Topic Modeling

Archit Gupta

September 9, 2016

Contents

Topic Modeling 1
Latent Dirichlet Allocation 1
Modeling the topics of online news stories 2
Model stability 6
Finding the number of topics 9
Word distributions 10
Summary 12
References: 13

Topic Modeling

In topic modeling, the key idea is that we can assign a mixture of different classes to an observation. As the field is inspired from information retrieval, we often think of our observations as documents and our output classes as topics. In many applications, this is actually the case and so we will focus on the domain of text documents and their topics, this being a very natural way to learn about this important model. In particular, we'll focus on a technique known as Latent Dirichlet Allocation (LDA), which is the most prominently used method for topic modeling.

Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is the prototypical method to perform topic modeling. First and foremost, we should learn about the Dirichlet distribution, which lends its name to LDA.

The Dirichlet distribution

Suppose we have a classification problem with K classes and the probability of each class is fixed. Given a vector of length K containing counts of the occurrence of each class, we can estimate the probabilities of each class by just dividing each entry in the vector by the sum of all the counts.

Now suppose we would like to predict the number of times each class will occur over a series of N trials. If we have two classes, we can model this with a binomial distribution, as we would normally do in a coin flip experiment. For K classes, the binomial distribution generalizes to the multinomial distribution, where the probability of each class, Pi, is fixed and the sum of all instances of pi equals one. Now, suppose that we wanted to model the random selection of a particular multinomial distribution with K categories. The Dirichlet distribution achieves just that.we are computing the probability of selecting a particular multinomial distribution given a particular parameter combination. Notice that we provide the Dirichlet distribution with a parameter vector whose length is the same.

A special case of the Dirichlet distribution is the Symmetrical Dirichlet distribution, in which all the ??k parameters have an identical value. When the ??k parameters are identical and large in value, we are likely to sample a multinomial distribution that is close to being uniform. Thus, the symmetrical Dirichlet distribution is used when we have no information about a preference over a particular topic distribution and we consider all topics to be equally likely.

The generative process

The generative process is aptly named as it describes how the LDA model assumes that documents, topics, and words are generated in our data. This process is essentially an illustration of the model's assumptions. The optimization procedures that are used in order to fit an LDA model to data, essentially estimate the parameters of the generative process. We'll now see how this process works.

1. For each of our K topics, draw a multinomial distribution, ??k, over the words in our vocabulary using a Dirichlet distribution parameterized by a vector ??. This vector has length V, the size of our vocabulary. Even though we sample from the same Dirichlet distribution each time, we've seen that the sampled multinomial distributions will likely differ from each other.
2. For every document, d, that we want to generate:
    i) Determine the mix of topics for this document by drawing a multinomial distribution, ??k, from a Dirichlet distribution, this time parameterized by a vector ?? of length K, the number of topics. Each document will thus have a different mix of topics.
    ii) For every word, w, in the document that we want to generate:
        A) Use the multinomial topic distribution for this document, ??k, to draw a topic with which this word will be associated.
        B) Use that particular topic's distribution, ??k, to pick the actual word.

Note that we use two differently parameterized Dirichlet distributions in our generative process, one for drawing a multinomial distribution of topics and another for drawing a multinomial distribution of words.

Modeling the topics of online news stories

To see how topic models perform on real data, we will look at two data sets containing articles originating from BBC News during the period of 2004-2005. The first data set, which we will refer to as the BBC data set, contains 2,225 articles that have been grouped into five topics. These are business, entertainment, politics, sports, and technology.

The second data set, which we will call the BBCSports data set, contains 737 articles only on sports. These are also grouped into five categories according to the type of sport being described. The five sports in question are athletics, cricket, football, rugby, and tennis.

Our objective will be to see if we can build topic models for each of these two data sets that will group together articles from the same major topic. The two data sets can be found at *http://mlg.ucd.ie/datasets/bbc.html*

The datasets have been pre-processed as follows: stemming (Porter algorithm), stop-word removal (stop word list) and low term frequency filtering (count < 3) have already been applied to the data. The files contained in the archives given above have the following formats:

- .mtx: Original term frequencies stored in a sparse data matrix in Matrix Market format.
- .terms: List of content-bearing terms in the corpus, with each line corresponding to a row of the sparse data matrix.
- .docs: List of document identifiers, with each line corresponding to a column of the sparse data matrix.
- .classes: Assignment of documents to natural classes, with each line corresponding to a document.
- .urls: Links to original articles, where appropriate.

```
bbc_folder <- file.path(path2data, "bbc_folder")
bbcsports_folder <- file.path(path2data, "bbcsports_folder")

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 order to load data into R from a file in Market Matrix format, we can use the readMM() function from the Matrix R package. This function loads the data and stores it into a sparse matrix object. We can convert this into a term document matrix that the tm package can interpret using the as.TermDocumentMatrix() function in the tm package. Aside from the matrix object that is the first argument to that function, we also need to specify the weighting parameter. This parameter describes what the numbers in the original matrix correspond to. In our case, we have raw term frequencies, so we specify the value weightTf:

```
library("tm")      #for converting sparse matrix into term document matrix
```

```
## Loading required package: NLP
```

```
library("Matrix")  #for reading file in market matrix format
bbc_matrix <- readMM(bbc_source)
bbc_tdm <- as.TermDocumentMatrix(bbc_matrix, weightTf)
bbcsports_matrix <- readMM(bbcsports_source)
bbcsports_tdm <- as.TermDocumentMatrix(bbcsports_matrix,weightTf)
```

Next, we load the terms and document identifiers from the two remaining files and use these to create appropriate row and column names respectively for the term document matrices. We can use the standard scan() function to read files with a single entry per line and load the entries into vectors. Once we have a term vector and a document identifier vector, we will use these to update the row and column names for the term document matrix. Finally, we'll transpose this matrix into a document term matrix as this is the format we will need for subsequent steps:

```
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))
```

```
## <<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")
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))
```

```
## <<DocumentTermMatrix (documents: 737, terms: 4613)>>
## Non-/sparse entries: 85576/3314205
## Sparsity           : 97%
## Maximal term length: 17
## Weighting          : term frequency (tf)
```

We now have the document term matrices for our two data sets ready. We can see that there are roughly twice as many terms for the BBC data set as there are for the BBCSports data set and the latter also has about a third of the number of documents, so it is a much smaller data set.

we must also create the vectors containing the original topic classification of the articles.

```
bbc_cols[1:5]
```

```
## [1] "business.001" "business.002" "business.003" "business.004" "business.005"
```

```
bbcsports_cols[1:5]
```

```
## [1] "athletics.001" "athletics.002" "athletics.003" "athletics.004" "athletics.005"
```

To create a vector with the correct topic assignments, we simply need to strip out the last four characters of each entry. If we then convert the result into a factor, we can see how many documents we have per topic:

```
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
```

For each of our two data sets, we will now build some topic models using the package **topicmodels**. This is a very useful package as it allows us to use data structures created with the tm package to perform topic modeling.

For each data set, we will build the following four different topic models:

- LDA_VEM: This is an LDA model trained with the Variational Expectation Maximization (VEM) method. This method automatically estimates the ?? Dirichlet parameter vector.
- LDA_VEM_??: This is an LDA model trained with VEM but the difference here is that that the ?? Dirichlet parameter vector is not estimated.
- LDA_GIB: This is an LDA model trained with Gibbs sampling.
- CTM_VEM: This is an implementation of the Correlated Topic Model (CTM) model trained with VEM. Currently, the topicmodels package does not support training this method with Gibbs sampling.

To train an LDA model, the topicmodels package provides us with the LDA() function. We will use four key parameters for this function. The first of these specifies the document term matrix for which we want to build an LDA model. The second of these, k, specifies the target number of topics we want to have in our model. The third parameter, method, allows us to select which training algorithm to use. This is set to VEM by default, so we only need to specify this for our LDA_GIB model that uses Gibbs sampling.

Finally, there is a control parameter, which takes in a list of parameters that affect the fitting process. As there is an inherent random component involved in the training of topic models, we can specify a seed parameter in this list in order to make the results reproducible.This is also where we can 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 train a CTM model, the topicmodels package provides us with the CTM() function, which has a similar syntax with the LDA() function.

```
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))
}
```

We'll now use this function to train a list of models for the two 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)
```

Given one of these trained models, we can use the topics() function to get a vector of the most likely topic chosen for each document.

This function actually takes a second parameter k, by default set to 1, which returns the top k topics predicted by the model. We only want one topic per model in this particular instance. Having found the most likely topic, we can then tabulate the predicted topics against the vector of labeled topics.These are the results for the LDA_VEM model for the 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 this table, we can see that topic 5 corresponds almost exclusively to the sports category. Similarly, topics 4 and 3 seem to match to the politics and business categories respectively. Unfortunately, models 1 and 2 both contain a mixture of entertainment and technology articles and as a result this model hasn't really succeeded in distinguishing between the categories that we want.

It should be clear that in an ideal situation, each model topic should match to one gold topic (we often use the adjective gold to refer to the correct or labeled value of a particular variable. This is derived from the expression gold standard which refers to a widely accepted standard). We can repeat this process on the LDA_GIB model, where the story is different:

```
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
```

Intuitively, we feel that this topic model is a better match to our original topics than the first, as evidenced by the fact that each model topic selects articles from primarily one gold topic.

A rough way to estimate the quality of the match between a topic model and our target vector of topics is to say that the largest value in every row corresponds to the gold topic assigned to the model topic represented by that row. Then, the total accuracy is the ratio of these maximum row values over the total number of documents. In the preceding example, for the LDA_GIB model, this number would be (471+506+371+399+364)/2225 = 2111/2225= 94.9 percent. The following function computes this value given a model and a vector of gold topics:

```
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 this notion of accuracy, let's see which model performs better in our two data sets:

```
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
```

For the BBC data set, we see that the LDA_GIB model significantly outperforms the others and the CTM_VEM model is significantly worse than the LDA models. For the BBCSports data set, all the models perform roughly the same, but the LDA_VEM model is slightly better.

Another way to assess the quality of a model fit is computing the **log likelihood of the data given the model**, remembering that the **larger this value, the better the fit**. We can do this with the logLik() function in the topicmodels package, which suggests that the best model is the LDA model trained with Gibbs sampling in both cases:

```
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
```

Model stability

It turns out that the random component of the optimization procedures involved in fitting these models often has a significant impact on the model that is trained. Put differently, we may find that if we use different random number seeds, the results may sometimes change appreciably.

Ideally, we would like our model to be stable, which is to say that we would like the effect of the initial conditions of the optimization procedure that are determined by a random number seed to be minimal. It is a good idea to investigate the effect of different seeds on our four models by training them on multiple seeds:

```
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))
```

Here we used a sequence of five consecutive seeds and trained our models on both data sets five times. Having done this, we can investigate the accuracy of our models for the various seeds. If the accuracy of a method does not vary by a large degree across the seeds, we can infer that the method is quite stable and produces similar topic models (although, in this case, we are only considering the most prominent topic per document).

```
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_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
```

On both data sets, we can clearly see that Gibbs sampling results in a more stable model and in the case of the BBC data set, it is also the clear winner in terms of accuracy. Gibbs sampling generally tends to produce more accurate models but even though it was not readily apparent on these data sets, it can become significantly slower than VEM methods once the data set becomes large.

The two LDA models trained with variational methods exhibit scores that vary within a roughly 10 percent range on both data sets. On both data sets, we see that LDA_VEM is consistently better than LDA_VEM_a by a small amount. This method also produces, on average, better accuracy among all models in the BBCSports data set. The CTM model is the least stable of all the models, exhibiting a high degree of variance on both data sets. Interestingly, though, the highest performance of the CTM model across the five iterations performs marginally worse than the best accuracy possible using the other methods.

If we see that our model is not very stable across a few seeded iterations, we can specify the nstart parameter during training, which specifies the number of random restarts that are used during the optimization procedure. To see how this works in practice, we have created a modified compute_model_list() function that we named compute_model_list_r(), which takes in an extra parameter, nstart.

The other difference is that the seed parameter now needs a vector of seeds with as many entries as the number of random restarts. To deal with this, we will simply create a suitably sized range of seeds starting from the one provided. Here is our new function:

```
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))

}
```

We will use this function to create a new model list. Note that using random restarts means we are increasing the amount of time needed to train, so these next few commands will take some time to complete.

```
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
```

Note that even after using only five random restarts, the accuracy of the models has improved. More importantly, we now see that using random restarts has overcome the fluctuations that the CTM model experiences and as a result it is now performing almost as well as the best model in each data set.

Finding the number of topics

In this predictive task, the number of different topics was known beforehand. This turned out to be very important because it is provided as an input to the functions that trained our models. The number of topics might not be known when we are using topic modeling as a form of exploratory analysis where our goal is simply to cluster documents together based on the similarity of their topics.

This is a challenging question and bears some similarity to the general problem of selecting the number of clusters when we perform clustering. One proposed solution to this problem is to perform cross-validation over a range of different numbers of topics. This approach will not scale well at all when the data set is large, especially since training a single topic model is already quite computationally intensive when we factor issues such as random restarts.

Topic distributions

We saw in the description of the generative process that we use a Dirichlet distribution to sample a multinomial distribution of topics. In the LDA_VEM model, the ??k parameter vector is estimated. Note that in all cases, a symmetric distribution is used in this implementation so that we are only estimating the value of ??, which is the value that all the ??k parameters take on. For the LDA models, we can investigate which value of this parameter is used with and without estimation:

```
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
```

As we can see, when we estimate the value of ??, we obtain a much lower value of ?? than we use by default, indicating that for both data sets, the topic distribution is thought to be peaky. We can use the posterior() function in order to view the distribution of topics for each model.

```
options(digits = 4)
head(posterior(bbc_models[[1]])$topics)
```

```
##                      1         2      3         4         5
## business.001 0.2700360 0.0477374 0.6818 0.0002222 0.0002222
## business.002 0.0002545 0.0002545 0.9990 0.0002545 0.0002545
## business.003 0.0003257 0.0003257 0.9987 0.0003257 0.0003257
## business.004 0.0002153 0.0002153 0.9991 0.0002153 0.0002153
## business.005 0.0337131 0.0004104 0.9651 0.0004104 0.0004104
## business.006 0.0423153 0.0004740 0.9563 0.0004740 0.0004740
```

Another approach to estimating the smoothness of the topic distributions is to compute the model entropy. We will define this as the average entropy of all the topic distributions across the different documents. Smooth distributions will exhibit higher entropy than peaky distributions. To compute the entropy of our model, we will define two functions. The function compute_entropy() computes the entropy of a particular topic distribution of a document, and the compute_model_mean_entropy() function computes the average entropy across all the different documents in the model:

```
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)))
}
```

Using these functions, we can compute the average model entropies for the models trained on our two data sets:

```
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
```

the LDA_VEM model,has a much lower entropy than the other models.

Word distributions

we are often also interested in understanding the most important terms that are frequent in documents that are assigned to the same topic. We can see the k most frequent terms of the topics of a model using the function terms(). This takes in a model and a number specifying the number of most frequent terms that we want retrieved. Let's see the ten most important words per topic in the LDA_GIB model of the BBC data set:

```
GIB_bbc_model <- bbc_models[[3]]
terms(GIB_bbc_model, 10)
```

```
##       Topic 1   Topic 2   Topic 3     Topic 4  Topic 5
##  [1,] "year"    "plai"    "peopl"     "govern" "film"
##  [2,] "compani" "game"    "game"      "labour" "year"
##  [3,] "market"  "win"     "servic"    "parti"  "best"
##  [4,] "sale"    "against" "technolog" "elect"  "show"
##  [5,] "firm"    "england" "mobil"     "minist" "includ"
##  [6,] "expect"  "first"   "on"        "plan"   "on"
##  [7,] "share"   "back"    "phone"     "sai"    "award"
##  [8,] "month"   "player"  "get"       "told"   "music"
##  [9,] "bank"    "world"   "work"      "peopl"  "top"
## [10,] "price"   "time"    "wai"       "public" "star"
```

A very handy way to visualize frequent terms in a collection of documents is through a word cloud. The R package wordcloud is useful for creating these.

Unfortunately, we will have to do some manipulation on the document term matrices in order to get the word frequencies by topic so that we can feed them into this function. To that end, we've created our own

function plot_wordcloud() as follows:

```
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)
}
```

Our function takes in a model, a document term matrix, an index of a topic, and the number of most frequent terms that we want to display in the word cloud. We begin by first computing the most frequent terms for the model by topic as we did earlier. We also compute the most probable topic assignments. Next, we subset the document term matrix so that we obtain only the cells involving the terms we are interested in and the documents corresponding to the topic with the index that we passed in as a parameter.

From this reduced document term matrix, we sum over the columns to compute the frequencies of the most frequent terms and finally we can plot the word cloud. We've used this function to plot the word clouds for the topics in the BBC data set using the LDA_GIB model and 25 words per topic. This is shown here:

```
library(wordcloud)
par(mfrow=c(2,3))
plot_wordcloud(GIB_bbc_model, bbc_dtm, 1,25)
plot_wordcloud(GIB_bbc_model, bbc_dtm, 2,25)
plot_wordcloud(GIB_bbc_model, bbc_dtm, 3,25)
plot_wordcloud(GIB_bbc_model, bbc_dtm, 4,25)
plot_wordcloud(GIB_bbc_model, bbc_dtm, 5,25)
```

Summary

The primary technique for topic modeling on which we focused was Latent Dirichlet Allocation (LDA). This derives its name from the fact that it assumes that the topic and word distributions that can be found inside a document arise from hidden multinomial distributions that are sampled from Dirichlet priors. We saw that the generative process of sampling words and topics from these multinomial distributions mirrors many of the natural intuitions that we have about this domain; however, it notably fails to account for correlations between the various topics that can co-occur inside a document.

In our experiments with LDA, we saw that there is more than one way to fit an LDA model, and in particular we saw that a method known as Gibbs sampling tends to be more accurate, even if it often is more computationally expensive. In terms of performance, we saw that when the topics in question are quite distinct from each other, such as the topics in the BBC data set, we could get very high accuracy in our topic prediction.

At the same time, however, when we classified documents with topics that are more similar to each other, such as the different sports documents in the BBCSports data set, we saw that this posed more of a challenge and our results were not quite as high. In our case, another factor that probably played a role is that both the number of documents and the available features were much fewer in number with the BBCSports data set. Currently, there is an increasing number of variations on LDA that are being researched and developed in order to deal with limitations in both performance and training speed.Topic models can be viewed as a form of clustering, and this was our first glimpse in this area.

References:

1. D. Greene and P. Cunningham. "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006.
2. Mastering Preditive Analytics with R - Rui Miguel Forte
3. Supervised Topic Models by David M. Blei and Jon D. McAuliffe.