The University of Akron

Management Department

Advanced Data Analytics 6500-663

Business Report

Topic Modeling of BBC News

By Kirill Samaray

Introduction

Topic modelling is a powerful technique used to uncover hidden patterns and structures within large collections of textual data. It involves extracting relevant topics from the text by identifying common themes and concepts that are repeated throughout the corpus. In recent years, Topic Modelling has gained significant attention in the field of natural language processing, and various algorithms have been developed to help researchers and businesses uncover the underlying topics within their data.

In this report, we focus on Topic Modelling of BBC news articles using the Latent Dirichlet Allocation (LDA) algorithm. BBC news is a prominent news organization that provides up-to-date news coverage on various topics, including politics, business, entertainment, technology, and sports. Using LDA, we aim to identify the main topics that have been covered by the BBC news team over the past few years and analyze the trends and patterns within those topics.

The LDA algorithm is a widely used probabilistic model for topic modelling that allows for the identification of latent topics within a corpus of documents. LDA assumes that each document is a mixture of a small number of topics, and each topic is characterized by a distribution of words. By estimating the topic mixtures for each document and the word distribution for each topic, LDA can provide a comprehensive overview of the main themes and concepts that are present in the BBC news corpus.

The report provides a detailed analysis of the topic modelling results, including the identified topics and their distribution, the most relevant keywords for each topic, and the evolution of topics over time. This information can be used by businesses and researchers to gain insights into the main news topics covered by the BBC and understand the underlying trends and patterns within those topics.

Problem Statement

In today's world, businesses and organizations are constantly overwhelemed with large volumes of textual data. This data includes news articles, social media posts, customer reviews, and other sources of unstructured text. To gain insights from this data, it is essential to identify the underlying themes and topics within it. However, manually analyzing such large datasets is time-consuming and can be prone to errors.

Therefore, the problem we aim to address in this report is to apply the Latent Dirichlet Allocation (LDA) algorithm to BBC news articles and identify the main topics covered by the news organization. The main challenge in this task is to handle various topics and the fact of having multi layers of the data while identifying relevant topics that accurately reflect the content of the news articles. Furthermore, it is essential to evaluate the quality of the topic modelling results and ensure that they are meaningful and relevant to the BBC news corpus.

By addressing this problem, we aim to provide businesses and researchers with a comprehensive understanding of the main topics covered by the BBC news and the underlying trends and patterns within those topics. This information can be used to gain insights into the interests and preferences of the audience and make informed decisions in areas such as marketing, product development, and content creation.

Objectives

The main objective of this report is to apply the Latent Dirichlet Allocation (LDA) algorithm to BBC news articles and identify the main topics covered by the news organization. To achieve this goal, we have set the following specific objectives:

- Collect and preprocess a dataset of BBC news articles.
- Apply the LDA algorithm to the preprocessed dataset and identify the main topics covered by the BBC news.
- Evaluate the quality of the topic modelling results by measuring smoothness and accuracy.
- Identify the most relevant keywords for each topic, analyze the semantic meaning of each topic, visualize it with the help of word clouds.
- Provide insights and recommendations for businesses and researchers based on the findings of the topic modelling analysis.

Methodology

The methodology used in this report is based on the analysis of publicly available data set "BBC" and a review of the existing literature: Brett Lantz, Machine Learning with R, 2nd Ed., Packet Publishing, 2015 (ISBN: 978-1-78439-390-8). The following report is split into the sections:

- Step 1. Overview of the model.
- Step 2. Collecting data.
- Step 3. Exploring and preparing data.
- Step 4. Model training.
- Step 5. Model Evaluation.
- Step 6. Model Optimization.
- Step 7. Topic Distribution.
- Step 8. Word Distribution.

Step 1. Overview of the model

Latent Dirichlet Allocation (LDA) is a powerful algorithm used in natural language processing for topic modelling. It assumes that each document in a corpus contains a mixture of a small number of latent topics, and each topic is characterized by a distribution of words. LDA can be used to uncover hidden patterns and structures within large collections of textual data, providing insights into the main themes and concepts that are present in the corpus.

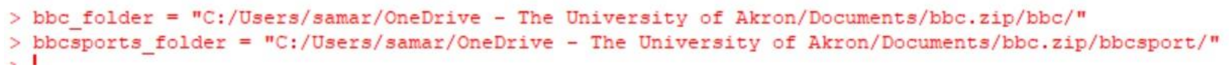
To apply LDA to a corpus of documents, we first need to create a DocumentTermMatrix, which is a sparse matrix that represents the frequency of each term in each document. This can be done using an MM file, which is a file format used for storing sparse matrices. The DocumentTermMatrix is then used as input to train the LDA models using different algorithms, including Variational Bayes (VB), Expectation Maximization (EM), Gibbs Sampling (GS), and Collapsed Gibbs Sampling (CGS).

To evaluate the performance and stability of the LDA models, it is important to use varying initial settings and measure smoothness and accuracy. Once the LDA models are trained, we can display and summarize the topic and word distributions using various visualization techniques, such as word clouds and others. This allows us to gain insights into the main topics covered by the corpus and the evolution of those topics over time.

In this report, we apply the LDA algorithm to BBC news articles and identify the main topics covered by the news organization. We use different algorithms to train the LDA models and evaluate their performance and stability. We also analyze the distribution of topics over time and identify the most relevant keywords for each topic.

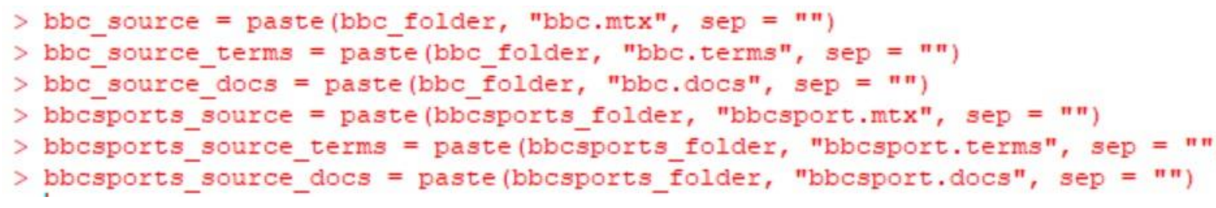
Step 2. Collecting Data

Two datasets were used consisting of articles published by BBC News between 2004 and 2005 in order to evaluate the performance of topic models on actual data. The first dataset, known as the BBC dataset, consisted of 2,225 articles categorized into five topics: business, entertainment, politics, sports, and technology. The second dataset, referred to as the BBCSports dataset, consisted of 737 articles solely related to sports and categorized into five different types of sports: athletics, cricket, football, rugby, and tennis. Our aim was to create topic models for each dataset that group articles together based on their primary topic.



```
> bbc_folder = "C:/Users/samar/OneDrive - The University of Akron/Documents/bbc.zip/bbc/"
> bbcsports_folder = "C:/Users/samar/OneDrive - The University of Akron/Documents/bbc.zip/bbcsport/"
```

As soon as the data is downloaded, there are two folders with files that we can use. To assign the paths of these folders on our computer we used variables bbc_folder and bbcsports_folder. Each of these folders includes: bbc.classes, bbc.terms, bbc.docs, bbc.mtx. The file bbc.terms contains the terms, one per line and which are also the row names of the matrix. In a similar fashion, the file bbc.docs stores documents names which are also the column names of the matrix. The file with the .mtx extension is a sparse matrix file containing a term document matrix. A document term matrix is a table where the rows represent documents (e.g., articles, essays, books) and the columns represent the terms (e.g., words, phrases, concepts) found in those documents. The cells of the matrix contain numerical values that indicate the frequency of occurrence of a term in a specific document. This matrix is used in natural language processing and text mining to represent textual data in a numerical form that can be used for further analysis. It is a common representation used in many text analysis techniques such as topic modeling, clustering, and classification. Just like any other text files, BBC articles must go through some pre-processing such as removing unwanted characters and numbers, removing punctuations, whitespaces, etc. However, with BBC data set, it has been already pre-processed which means the words were stemmed, stop words were removed as well as the terms that were used less than 3 times.

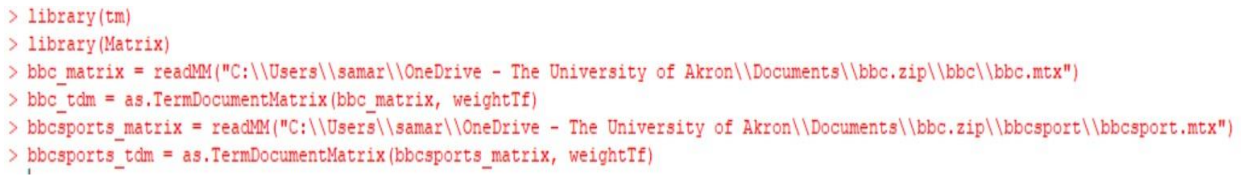


```
> bbc_source = paste(bbc_folder, "bbc.mtx", sep = "")
> bbc_source_terms = paste(bbc_folder, "bbc.terms", sep = "")
> bbc_source_docs = paste(bbc_folder, "bbc.docs", sep = "")
> bbcsports_source = paste(bbcsports_folder, "bbcsport.mtx", sep = "")
> bbcsports_source_terms = paste(bbcsports_folder, "bbcsport.terms", sep = "")
> bbcsports_source_docs = paste(bbcsports_folder, "bbcsport.docs", sep = "")
```

In our case, the matrix rows represent terms found in the articles, and the columns represent the articles themselves. Each entry M[i,j] within the matrix indicates how many times the corresponding term in row i appeared in the document related to column j. The format used to store the matrix in the file is called Matrix Market format. Matrix Market is a file format used for storing matrices in a compact and efficient way. It is designed to facilitate the exchange of matrix data between different scientific computing programs. The Matrix Market format consists of a header section that describes the matrix type, size, and structure, followed by the matrix data stored in coordinate or dense format.

The header section of a Matrix Market file begins with a line containing the word "%%MatrixMarket", followed by a matrix type indicator (e.g., "matrix", "vector"), the storage format (e.g., "coordinate", "dense"), and any additional parameters required for the specific matrix type. The header is followed by the matrix data, which is stored in either coordinate format (row, column, value) or dense format (row-wise or column-wise). Matrix Market format is widely used in numerical linear algebra and related fields such as graph theory, finite element analysis, and optimization. Many scientific computing libraries and software tools support Matrix Market format, making it a standard for exchanging matrix data between different applications.

Since we cannot load data stored in Market Matrix format directly into R, we used readMM() function from the Matrix package. With the help of this package we can load the data and store it into a spare matrix object. Further on with the help of tm package we can also convert it into a term document matrix using as.TermDocumentMatrix() function. Weighting parameter is required in this case to identify the numbers from the original matrix and what they correspond to. Because we have raw term frequencies, we can specify the value weightTf.

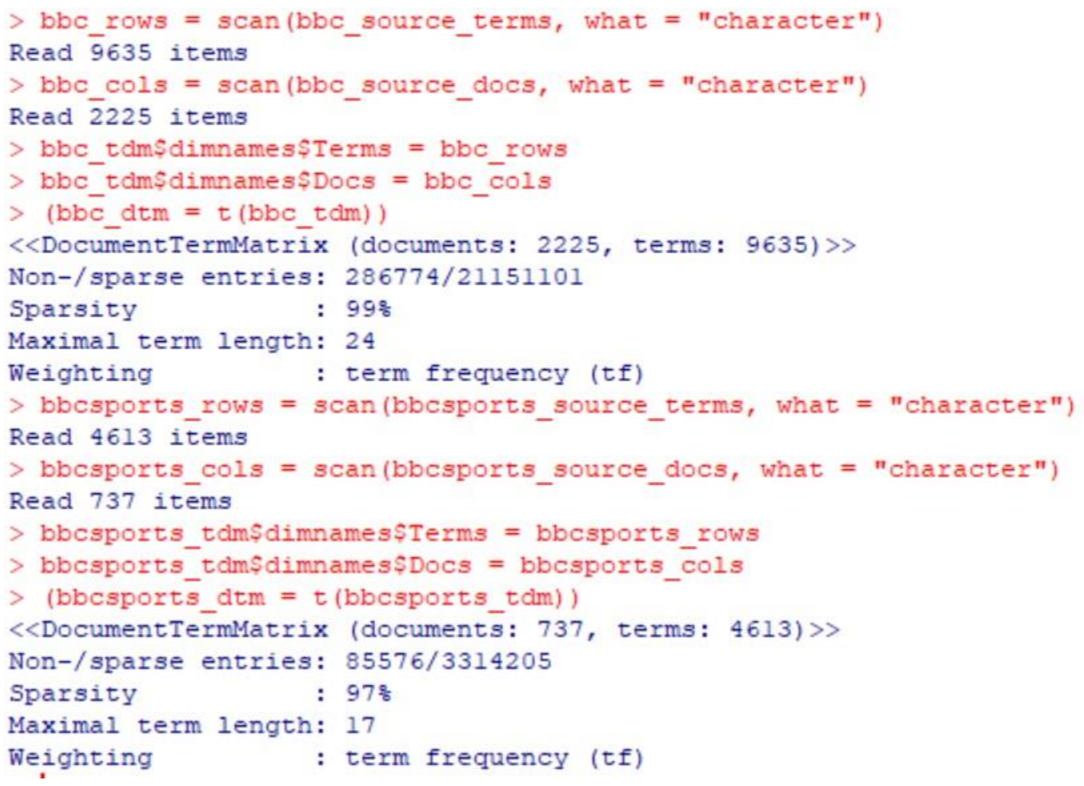


```
> library(tm)
> library(Matrix)
> bbc_matrix = readMM("C:\\Users\\samar\\OneDrive - The University of Akron\\Documents\\bbc.zip\\bbc\\bbc.mtx")
> bbc_tdm = as.TermDocumentMatrix(bbc_matrix, weightTf)
> bbcsports_matrix = readMM("C:\\Users\\samar\\OneDrive - The University of Akron\\Documents\\bbc.zip\\bbcsport\\bbcsport.mtx")
> bbcsports_tdm = as.TermDocumentMatrix(bbcsports_matrix, weightTf)
```

The tm and Matrix packages in R are used for text mining and matrix manipulation, respectively. The tm package provides a set of tools for cleaning, preprocessing, and analyzing text data. The Matrix package provides a set of functions for creating and manipulating sparse and dense matrices in R. Sparse matrices are matrices where most of the values are zero and are commonly used in data analysis when the data is very large and sparse, such as in text mining. Together, these packages are useful for processing and analyzing large amounts of text data in R, and for creating and manipulating matrices efficiently, especially when dealing with large and sparse data sets.

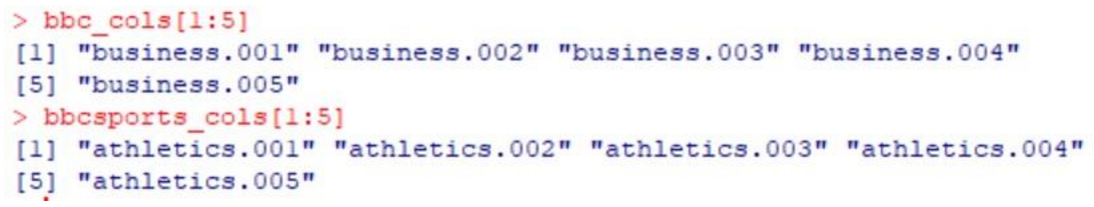
Afterwards, we import the terms and identifiers for the documents from the two remaining files and utilize them to produce suitable names for the rows and columns of the matrices that contain the term and document data. We can employ the standard scan() function to read files that have one entry per line and store the data in vectors. With the term vector and document identifier vector in hand, we will update the names of the rows and columns in the term document matrix. Ultimately, we will transpose this matrix into a document term matrix since it is the format necessary for the following steps.

Step 3. Exploring and preparing data



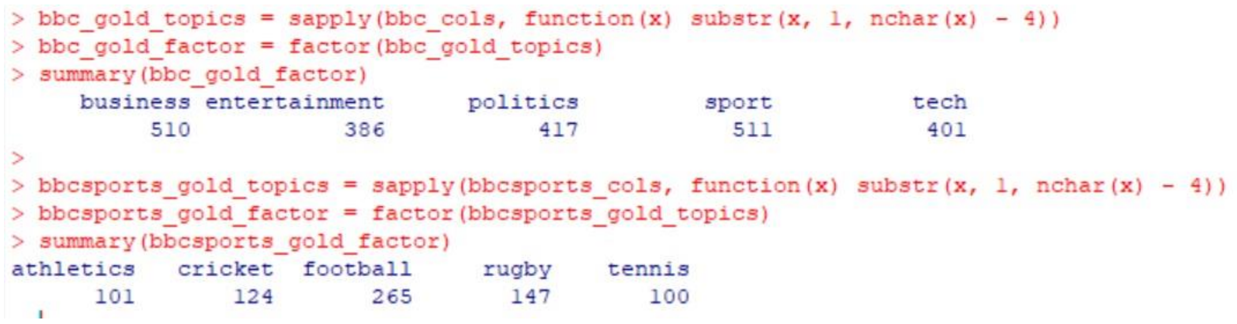
```
> bbc_rows = scan(bbc_source_terms, what = "character")
Read 9635 items
> bbc_cols = scan(bbc_source_docs, what = "character")
Read 2225 items
> bbc_tdm$dimnames$Terms = bbc_rows
> bbc_tdm$dimnames$Docs = bbc_cols
>  (bbc_dtm = t(bbc_tdm))
<<DocumentTermMatrix (documents: 2225, terms: 9635)>>
Non-/sparse entries: 286774/21151101
Sparsity           : 99%
Maximal term length: 24
Weighting          : term frequency (tf)
> bbcsports_rows = scan(bbcsports_source_terms, what = "character")
Read 4613 items
> bbcsports_cols = scan(bbcsports_source_docs, what = "character")
Read 737 items
> bbcsports_tdm$dimnames$Terms = bbcsports_rows
> bbcsports_tdm$dimnames$Docs = bbcsports_cols
>  (bbcsports_dtm = t(bbcsports_tdm))
<<DocumentTermMatrix (documents: 737, terms: 4613)>>
Non-/sparse entries: 85576/3314205
Sparsity           : 97%
Maximal term length: 17
Weighting          : term frequency (tf)
```

By analyzing the above results we can see that BBC data set is having 9635 terms in comparison to 4613 terms in BBCsports dataset as well as 2225 and 737 documents in each one of them respectively. By this we can suggest that BBC dataset is considerably larger than a BBCsports one. Then we had to create vectors with the original classification of the articles. We also found out the format of each document identifier is <topic>.><counter>., by looking at the document IDs.



```
> bbc_cols[1:5]
[1] "business.001" "business.002" "business.003" "business.004"
[5] "business.005"
> bbcsports_cols[1:5]
[1] "athletics.001" "athletics.002" "athletics.003" "athletics.004"
[5] "athletics.005"
```

In order to create a vector with a proper topic assignment, we had to extract the last four characters of each entry and convert it into a factor.



```
> bbc_gold_topics = sapply(bbc_cols, function(x) substr(x, 1, nchar(x) - 4))
> bbc_gold_factor = factor(bbc_gold_topics)
> summary(bbc_gold_factor)
     business entertainment      politics         sport          tech
          510           386           417           511           401
>
> bbcsports_gold_topics = sapply(bbcsports_cols, function(x) substr(x, 1, nchar(x) - 4))
> bbcsports_gold_factor = factor(bbcsports_gold_topics)
> summary(bbcsports_gold_factor)
athletics   cricket  football     rugby    tennis
      101       124       265       147       100
```

It helped us to identify the distribution of various topics in the data set. We can see that there 510 articles on business, 386 on entertainment, 417 on politics, 511 on sport and 401 on tech respectively. As for the BBCsports dataset, there is a slight dominance of football articles – 265, with the other topics having less than 150 articles each. Then we used the above-mentioned data sets to create some topic models using topicmodels package.

Step 4. Model training

The topicmodels package in R provides a set of functions for fitting and analyzing topic models, which are a type of unsupervised machine learning algorithm used to identify patterns and themes in large, unstructured datasets such as text data. The topicmodels package provides a range of tools for working with topic models, including:

Topic modeling algorithms: The package includes several algorithms for fitting topic models, including Latent Dirichlet Allocation (LDA), Correlated Topic Models (CTM), and Hierarchical Dirichlet Process (HDP) models.

Model diagnostics: The package provides several tools for evaluating the quality of a topic model, such as computing model perplexity and coherence scores, and visualizing the distribution of topics and topic-word probabilities.

Model visualization: The package includes several functions for visualizing the results of a topic model, such as word clouds, topic-word heatmaps, and dendrograms of topic hierarchies.

Topic classification: The package includes functions for classifying documents into topics, based on the probabilities assigned to each topic by the model.

In addition, the topicmodels package is designed to integrate with other popular packages for text analysis in R, such as tm and quanteda. For example, users can use the DocumentTermMatrix function from tm to create a document-term matrix, and then use the LDA function from topicmodels to fit an LDA model to the matrix.

For both of data sets we will use four different models:

LDA_VEM (Latent Dirichlet Allocation with Variational Expectation Maximization): This is a standard algorithm for fitting LDA topic models, which assumes that the topic proportions for each document follow a Dirichlet distribution, and the topic-word probabilities follow another Dirichlet distribution. LDA_VEM uses a variational inference approach to estimate the model parameters and iteratively updates the topic proportions and the topic-word probabilities until convergence.

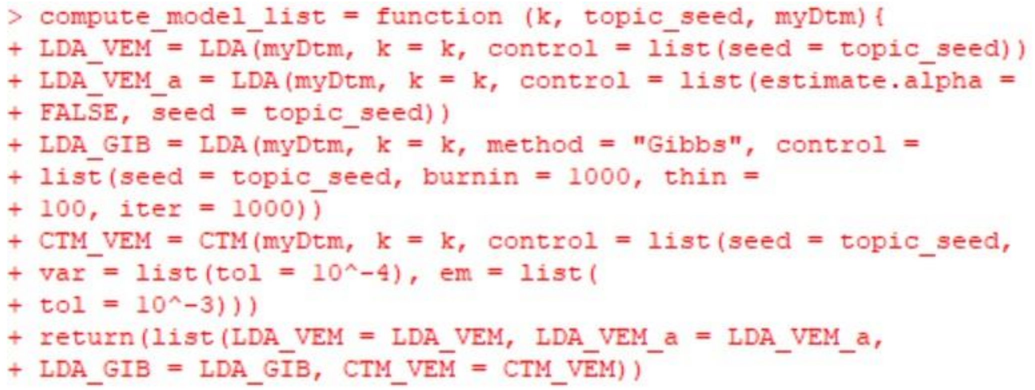
LDA_VEM_α (Latent Dirichlet Allocation with Variational Expectation Maximization and an asymmetric Dirichlet prior): This is a variation of the standard LDA_VEM algorithm that uses an asymmetric Dirichlet prior for the topic proportions. This allows the model to account for situations where certain topics may be more prevalent in certain documents, and can be useful for datasets where the topic distribution is highly imbalanced.

LDA_GIB (Latent Dirichlet Allocation with Gibbs Sampling): This is another algorithm for fitting LDA topic models that uses Gibbs sampling, a Markov Chain Monte Carlo (MCMC) method, to estimate the model parameters. LDA_GIB is often used when the dataset is too large to fit in memory or when the standard LDA_VEM algorithm does not converge.

CTM_VEM (Correlated Topic Models with Variational Expectation Maximization): This is an alternative model to LDA that assumes that the topic proportions are correlated across documents, rather than independently drawn from a Dirichlet distribution. CTM_VEM also uses a variational inference approach to estimate the model parameters, but the optimization problem is more complex than in LDA_VEM due to the correlations between the topic proportions.

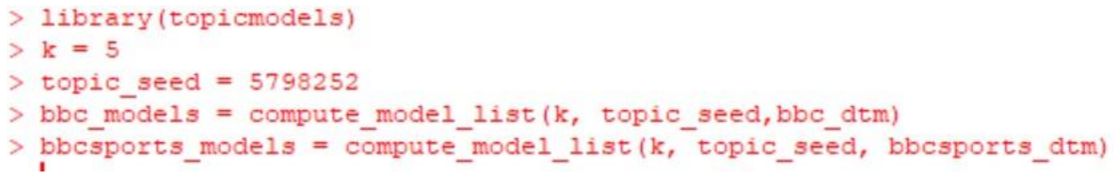
In order to build an LDA model using the topicmodels package, we can utilize the LDA() function which requires four main parameters. Firstly, we need to specify the document term matrix that we want to use for the model. The second parameter, k, is used to set the number of topics we want to generate. The third parameter, method, determines which training algorithm to use. By default, the function uses the VEM algorithm, but we need to specify this parameter for the LDA_GIB model that uses Gibbs sampling.

There is also a control parameter that takes a list of parameters that affect the fitting process. Since the training of topic models involves a random element, we can set a seed parameter to ensure reproducibility. This is also where we can specify whether we want to estimate the α Dirichlet parameter, and include parameters for the Gibbs sampling procedure, such as the number of omitted Gibbs iterations at the start of the training procedure (burnin), the number of omitted in-between iterations (thin), and the total number of Gibbs iterations (iter). To create a function that generates a list of four trained models using a specific document term matrix, required number of topics, and seed, we can set standard values for the training parameters.



```
> compute_model_list = function (k, topic_seed, myDtm){
+ LDA_VEM = LDA(myDtm, k = k, control = list(seed = topic_seed))
+ LDA_VEM_a = LDA(myDtm, k = k, control = list(estimate.alpha =
+ FALSE, seed = topic_seed))
+ LDA_GIB = LDA(myDtm, k = k, method = "Gibbs", control =
+ list(seed = topic_seed, burnin = 1000, thin =
+ 100, iter = 1000))
+ CTM_VEM = CTM(myDtm, k = k, control = list(seed = topic_seed,
+ var = list(tol = 10^-4), em = list(
+ tol = 10^-3)))
+ return(list(LDA_VEM = LDA_VEM, LDA_VEM_a = LDA_VEM_a,
+ LDA_GIB = LDA_GIB, CTM_VEM = CTM_VEM))
```

After the function is ready we can train all four models with both data sets:

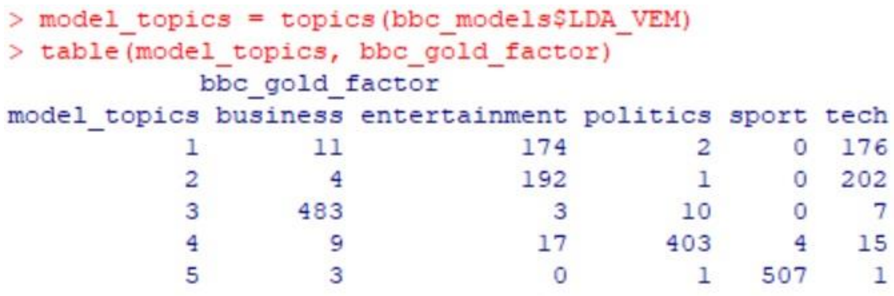


```
> library(topicmodels)
> k = 5
> topic_seed = 5798252
> bbc_models = compute_model_list(k, topic_seed,bbc_dtm)
> bbcsports_models = compute_model_list(k, topic_seed, bbcsports_dtm)
```

Step 5. Model Evaluation

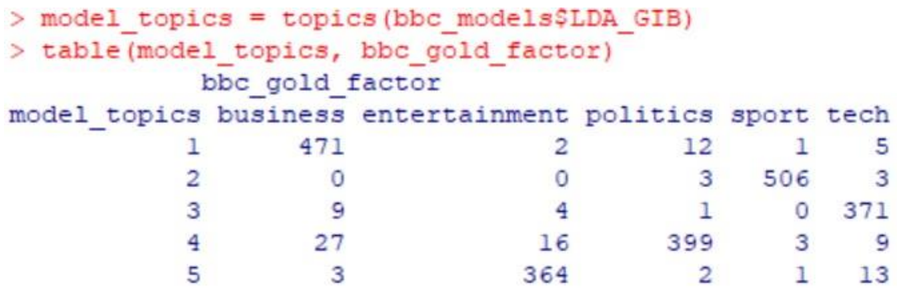
In order to evaluate the performance of the package we can compare the trained topics with the actual topics assigned to the articles. With the help of topics() function we can extract a vector of the most probable topic selected for each document. By default, a k parameter is set to 1, with shows top 1 predicted topic per model. After the topic is found we can compare predicted topics to labeled ones.

LDA_VEM model showed the following results on a BBC data set:



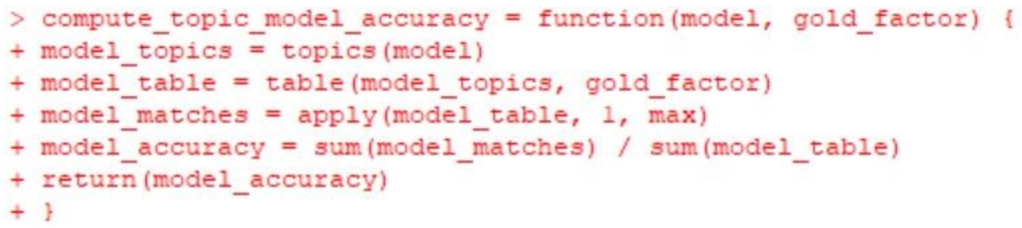
```
> model_topics = topics(bbc_models$LDA_VEM)
> table(model_topics, bbc_gold_factor)
            bbc_gold_factor
model_topics business entertainment politics sport tech
           1       11           174        2     0  176
           2        4           192        1     0  202
           3      483             3       10     0    7
           4        9            17      403     4   15
           5        3             0        1   507    1
```

Looking at the picture we can conclude that topics 3, 4, and 5 match with business, politics and sport areas whereas topics 1 and 2 showed mixed results which proves that the model didn’t succeed in identifying all categories. Ideally, each model topic should match with each so-called gold topic which represents a labeled value. We can test another model LDA_GIB and check whether it shows better results:



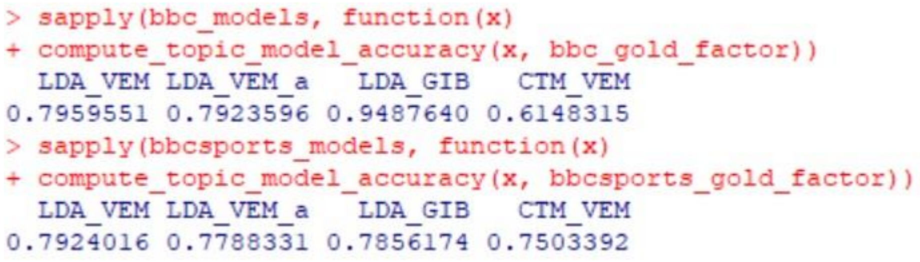
```
> model_topics = topics(bbc_models$LDA_GIB)
> table(model_topics, bbc_gold_factor)
            bbc_gold_factor
model_topics business entertainment politics sport tech
           1      471             2       12     1    5
           2        0             0        3   506    3
           3        9             4        1     0  371
           4       27            16      399     3    9
           5        3           364        2     1   13
```

In contrast with the previous model, LDA_GIB shows a better performance with each topic matching predominantly with one of labeled values. In case more precise values of performance are required, we can calculate the accuracy by stating the ratio of maximum row values over the total number of the documents. For LDA_VEM this value would be (176+202+483+403+507)/2225 = 1771/2225 = 79.6 percent and for LDA_GIB (471+506+371+399+364)/2225 = 2111/2225= 94.9 percent respectively. We can also use the function to compute this value automatically for all the models which helped us to compare them.



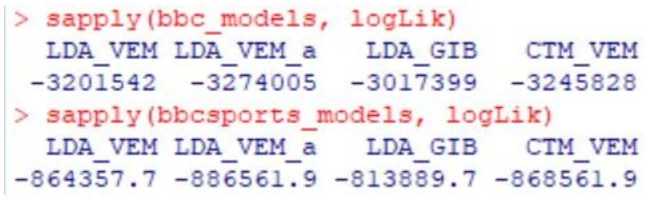
```
> compute_topic_model_accuracy = function(model, gold_factor) {
+ model_topics = topics(model)
+ model_table = table(model_topics, gold_factor)
+ model_matches = apply(model_table, 1, max)
+ model_accuracy = sum(model_matches) / sum(model_table)
+ return(model_accuracy)
+ }
```

Using BBC data set, the LDA_GIB showed the best performance of 95% and the CTM_VEM model showed the lowest one of 61%. LDA_VEM showed 80% and LDA_VEM α showed 79% performance respectively. For the BBCsports data set, all the models illustrated a similar performance, with the LDA_VEM model having 79% which is a bit better than others.



```
> sapply(bbc_models, function(x)
+ compute_topic_model_accuracy(x, bbc_gold_factor))
  LDA_VEM LDA_VEM_a   LDA_GIB   CTM_VEM
0.7959551 0.7923596 0.9487640 0.6148315
> sapply(bbcsports_models, function(x)
+ compute_topic_model_accuracy(x, bbcsports_gold_factor))
  LDA_VEM LDA_VEM_a   LDA_GIB   CTM_VEM
0.7924016 0.7788331 0.7856174 0.7503392
```

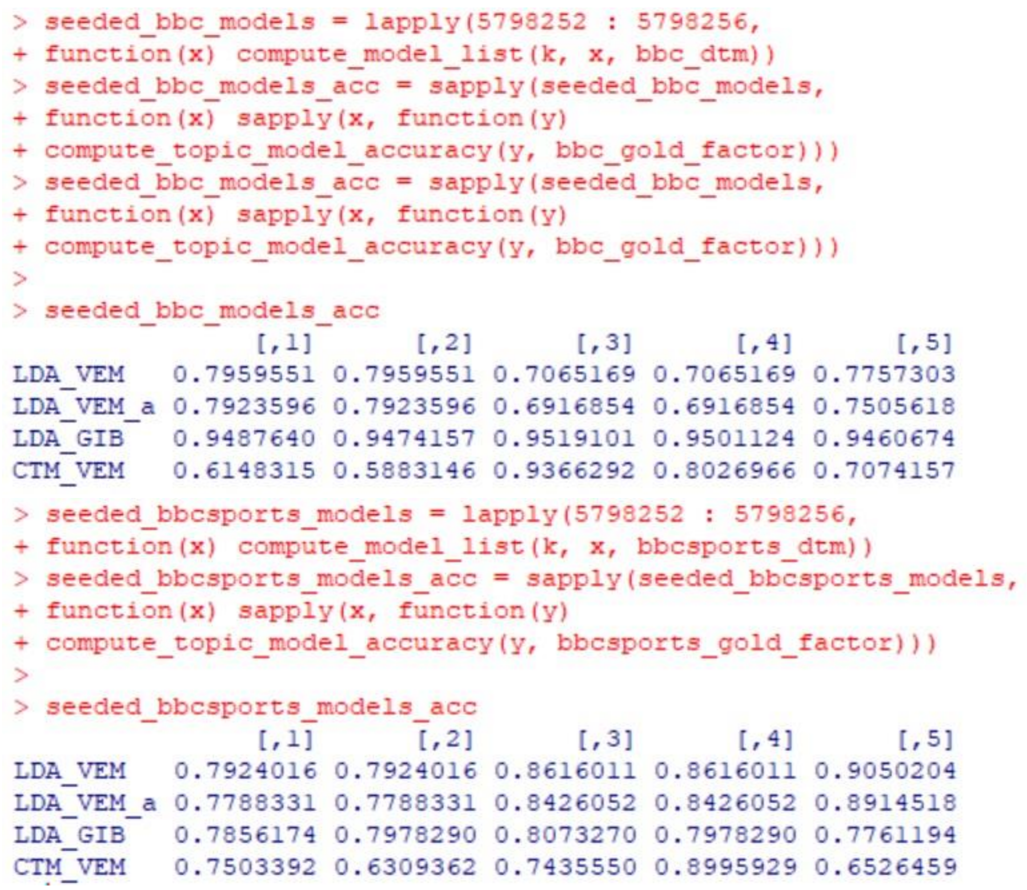
The logLik() function from a topicmodels package can also help us with evaluation of the model performance. It stands for a log likelihood of the data from the model. After applying this function we could see that the LDA model with Gibbs sampling showed the largest values which indicates the better quality for both data sets. The second best model with -3201542 for BBC data set and -864357.7 for BBCsports data set was LDA_VEM model.



```
> sapply(bbc_models, logLik)
  LDA_VEM LDA_VEM_a   LDA_GIB   CTM_VEM
 -3201542  -3274005  -3017399  -3245828
> sapply(bbcsports_models, logLik)
  LDA_VEM LDA_VEM_a   LDA_GIB   CTM_VEM
-864357.7 -886561.9 -813889.7 -868561.9
```

Step 6. Model Optimization

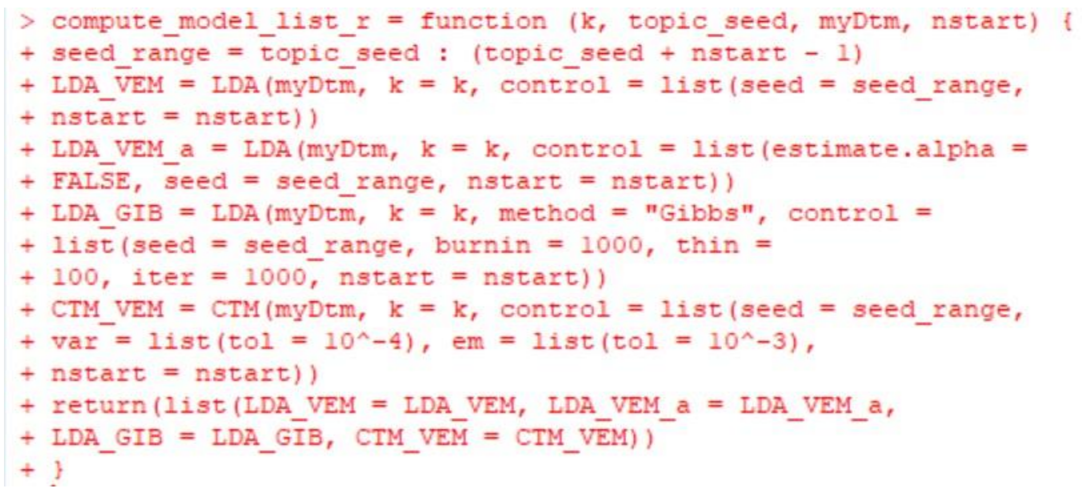
The randomness factor in the optimization process which used to fit these models can greatly influence the trained model. Using different random number seeds can sometimes produce significantly different results. Ideally, we want our model to be consistent, meaning that the impact of the random number seed on the optimization procedure should be minimal. To achieve this, it's recommended to train our four models with multiple random number seeds and analyze the effect of each seed.



```
> seeded_bbc_models = lapply(5798252 : 5798256,
+ function(x) compute_model_list(k, x, bbc_dtm))
> seeded_bbc_models_acc = sapply(seeded_bbc_models,
+ function(x) sapply(x, function(y)
+ compute_topic_model_accuracy(y, bbc_gold_factor)))
> seeded_bbc_models_acc = sapply(seeded_bbc_models,
+ function(x) sapply(x, function(y)
+ compute_topic_model_accuracy(y, bbc_gold_factor)))
>
> seeded_bbc_models_acc
               [,1]      [,2]      [,3]      [,4]      [,5]
LDA_VEM   0.7959551 0.7959551 0.7065169 0.7065169 0.7757303
LDA_VEM_a 0.7923596 0.7923596 0.6916854 0.6916854 0.7505618
LDA_GIB   0.9487640 0.9474157 0.9519101 0.9501124 0.9460674
CTM_VEM   0.6148315 0.5883146 0.9366292 0.8026966 0.7074157
> seeded_bbcsports_models = lapply(5798252 : 5798256,
+ function(x) compute_model_list(k, x, bbcsports_dtm))
> seeded_bbcsports_models_acc = sapply(seeded_bbcsports_models,
+ function(x) sapply(x, function(y)
+ compute_topic_model_accuracy(y, bbcsports_gold_factor)))
>
> seeded_bbcsports_models_acc
               [,1]      [,2]      [,3]      [,4]      [,5]
LDA_VEM   0.7924016 0.7924016 0.8616011 0.8616011 0.9050204
LDA_VEM_a 0.7788331 0.7788331 0.8426052 0.8426052 0.8914518
LDA_GIB   0.7856174 0.7978290 0.8073270 0.7978290 0.7761194
CTM_VEM   0.7503392 0.6309362 0.7435550 0.8995929 0.6526459
```

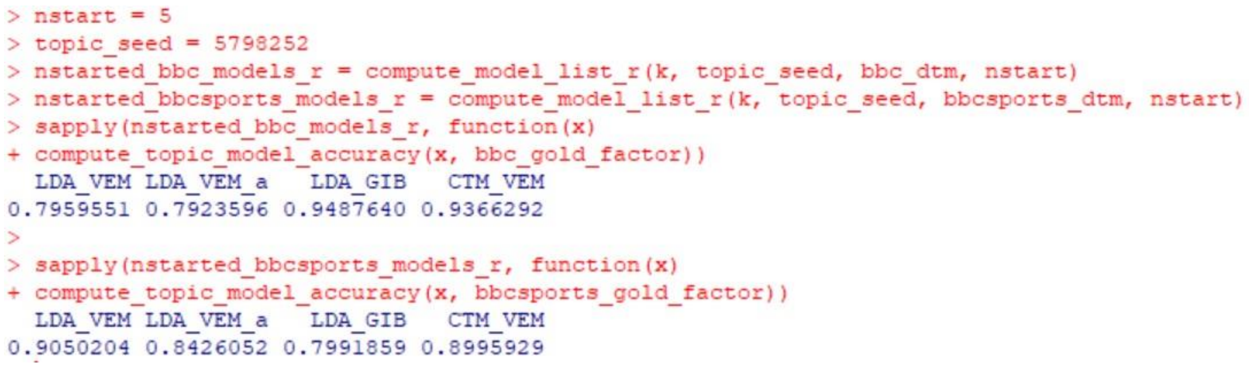
We utilized a sequence of five consecutive seeds and trained our models on both data sets five times. By doing so, we can examine the precision of our models for each of the different seeds. If the precision of a particular approach doesn't differ significantly across the different seeds, we can conclude that the approach is stable and generates comparable topic models, albeit for only the most dominant topic per document in this instance. As shown in the picture above the LDA_GIB model seems to be the most stable since its values across all five seeds doesn't alter much for both data sets. BBC data set: 0.948, 0.947, 0.952, 0,950, 0.946. BBCsports data set: 0.786, 0.798, 0.807, 0.798, 0.776. The least stable model seems to be CTM_VEM with more varying values across both data sets. As for accuracy, Gibbs sampling also looks much better than the other models in a BBC data set, although in BBCsports data set, LDA_VEM model looks slightly better than others including LDA_VEM_a model. Moreover, VEM models tend to be faster than GIB while working with larger data sets.

The other method of optimization could be utilized with the help of nstart parameter while training. This parameter specifies the number of random restarts that will be employed during the optimization process. We have adjusted the compute_model_list() function to create a new function called compute_model_list_r(), which includes an additional parameter, nstart. The seed parameter now requires a sequence of seeds with the same number of entries as the random restarts. We can accomplish this by generating a range of seeds starting from the initial seed provided.



```
> compute_model_list_r = function (k, topic_seed, myDtm, nstart) {
+ seed_range = topic_seed : (topic_seed + nstart - 1)
+ LDA_VEM = LDA(myDtm, k = k, control = list(seed = seed_range,
+ nstart = nstart))
+ LDA_VEM_a = LDA(myDtm, k = k, control = list(estimate.alpha =
+ FALSE, seed = seed_range, nstart = nstart))
+ LDA_GIB = LDA(myDtm, k = k, method = "Gibbs", control =
+ list(seed = seed_range, burnin = 1000, thin =
+ 100, iter = 1000, nstart = nstart))
+ CTM_VEM = CTM(myDtm, k = k, control = list(seed = seed_range,
+ var = list(tol = 10^-4), em = list(tol = 10^-3),
+ nstart = nstart))
+ return(list(LDA_VEM = LDA_VEM, LDA_VEM_a = LDA_VEM_a,
+ LDA_GIB = LDA_GIB, CTM_VEM = CTM_VEM))
+ }
```

This function was used to create a new list of models by random restarts for training.



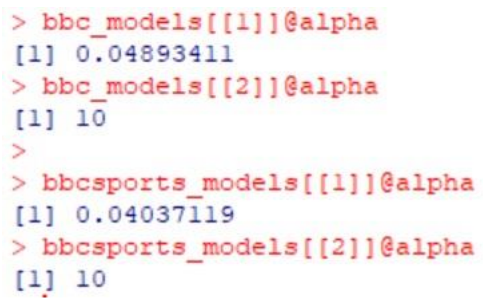
```
> nstart = 5
> topic_seed = 5798252
> nstarted_bbc_models_r = compute_model_list_r(k, topic_seed, bbc_dtm, nstart)
> nstarted_bbcsports_models_r = compute_model_list_r(k, topic_seed, bbcsports_dtm, nstart)
> sapply(nstarted_bbc_models_r, function(x)
+ compute_topic_model_accuracy(x, bbc_gold_factor))
  LDA_VEM LDA_VEM_a   LDA_GIB   CTM_VEM
0.7959551 0.7923596 0.9487640 0.9366292
>
> sapply(nstarted_bbcsports_models_r, function(x)
+ compute_topic_model_accuracy(x, bbcsports_gold_factor))
  LDA_VEM LDA_VEM_a   LDA_GIB   CTM_VEM
0.9050204 0.8426052 0.7991859 0.8995929
```

With this function we can clearly see how the accuracy of the models increased, especially for BBCsports data set. For CTM_VEM a significant improvement of the model was noticed in both data sets.

We could see that in this particular project the number of topics for our data sets was predetermined. We could use this information while working with functions for training. Although, sometimes the number of topics can remain being unknown and we would have to group documents based on some similarity in this case. This endeavor would have been much harder to accomplish and would have required as an option, a cross-validation for a wide range of topics. In case of large data sets, it would require lots of computational and time resources.

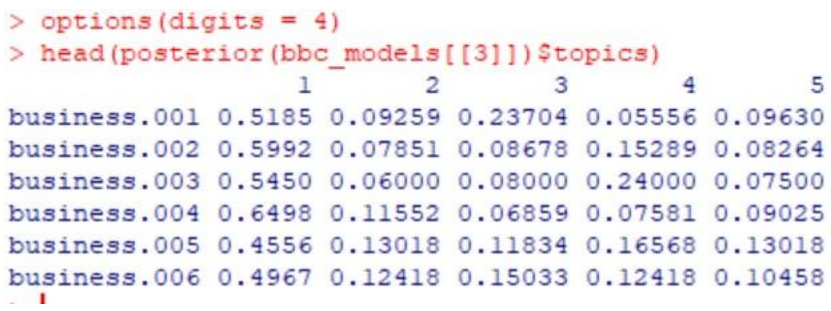
Topic 7. Topic distribution

As we learned from the explanation of the generative process, we employ a Dirichlet distribution to sample a multinomial distribution of topics. In the LDA_VEM model, we estimate the αk parameter vector. It's worth noting that in this implementation, a symmetric distribution is used, meaning we only estimate the value of α, which is applied to all αk parameters. We can compare the values of this parameter used with and without estimation for the LDA models.



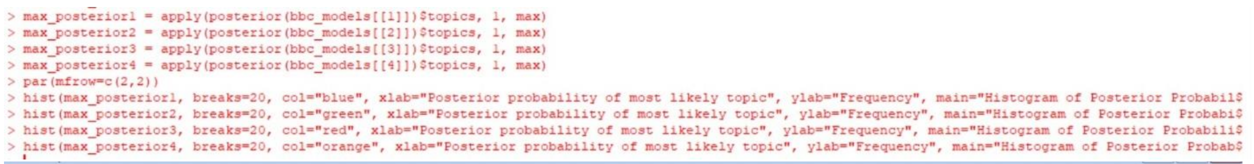
```
> bbc_models[[1]]@alpha
[1] 0.04893411
> bbc_models[[2]]@alpha
[1] 10
>
> bbcsports_models[[1]]@alpha
[1] 0.04037119
> bbcsports_models[[2]]@alpha
[1] 10
```

From the picture above we can see that the estimated value of α is much lower than the one we used. It means that there was a peaky topic distribution for both data sets. We could also see that by illustrating the posterior probabilities of topics using posterior() function. As an example, an LDA_GIB model on BBC data set showed the following list of posterior probabilities:

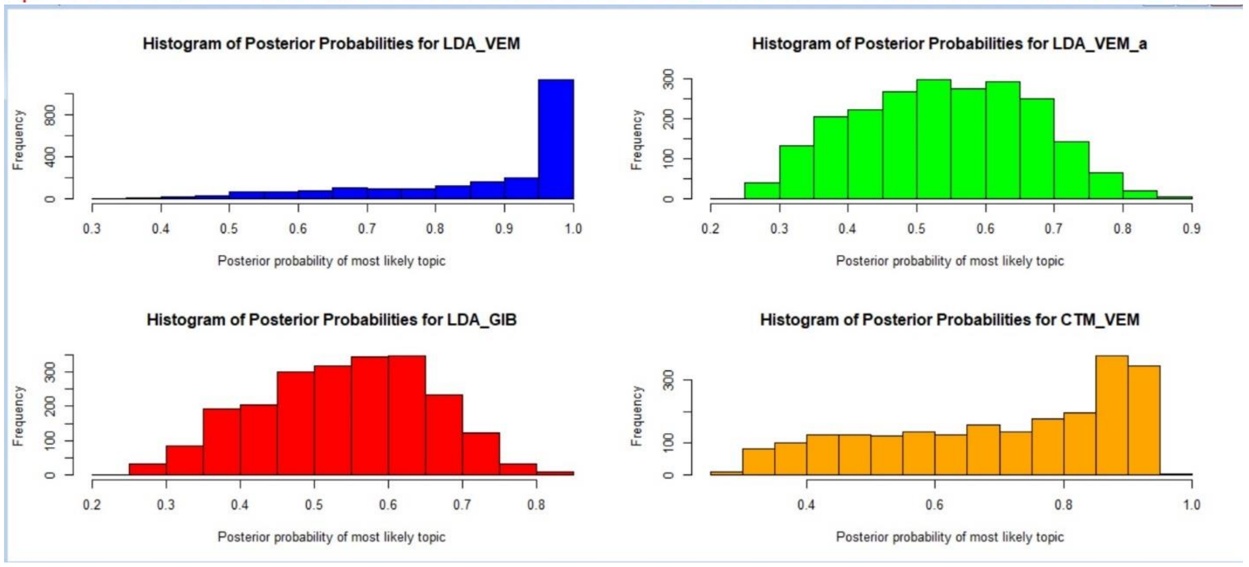


```
> options(digits = 4)
> head(posterior(bbc_models[[3]])$topics)
                  1       2       3       4       5
business.001 0.5185 0.09259 0.23704 0.05556 0.09630
business.002 0.5992 0.07851 0.08678 0.15289 0.08264
business.003 0.5450 0.06000 0.08000 0.24000 0.07500
business.004 0.6498 0.11552 0.06859 0.07581 0.09025
business.005 0.4556 0.13018 0.11834 0.16568 0.13018
business.006 0.4967 0.12418 0.15033 0.12418 0.10458
```

With the help of a histogram, we could demonstrate the probabilities of the most likely topic for all models. Both LDA_VEM and CTA_VEM appear to be peaky although the latter one shows a wider spread of range. On the other hand, LDA_VEM_a and LDA_GIB seem to be flatter distributions with values distributed evenly across the range without any considerable peaks.

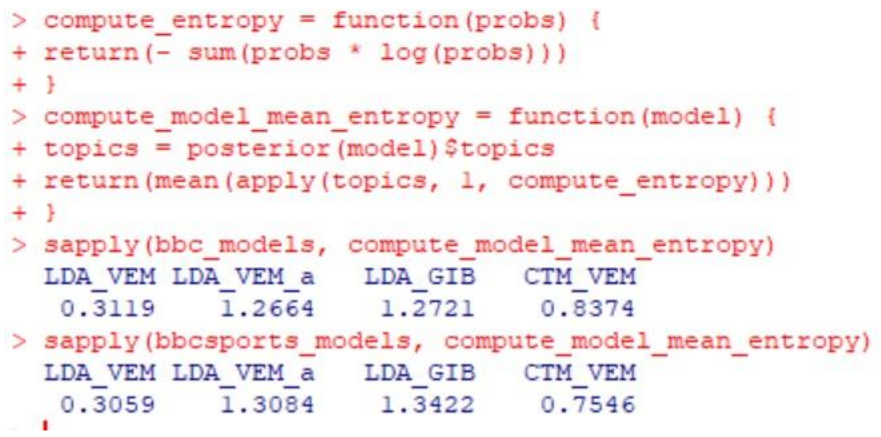


```
> max_posterior1 = apply(posterior(bbc_models[[1]])$topics, 1, max)
> max_posterior2 = apply(posterior(bbc_models[[2]])$topics, 1, max)
> max_posterior3 = apply(posterior(bbc_models[[3]])$topics, 1, max)
> max_posterior4 = apply(posterior(bbc_models[[4]])$topics, 1, max)
> par(mfrow=c(2,2))
> hist(max_posterior1, breaks=20, col="blue", xlab="Posterior probability of most likely topic", ylab="Frequency", main="Histogram of Posterior Probabil$
> hist(max_posterior2, breaks=20, col="green", xlab="Posterior probability of most likely topic", ylab="Frequency", main="Histogram of Posterior Probabi$
> hist(max_posterior3, breaks=20, col="red", xlab="Posterior probability of most likely topic", ylab="Frequency", main="Histogram of Posterior Probabili$
> hist(max_posterior4, breaks=20, col="orange", xlab="Posterior probability of most likely topic", ylab="Frequency", main="Histogram of Posterior Probab$
```



Another method we used for checking the quality of distribution was calculating the average entropy of all topic's distributions for various documents in models. In statistics and information theory, entropy is a measure of the uncertainty or randomness in a probability distribution. A probability distribution with high entropy has many possible outcomes with approximately equal probabilities, while a distribution with low entropy has few possible outcomes with very different probabilities. Mathematically, entropy is often defined as the negative sum of the probabilities of each outcome multiplied by the logarithm of the probability. The concept of entropy has applications in a wide range of fields, including physics, computer science, and economics.

To calculate entropy in our case we used two functions. The first one compute_entropy() was used to calculate the entropy of a specific topic distribution of a document, compute_model_mean_entropy() function was used to defining the average entropy through all the documents. After we used these two functions, we could define average entropies for the models we trained earlier in both data sets.

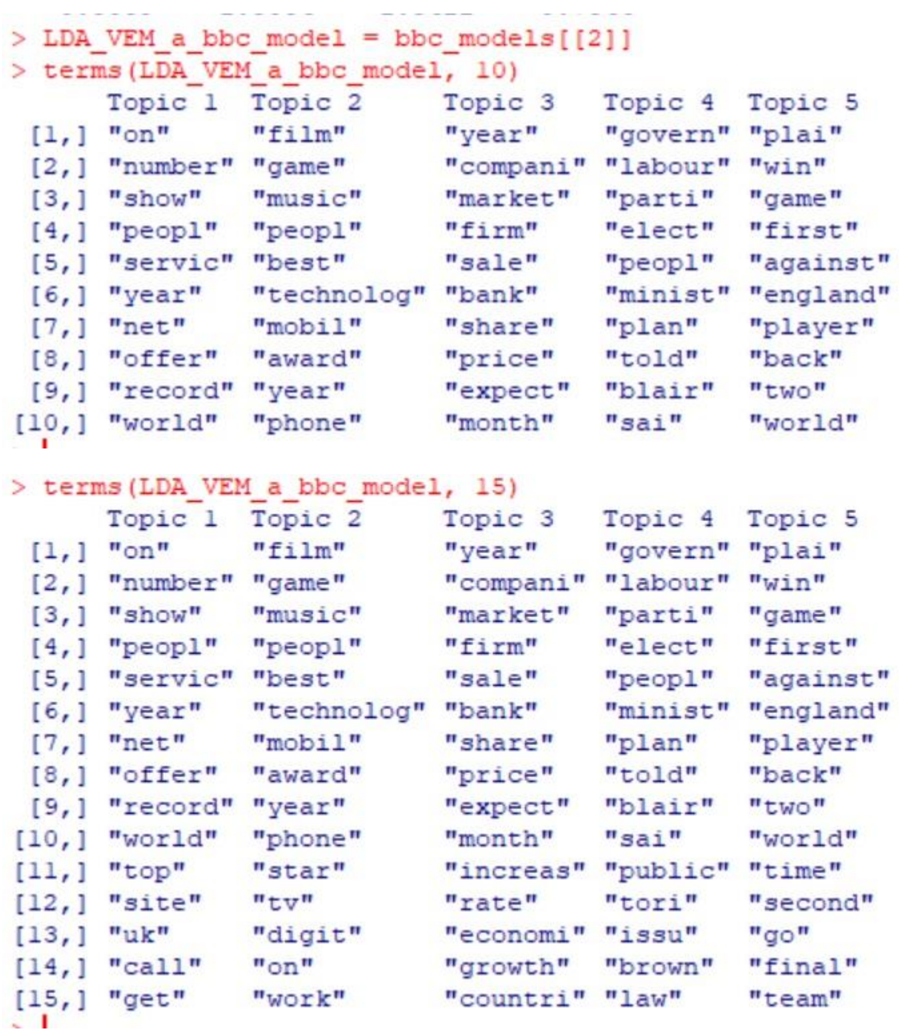


```
> compute_entropy = function(probs) {
+ return(- sum(probs * log(probs)))
+ }
> compute_model_mean_entropy = function(model) {
+ topics = posterior(model)$topics
+ return(mean(apply(topics, 1, compute_entropy)))
+ }
> sapply(bbc_models, compute_model_mean_entropy)
  LDA_VEM LDA_VEM_a   LDA_GIB   CTM_VEM
   0.3119    1.2664    1.2721    0.8374
> sapply(bbcsports_models, compute_model_mean_entropy)
  LDA_VEM LDA_VEM_a   LDA_GIB   CTM_VEM
   0.3059    1.3084    1.3422    0.7546
```

As shown on the picture above, the values of entropy match with our earlier conclusions about the models smoothness. LDA_VEM with the lowest values across both data sets (0.3119 and 0.3059) is the peakiest one, CTM_VEM (0.8374 and 0.7546) is the second peakiest distribution. The other two models showed similar results and smoother distributions which was also seen on the histograms earlier.

Step 8. Word distribution

The final step in the project was to check the most frequent terms for each of the documents and topics. At first we checked the 10 and then 15 most frequent terms of the topics of a LDA_VEM_a model with the help of terms() function. We can alter the number of frequent words by changing k value.



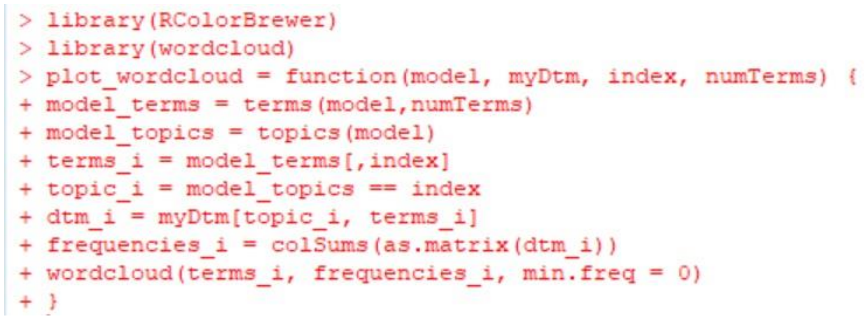
```
> LDA_VEM_a_bbc_model = bbc_models[[2]]
> terms(LDA_VEM_a_bbc_model, 10)
      Topic 1  Topic 2      Topic 3   Topic 4  Topic 5
 [1,] "on"     "film"       "year"    "govern" "plai"
 [2,] "number" "game"       "compani" "labour" "win"
 [3,] "show"   "music"      "market"  "parti"  "game"
 [4,] "peopl"  "peopl"      "firm"    "elect"  "first"
 [5,] "servic" "best"       "sale"    "peopl"  "against"
 [6,] "year"   "technolog"  "bank"    "minist" "england"
 [7,] "net"    "mobil"      "share"   "plan"   "player"
 [8,] "offer"  "award"      "price"   "told"   "back"
 [9,] "record" "year"       "expect"  "blair"  "two"
[10,] "world"  "phone"      "month"   "sai"    "world"

> terms(LDA_VEM_a_bbc_model, 15)
      Topic 1  Topic 2      Topic 3   Topic 4  Topic 5
 [1,] "on"     "film"       "year"    "govern" "plai"
 [2,] "number" "game"       "compani" "labour" "win"
 [3,] "show"   "music"      "market"  "parti"  "game"
 [4,] "peopl"  "peopl"      "firm"    "elect"  "first"
 [5,] "servic" "best"       "sale"    "peopl"  "against"
 [6,] "year"   "technolog"  "bank"    "minist" "england"
 [7,] "net"    "mobil"      "share"   "plan"   "player"
 [8,] "offer"  "award"      "price"   "told"   "back"
 [9,] "record" "year"       "expect"  "blair"  "two"
[10,] "world"  "phone"      "month"   "sai"    "world"
[11,] "top"    "star"       "increas" "public" "time"
[12,] "site"   "tv"         "rate"    "tori"   "second"
[13,] "uk"     "digit"      "economi" "issu"   "go"
[14,] "call"   "on"         "growth"  "brown"  "final"
[15,] "get"    "work"       "countri" "law"    "team"
```

From the pictures above we can assume some assigned topics these trained topics belong to. Since this is not the most accurate and smooth model, Topic 1 and Topic 2 are not very well defined and may belong to both Technology and Entertainment topics. Whereas topics 3-5 are clearly belong to Business, Politics, and Sports respectively.

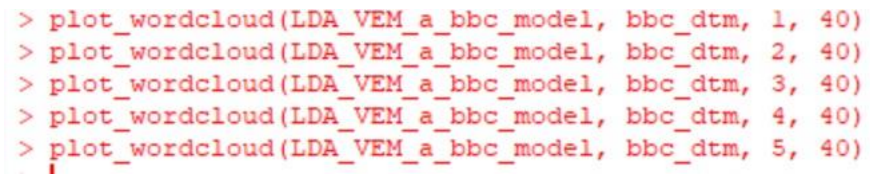
To visualize the above-mentioned results we used a wordcloud package in R. The wordcloud package in R is used to create visually appealing word clouds. The main function of this package is to plot a set of words, where the size of each word is proportional to its frequency or importance in a given text corpus. The package provides various customization options to control the appearance of the word cloud, such as the color palette, font size, shape, and rotation of the words. Users can also specify the number of words to display in the word cloud and the minimum frequency required for a word to be included in the plot.

In our case we had to adjust the DTMs to get the list of frequencies by topics before using it in function. A new function plot_wordcloud() was utilized for this purpose:



```
> library(RColorBrewer)
> library(wordcloud)
> plot_wordcloud = function(model, myDtm, index, numTerms) {
+ model_terms = terms(model,numTerms)
+ model_topics = topics(model)
+ terms_i = model_terms[,index]
+ topic_i = model_topics == index
+ dtm_i = myDtm[topic_i, terms_i]
+ frequencies_i = colSums(as.matrix(dtm_i))
+ wordcloud(terms_i, frequencies_i, min.freq = 0)
+ }
```

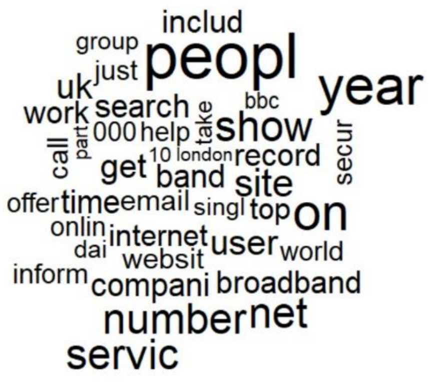
With the help of this function, we can choose a model we need, DTM, index of a topic and the number of most frequent words we want to use with our wordcloud. The function calculates the most frequent terms for the model by the topic, most probable assignments, selects the cells with the terms and documents in question, sums up the columns for computing the frequency of the terms and plots the word cloud. In our example we used LDA_VEM_a model, bbc data set, all 5 topics and 40 most frequent terms per topic:



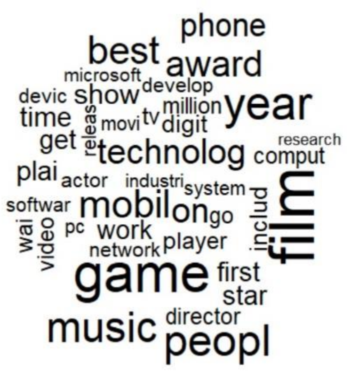
```
> plot_wordcloud(LDA_VEM_a_bbc_model, bbc_dtm, 1, 40)
> plot_wordcloud(LDA_VEM_a_bbc_model, bbc_dtm, 2, 40)
> plot_wordcloud(LDA_VEM_a_bbc_model, bbc_dtm, 3, 40)
> plot_wordcloud(LDA_VEM_a_bbc_model, bbc_dtm, 4, 40)
> plot_wordcloud(LDA_VEM_a_bbc_model, bbc_dtm, 5, 40)
```

The following visualizations show the wordclouds of all 5 topics of LDA_VEM_a model:

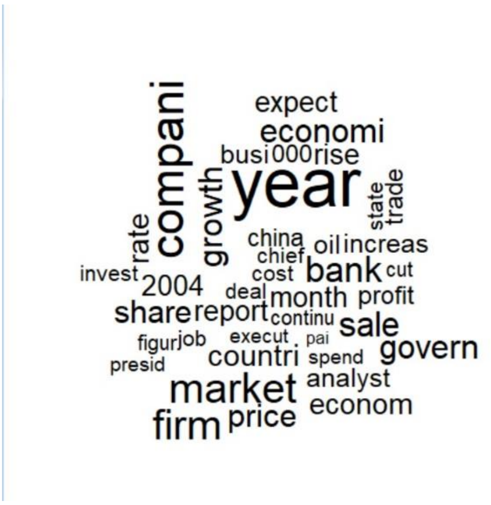
Topic 1



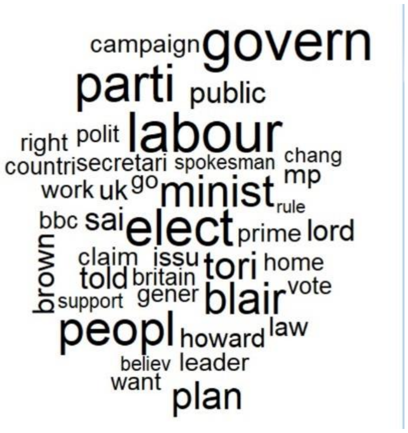
Topic 2



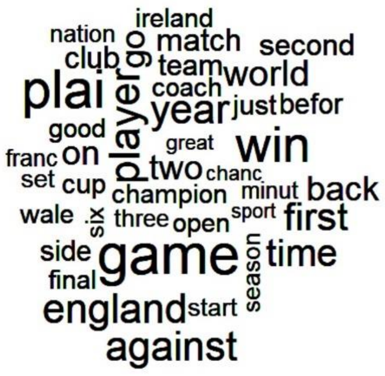
Topic 3



Topic 4



Topic 5



Recommendations

After a thorough consideration of the projects results and analysis of the data with Topic Modelling in BBC News using LDA algorithm, the following recommendations are suggested:

- Further investigation can be carried out to understand the performance of other topic modelling algorithms in comparison to LDA. We have tried using LDA_VEM, LDA_VEM_a, CTM_VEM, and LDA_GIB and different results with different level of accuracy were achieved which suggests using other algorithms for the improvement of the performance.
- The model can be improved by optimizing some parameters used in the LDA algorithm and functions we used, such as the number of topics and alpha and beta, nstart, seed values.
- Additional data can be gathered from other news sources to further broaden the scope of the analysis and provide a more diverse set of topics.
- The output of the model showed decent results and can be used in various applications such as content recommendation, news summarization and filtering, and even sentiment analysis.

Conclusion

In conclusion, the project on Topic Modelling in BBC News using LDA algorithm was successful in identifying relevant topics from the news articles. The LDA algorithm performed well in clustering the news articles into different topics based on the words used in them. The analysis showed that the model was able to capture a wide range of topics, from politics and business to technology and sports.

The results of the project can be used by various stakeholders such as journalists, media companies, and researchers to gain insights into the most relevant topics in the news. The model can also be further refined and used in various applications such as content recommendation, news summarization and filtering, and even sentiment analysis.

Overall, the project has demonstrated the potential of using topic modelling and LDA algorithm in understanding the underlying themes in news articles, which can be useful in various fields such as journalism, media, and data science.

Coding part

```
bbc_folder = "C:/Users/samar/OneDrive - The University of Akron/Documents/bbc.zip/bbc/"

bbcsports_folder = "C:/Users/samar/OneDrive - The University of Akron/Documents/bbc.zip/bbcsport/"

bbc_source = paste(bbc_folder, "bbc.mtx", sep = "")

bbc_source_terms = paste(bbc_folder, "bbc.terms", sep = "")

bbc_source_docs = paste(bbc_folder, "bbc.docs", sep = "")

bbcsports_source = paste(bbcsports_folder, "bbcsport.mtx", sep = "")

bbcsports_source_terms = paste(bbcsports_folder, "bbcsport.terms", sep = "")

bbcsports_source_docs = paste(bbcsports_folder, "bbcsport.docs", sep = "")


library("tm")

library("Matrix")

bbc_matrix = readMM("C:\\Users\\samar\\OneDrive - The University of
Akron\\Documents\\bbc.zip\\bbc\\bbc.mtx")

bbc_tdm = as.TermDocumentMatrix(bbc_matrix, weightTf)

bbcsports_matrix = readMM("C:\\Users\\samar\\OneDrive - The University of
Akron\\Documents\\bbc.zip\\bbcsport\\bbcsport.mtx")

bbcsports_tdm = as.TermDocumentMatrix(bbcsports_matrix, weightTf)


bbc_rows = scan(bbc_source_terms, what = "character")

bbc_cols = scan(bbc_source_docs, what = "character")

bbc_tdm$dimnames$Terms = bbc_rows

bbc_tdm$dimnames$Docs = bbc_cols

(bbc_dtm = t(bbc_tdm))


bbcsports_rows = scan(bbcsports_source_terms, what = "character")

bbcsports_cols = scan(bbcsports_source_docs, what = "character")

bbcsports_tdm$dimnames$Terms = bbcsports_rows

bbcsports_tdm$dimnames$Docs = bbcsports_cols
```

```
(bbcsports_dtm = t(bbcsports_tdm))



bbc_cols[1:5]

bbcsports_cols[1:5]



bbc_gold_topics = sapply(bbc_cols, function(x) substr(x, 1, nchar(x) - 4))

bbc_gold_factor = factor(bbc_gold_topics)

summary(bbc_gold_factor)


bbcsports_gold_topics = sapply(bbcsports_cols, function(x) substr(x, 1, nchar(x) - 4))

bbcsports_gold_factor = factor(bbcsports_gold_topics)

summary(bbcsports_gold_factor)


compute_model_list = function (k, topic_seed, myDtm){

LDA_VEM = LDA(myDtm, k = k, control = list(seed = topic_seed))

LDA_VEM_a = LDA(myDtm, k = k, control = list(estimate.alpha =

FALSE, seed = topic_seed))

LDA_GIB = LDA(myDtm, k = k, method = "Gibbs", control =

list(seed = topic_seed, burnin = 1000, thin =

100, iter = 1000))

CTM_VEM = CTM(myDtm, k = k, control = list(seed = topic_seed,

var = list(tol = 10^-4), em = list(

tol = 10^-3)))

return(list(LDA_VEM = LDA_VEM, LDA_VEM_a = LDA_VEM_a,

LDA_GIB = LDA_GIB, CTM_VEM = CTM_VEM))

}
```

```
library("topicmodels")

k = 5

topic_seed = 5798252

bbc_models = compute_model_list(k, topic_seed,bbc_dtm)

bbcsports_models = compute_model_list(k, topic_seed, bbcsports_dtm)



model_topics = topics(bbc_models$LDA_VEM)

table(model_topics, bbc_gold_factor)


model_topics = topics(bbc_models$LDA_GIB)

table(model_topics, bbc_gold_factor)



compute_topic_model_accuracy = function(model, gold_factor) {

model_topics = topics(model)

model_table = table(model_topics, gold_factor)

model_matches = apply(model_table, 1, max)

model_accuracy = sum(model_matches) / sum(model_table)

return(model_accuracy)

}



sapply(bbc_models, function(x)

compute_topic_model_accuracy(x, bbc_gold_factor))


sapply(bbcsports_models, function(x)

compute_topic_model_accuracy(x, bbcsports_gold_factor))
```

```
sapply(bbc_models, logLik)

sapply(bbcsports_models, logLik)



seeded_bbc_models = lapply(5798252 : 5798256,

function(x) compute_model_list(k, x, bbc_dtm))


seeded_bbcsports_models = lapply(5798252 : 5798256,

function(x) compute_model_list(k, x, bbcsports_dtm))


seeded_bbc_models_acc = sapply(seeded_bbc_models,

function(x) sapply(x, function(y)

compute_topic_model_accuracy(y, bbc_gold_factor)))


seeded_bbc_models_acc


seeded_bbcsports_models_acc <- sapply(seeded_bbcsports_models,

function(x) sapply(x, function(y)

compute_topic_model_accuracy(y, bbcsports_gold_factor)))


seeded_bbcsports_models_acc





compute_model_list_r = function (k, topic_seed, myDtm, nstart) {

seed_range = topic_seed : (topic_seed + nstart - 1)

LDA_VEM = LDA(myDtm, k = k, control = list(seed = seed_range,

nstart = nstart))
LDA_VEM_a = LDA(myDtm, k = k, control = list(estimate.alpha =
FALSE, seed = seed_range, nstart = nstart))
LDA_GIB = LDA(myDtm, k = k, method = "Gibbs", control =
list(seed = seed_range, burnin = 1000, thin =
100, iter = 1000, nstart = nstart))
CTM_VEM = CTM(myDtm, k = k, control = list(seed = seed_range,
var = list(tol = 10^-4), em = list(tol = 10^-3),
nstart = nstart))
return(list(LDA_VEM = LDA_VEM, LDA_VEM_a = LDA_VEM_a,
LDA_GIB = LDA_GIB, CTM_VEM = CTM_VEM))
}


nstart = 5
topic_seed = 5798252
nstarted_bbc_models_r = compute_model_list_r(k, topic_seed, bbc_dtm, nstart)
nstarted_bbcsports_models_r = compute_model_list_r(k, topic_seed, bbcsports_dtm, nstart)
sapply(nstarted_bbc_models_r, function(x)
compute_topic_model_accuracy(x, bbc_gold_factor))

sapply(nstarted_bbcsports_models_r, function(x)
compute_topic_model_accuracy(x, bbcsports_gold_factor))



bbc_models[[1]]@alpha
bbc_models[[2]]@alpha

bbcsports_models[[1]]@alpha
bbcsports_models[[2]]@alpha
```

```
options(digits = 4)

head(posterior(bbc_models[[3]])$topics)

max_posterior1 = apply(posterior(bbc_models[[1]])$topics, 1, max)

max_posterior2 = apply(posterior(bbc_models[[2]])$topics, 1, max)

max_posterior3 = apply(posterior(bbc_models[[3]])$topics, 1, max)

max_posterior4 = apply(posterior(bbc_models[[4]])$topics, 1, max)


par(mfrow=c(2,2))


hist(max_posterior1, breaks=20, col="blue", xlab="Posterior probability of most likely topic",
ylab="Frequency", main="Histogram of Posterior Probabilities for LDA_VEM")

hist(max_posterior2, breaks=20, col="green", xlab="Posterior probability of most likely topic",
ylab="Frequency", main="Histogram of Posterior Probabilities for LDA_VEM_a")

hist(max_posterior3, breaks=20, col="red", xlab="Posterior probability of most likely topic",
ylab="Frequency", main="Histogram of Posterior Probabilities for LDA_GIB")

hist(max_posterior4, breaks=20, col="orange", xlab="Posterior probability of most likely topic",
ylab="Frequency", main="Histogram of Posterior Probabilities for CTM_VEM")




compute_entropy = function(probs) {

return(- sum(probs * log(probs)))

}

compute_model_mean_entropy = function(model) {

topics = posterior(model)$topics

return(mean(apply(topics, 1, compute_entropy)))

}


sapply(bbc_models, compute_model_mean_entropy)
sapply(bbcsports_models, compute_model_mean_entropy)



LDA_VEM_a_bbc_model = bbc_models[[2]]

terms(LDA_VEM_a_bbc_model, 10)



plot_wordcloud = function(model, myDtm, index, numTerms) {

model_terms = terms(model,numTerms)

model_topics = topics(model)

terms_i = model_terms[,index]

topic_i = model_topics == index

dtm_i = myDtm[topic_i, terms_i]

frequencies_i = colSums(as.matrix(dtm_i))

wordcloud(terms_i, frequencies_i, min.freq = 0)

}


plot_wordcloud(LDA_VEM_a_bbc_model, bbc_dtm, 1, 40)

plot_wordcloud(LDA_VEM_a_bbc_model, bbc_dtm, 2, 40)

plot_wordcloud(LDA_VEM_a_bbc_model, bbc_dtm, 3, 40)

plot_wordcloud(LDA_VEM_a_bbc_model, bbc_dtm, 4, 40)

plot_wordcloud(LDA_VEM_a_bbc_model, bbc_dtm, 5, 40)
```

References

1. Brett Lantz, Machine Learning with R, 2nd Ed., Packet Publishing, 2015 (ISBN: 978-1-78439-390-8)