**Problem Description:** In today's world, we're flooded with tons of digital information, especially from news sources. Sorting through all of it can be overwhelming. That's where topic modeling comes in. It's a way to automatically organize and understand large amounts of text by figuring out the main themes or topics in a bunch of documents. So, instead of reading everything, we can let the computer do the heavy lifting and find the important stuff for us.

The BBC is a big and well-known news organization that writes about a lot of different topics every day. By using a technique called topic modeling, we can look at all the articles they write and figure out what main themes or trends they cover the most. But doing this isn't easy. One of the main problems is using a specific method called Latent Dirichlet Allocation (LDA) to analyze all the articles. LDA tries to find patterns in the words used in the articles to understand what they're about. One big challenge is making sure LDA can accurately understand what the articles are about. But it's tough because it involves a lot of math and computer work to make sure the analysis is accurate.

News articles are always changing because they reflect what's happening in the world right now. This makes it tricky to use topic modeling to figure out what they're about. Also, understanding what the topics mean and whether they make sense is not easy. Even though we use a method like LDA to help, it still requires a lot of judgment and careful thinking. To make sense of BBC News articles using topic modeling, we need to really understand how LDA works and how to use it with real news data. If we can overcome these challenges, we can learn a lot about what the BBC covers and how they do it, which could help with analyzing news and finding information.

**Introduction to LDA:** Latent Dirichlet Allocation (LDA) is a probabilistic generative model used for topic modeling, which is a statistical technique for uncovering hidden thematic structures within a collection of documents. Latent Dirichlet Allocation (LDA) is like a tool that helps find topics in a bunch of documents. It assumes each document is made up of a few hidden topics and each topic has its own set of words. LDA tries to figure out these hidden topics and what words they're made of by looking at the words in the documents.

**Initialization:** Before using LDA (Latent Dirichlet Allocation), we have to decide how many topics we want to extract, and we need a collection of documents where each document is represented as a bag of words (just a list of words without any particular order).

**Topic Assignment and Iterative Inference:** During the initialization step, every word in every document is randomly assigned to one of the predefined K topics, kickstarting the process of assigning words to topics. In LDA, words are assigned to topics based on data and adjusted over iterations. Each iteration tweaks assignments to better match the data.

**Gibbs Sampling:** In Gibbs sampling for Latent Dirichlet Allocation (LDA), we're basically shuffling around words in documents to find the best fit of topics. We do this by repeatedly picking a word, guessing which topic it belongs to based on the topics of other words in the document, and adjusting our guesses to match the overall topic distribution. This helps us figure out which topics are most likely associated with which words in our documents.

**Notation:**

* D = Number of documents in the corpus.
* ​ = Number of words in document d.
* K = Number of topics.
* V = Size of the vocabulary (number of unique words in the corpus).
* = Number of words assigned to topic k in document d.
* = Number of times word v is assigned to topic k in all documents.

**Generative Process:**

* + For each document d ∈{1,2,...,D}:
    - Choose a distribution over topics ∼Dirichlet()
    - For each word w in document d:
      * Choose a topic z∼Multinomial().
      * Choose a word w from the topic's multinomial distribution p(w|z,β), where β is the parameter controlling the topic-word distribution.

**Probability Distributions:**

* + Dirichlet Distribution: Denoted as Dirichlet(), it is a multivariate generalization of the beta distribution. It generates probability distributions over a simplex (topic proportions for documents).
    - ()∼ Dirichlet() represents the distribution of topics for document d.
  + Multinomial Distribution: Denoted as Multinomial(), it models the probability of observing a particular word given a topic or document-topic distribution.
    - z∼Multinomial() represents the selection of a topic for a word in document d.
    - w∼Multinomial( represents the selection of a word from the topic's multinomial distribution.

**Parameter Estimation:** LDA aims to estimate two sets of parameters:

* + - Document-Topic Distribution: () for each document d. The distribution of topics within each document, which represents the proportion of each topic present in the document.
    - Topic-Word Distribution: ( for each topic k. The distribution of words within each topic, which represents the probability of each word occurring in that topic.

This estimation is typically done using variational inference or Gibbs sampling.

**Interpretation:** After analyzing data to estimate parameters, we can interpret the resulting distributions of topics across words and documents to understand the main themes in the corpus. Each topic is described by a list of words it represents, helping users identify key themes, while each document is represented by the prevalence of different topics it contains.

**Evaluation:** We can judge the effectiveness of the identified topics by looking at metrics like coherence, which checks how closely related the main words in a topic are, and interpretability, which gauges how understandable the topics are to people.

**Loading and Analyzing the Data:** First, we'll download some datasets, and each dataset will be a folder with a few different files inside. We'll use variables called “bbc\_folder” and “bbcsports\_folder” to keep track of where these folders are on our computer. Each folder has three important files: one with a .mtx extension, another with a .terms extension, and the last one with a .docs extension. The .mtx file contains a matrix where rows represent terms found in articles and columns represent the articles themselves. This matrix is in a special format called the Matrix Market format. The .terms files contain the actual terms found in the articles, with one term per line. These are the row names of the matrix. Similarly, the .docs files contain the names of the articles, with one name per line. These are the column names of the matrix. Luckily, the articles in these datasets have already been processed, meaning they've been simplified by removing common words (stop words) and terms that don't appear often.

A screenshot of a computer program

Description automatically generated

To bring data from a file in Matrix Market format into R, we use the “readMM()” function from the Matrix package. This function loads the data and saves it as a sparse matrix object. To make it compatible with the tm package, we convert this sparse matrix into a term document matrix using the “as.TermDocumentMatrix()” function. When using this function, we need to specify a weighting parameter. This parameter tells us what the numbers in the original matrix represent. Since we have raw term frequencies, we set the parameter to “weightTf”.

A computer code with black text

Description automatically generated

A computer screen shot of a computer code

Description automatically generated

Next, we'll load the terms and document identifiers from the remaining files. We'll use the scan() function to read these files, which have one entry per line, and store them in vectors. Once we have these vectors containing terms and document identifiers, we'll use them to update the row and column names of the term document matrix. Finally, we'll transpose this matrix into a document term matrix, which is the format needed for further analysis.

A black text on a white background

Description automatically generatedA screenshot of a computer

Description automatically generated

The output represents a document term matrix containing 2225 documents and 9635 unique terms. Among the entries in the matrix, 286,774 are non-zero, while the majority, 21,151,101, are zero, resulting in a sparsity of 99%. This high sparsity indicates that most terms do not appear in most documents. The longest term in the dataset consists of 24 characters. The matrix is weighted using term frequency (tf), meaning that the numbers in the matrix indicate how frequently each term appears in each document, with higher values indicating higher frequencies.

A computer code with text

Description automatically generated with medium confidence

A black text on a white background

Description automatically generated

The output describes a document term matrix with 737 documents and 4613 unique terms. Among the entries, there are 85,576 non-zero values, while the remaining 3,314,205 are zero, resulting in a sparsity of 97%. This high sparsity suggests that most terms do not appear frequently across documents. The longest term in the dataset contains 17 characters. The matrix is weighted based on term frequency (tf), indicating that the numbers in the matrix represent how often each term appears in each document, with higher values indicating greater frequency.

Now that we have prepared the document term matrices for both datasets, we notice that the BBC dataset has approximately twice as many terms as the BBCSports dataset, and the BBCSports dataset also has about a third of the number of documents, making it much smaller. Before we proceed to build our topic models, we need to create vectors that contain the original topic classification of the articles. By examining the document IDs, we observe that each identifier follows the format of "<topic>.<counter>".

To create a vector with the correct topic assignments, we just need to remove the last four characters from each entry, which represent the counter. Once we've done this, we can convert the resulting values into a factor. This allows us to easily see how many documents we have for each topic.

A computer code with black text

Description automatically generated

A screen shot of a computer

Description automatically generated

The output provides a breakdown of the number of documents categorized under each topic and subtopic. Specifically, there are 510 documents related to business, 386 to entertainment, 417 to politics, 511 to sport, and 401 to tech. Within the sport category, there are further subtopics: 101 documents on athletics, 124 on cricket, 265 on football, 147 on rugby, and 100 on tennis. This breakdown helps to understand the distribution of articles across different subjects in the dataset, facilitating further analysis and exploration of the data.

We're going to use the package "topicmodels" to build topic models for our two datasets. This package lets us use data structures created with the "tm" package for topic modeling. For each dataset, we'll build four different types of topic models:

LDA\_VEM: An LDA model trained using the Variational Expectation Maximization (VEM) method, which automatically estimates the Dirichlet parameter vector.

LDA\_VEM\_: An LDA model trained with VEM, but without estimating the α Dirichlet parameter vector.

LDA\_GIB: An LDA model trained with Gibbs sampling.

CTM\_VEM: A Correlated Topic Model (CTM) model trained with VEM.

To train an LDA model, we'll use the "LDA()" function, specifying parameters like the document term matrix, the number of topics (k), and the training method (VEM or Gibbs). For the LDA\_GIB model, we explicitly specify Gibbs sampling as the method. We'll also use a control parameter to set parameters affecting the fitting process, such as the random seed for reproducibility, and parameters for the Gibbs sampling procedure if needed. For the CTM model, we'll use the "CTM()" function with similar syntax to the LDA function.

We made a function that makes four trained models. We give it a document term matrix, the number of topics we want, and a seed. The function uses standard values for training.   
A computer screen shot of a program

Description automatically generated

To evaluate how well the topic models have performed, we'll check if the five topics generated by each model match the five topics originally assigned to the articles. We can do this by using the “topics()” function with the model, which gives us the most probable topic for each document. By default, it returns the top topic, but we only need one in this case. Once we have the predicted topics, we'll compare them with the labeled topics to see how they align. From the output, we can see that topic 5 mostly represents sports articles. Topics 4 and 3 align with politics and business categories respectively. However, models 1 and 2 have a mix of entertainment and technology articles, failing to clearly distinguish between the desired categories. Ideally, each model topic should match with one correct (or labeled) topic, which we refer to as the "gold standard."

A computer code with black text

Description automatically generated

We intuitively believe that this topic model is a better fit for our original topics compared to the first one because each model topic mostly selects articles from just one of our intended topics.

A rough way to measure how well the topic model matches our target topics is by looking at the highest value in each row. This highest value represents the gold topic assigned to the model topic in that row. Then, we calculate the total accuracy by dividing the sum of these maximum row values by the total number of documents. For example, for the LDA\_GIB model, this calculation would be (471 + 506 + 371 + 399 + 364) / 2225 = 2111 / 2225 = 94.9 percent. For LDA\_VEM the calculation would be (176+202+484+403+507)/2225=1772/2225 = 79.64 percent.

A computer code with black text

Description automatically generated

For the BBC dataset, the LDA\_GIB model performs much better than the others, while the CTM\_VEM model performs notably worse compared to the LDA models. In the BBCSports dataset, all models perform similarly, but the LDA\_VEM model slightly outperforms the others.

Another method to evaluate model quality is by calculating the log likelihood of the data given the model. A higher log likelihood value indicates a better fit. Using the `logLik()` function from the topicmodels package, it suggests that the best model is the LDA model trained with Gibbs sampling in both cases. LDA model trained with Gibbs sampling (LDA\_GIB) consistently outperforms other models in terms of both evaluation metrics, likelihood values, and performance measures across the datasets.

The random component in the optimization process of fitting these models can significantly influence the trained model. Ideally, we seek model stability, meaning minimal impact from initial random conditions. By training models with different random number seeds multiple times on both datasets, we can assess their stability. If a method's accuracy remains consistent across different seeds, it suggests stability and similarity in the produced topic models, even though we're only considering the most prominent topic per document in this analysis.

A computer code on a white background

Description automatically generated

On both datasets, Gibbs sampling leads to a more stable model, with better accuracy particularly evident in the BBC dataset. Although Gibbs sampling generally produces more accurate models, it can become slower with larger datasets. The two LDA models trained with variational methods show scores varying within a roughly 10 percent range on both datasets, with LDA\_VEM consistently slightly outperforming LDA\_VEM\_. The CTM model exhibits the least stability among all models, showing high variance on both datasets. However, it's worth noting that even though the best performance of the CTM model across five iterations is slightly worse than the best accuracy achieved by other methods, it still performs reasonably well.

A computer screen shot of a program

Description automatically generated

If our model isn't stable across iterations, we can adjust the training process by specifying the “nstart” parameter, which determines the number of random restarts during optimization. We've created a modified function called “compute\_model\_list\_r()” that includes this parameter. Additionally, the `seed` parameter now requires a vector of seeds corresponding to the number of random restarts. By using random restarts, training time increases, but even with just five restarts, model accuracy improves. Importantly, this approach helps mitigate fluctuations in the CTM model, bringing its performance closer to the best model in each dataset.

In the LDA\_VEM model, we estimate the parameter vector α. This parameter determines the shape of the Dirichlet distribution used to sample topic distributions. In this implementation, a symmetric distribution is assumed, meaning all αk parameters take the same value. By estimating α, we find it to be much lower than the default value, indicating a peakier topic distribution. We can use the posterior() function to visualize the distribution of topics for each model.

A white background with black text

Description automatically generated

The visualization shows that the LDA\_VEM model assumes a sharply peaked distribution of probabilities for topics, while the other models exhibit a broader spread. The CTM\_VEM model also shows a peak, but with probabilities spread across a wider range of values. Notably, since there are five topics, the minimum probability for the most likely topic is 0.2.

We can assess the smoothness of topic distributions by calculating the model entropy, defined as the average entropy across all topic distributions in the documents. Smooth distributions will have higher entropy compared to peaky distributions. To compute the entropy of a specific topic distribution within a document, we'll use the function “compute\_entropy()”, and to find the average entropy across all documents in the model, we'll use the function “compute\_model\_mean\_entropy()”.

A computer code with text

Description automatically generated

This shows that the LDA\_VEM model, which is the peakiest, has a much lower entropy than the other models.

A computer code with black text

Description automatically generated

We can use the function `terms()` to display the k most frequent terms of each topic in a model. Our custom function takes a model, document term matrix, topic index, and the number of frequent terms to display. First, we compute the most frequent terms for each topic and the most probable topic assignments. Then, we extract the relevant cells from the document term matrix for the specified topic. After summing over the columns, we plot the word cloud using these frequencies. Using this function, we created word clouds for the topics in the BBC dataset using the LDA\_GIB model with 25 words per topic.

**Conclusion:** Topic modeling techniques like LDA and CTM offer effective ways to uncover hidden themes in large text datasets such as news articles. By evaluating metrics like coherence and using visualization methods, we can assess model performance and interpret results. Gibbs sampling tends to provide more stable and accurate models, especially in larger datasets. Overall, understanding parameters and evaluation metrics is crucial for deriving meaningful insights from topic modeling analyses.