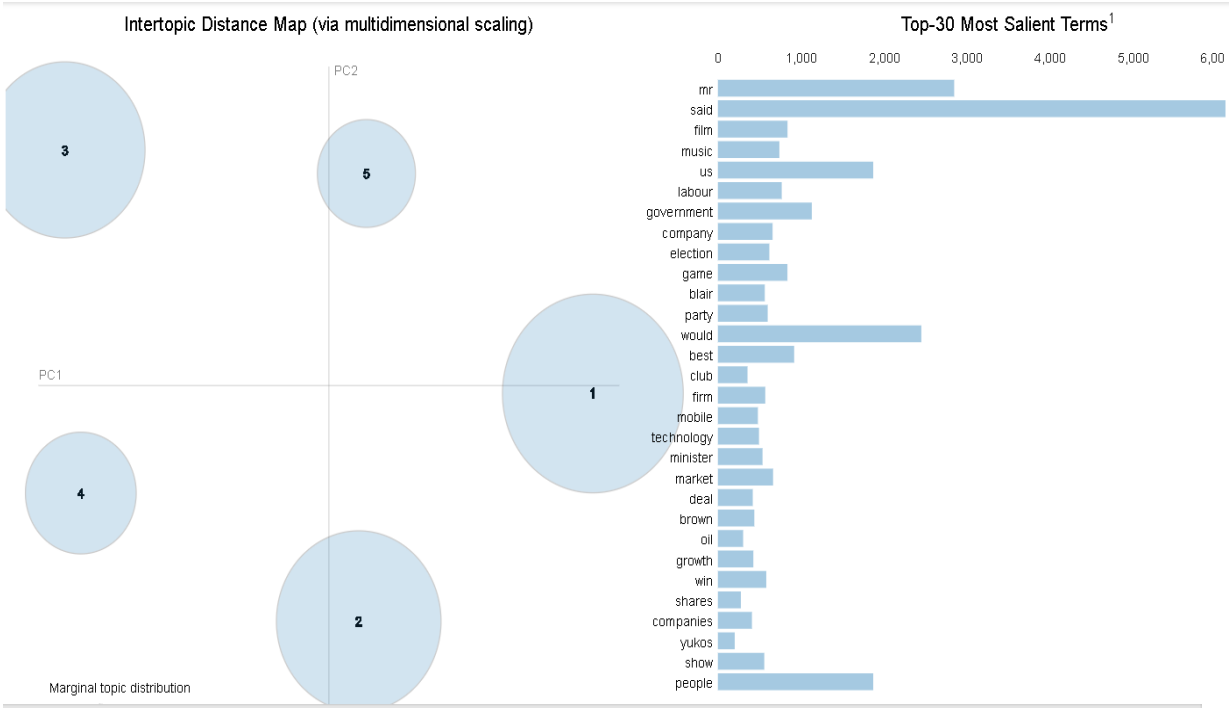
4.5. Learning Outcomes

LDAVis

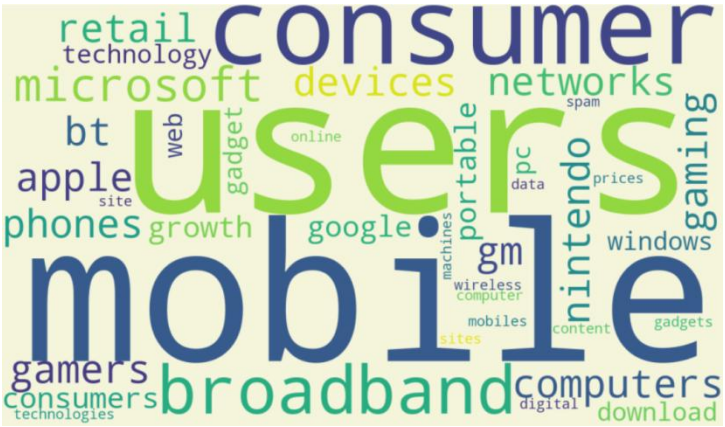
We have used LDAVis to represent the findings of our model. The graph below displays the top 30 terms for each tags/keywords in our data.

This LDAVis also represents the set of words most relevant to each category.

Each term in here is extracted from a respective article set and based on their relevancy they're sorted out. Top 30 terms resembles keywords that were encountered the most throughout that category. If we just hover over any of these circles we can see the terms related to that particular category.



After extracting these words we then segregated all the terms with their respective frequencies belonging to each topic/category in a new DataFrame (containing top 50 keywords for every category. We used this data to develop word -clouds for each topic to represent all relevant terms for that domain.



Above is the word cloud for Topic 1 (**Tech**). We can see lot of terms related to the tech category in this word cloud. For example; **Mobile**, **Technology**, **Broadband** and many more.

We can also see some tags like gamers, gaming, Nintendo, etc, which direct towards the **Gaming** industry in general. Also we have major tech giants like **Apple**, **Google**, and **Microsoft**. These are some of most famous tech companies.

The results seem fairly accurate for the category.

4.6. Conclusions

1. While reading the text files, we noticed that the file encoding was different in a few off-cases. We found that considering such factors, and engineering based on such knowledge, is very important while handling such data, in order to do so efficiently.

2. Upon experimenting with stemming and lemmatization on our dataset, we found that although it saves space and perhaps time, in our case, it's better to focus on quality, and avoid nuances. In our own 'cost-benefit' analysis, the difference weren't all that significant. Perhaps at a massive scale, the former approach would be ideal.

3. We noticed that it's more optimal to tokenize with no factual differences. so we lowercased the contents to unify tokens that may have just case-differences.

4. These are the optimal LDA metrics that we got after implementing **GridSearchCV**:

    - Best LDA model's parameter {'n_components': 5}
    - Best log likelihood Score for the LDA model - 643494.9704171557
    - LDA model Perplexity on train data 1696.6352006244963

5. Upon looking at the top ‘n’ topics generated, we were able to correlate it with relevance to what was expected at a significant degree, whilst also shedding light on some unseen aspects.. Hence, we see that the model effectively beared fruit.
