Learning from Big Data: Tutorial 1.2

Tayyip Altan

September 2024

Introduction

This file illustrates LDA and word2vec using the review data.

1. Loading libraries

Before starting the tutorial, make sure you have all the required libraries properly installed. Simply run this chunk of code below.

```
# Required packages. P_load ensures these will be installed and loaded.
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tm, openNLP, nnet, dplyr, tidyr, ggplot2, reshape2,
               latex2exp, topicmodels,word2vec,tokenizers)
```

2. Load the reviews

```
# Load the review data. Note that we are now using the fileEncoding parameter when
#      calling read.csv() - this helps reading the review text correctly for further
#      processing (by correctly interpreting the non-ASCII symbols)
reviews_raw <- read.csv('Reviews_tiny.csv', fileEncoding="ISO-8859-1")

# Selecting only the relevant columns from the entire dataset
reviews_raw <- reviews_raw %>%
               select(movie_name,review_code,   reviewer,   review_date, num_eval,
                      prob_sentiment,words_in_lexicon_sentiment_and_review, ratio_helpful,
                      raters,
                      prob_storyline,   prob_acting,    prob_sound_visual,
                      full_text,    processed_text,
                      release_date, first_week_box_office,MPAA, studio, num_theaters )

# Determining the total number of reviews in our dataset
total_reviews <- nrow(reviews_raw)


# Loading fake likelihoods data
likelihoods <- read.csv("example_100_fake_likelihood_topic.csv")
```

Inspect list of words to be passed to LDA:

```
# set out lexicon equal to the first column of the likelihoods data and inspecting its structure
lexicon_content  <- as.character(likelihoods[ ,1] )
str(lexicon_content)
```

3. Unsupervised Learning: Latent Dirichlet Allocation (LDA)

```
 # Put processed reviews in corpus format
corpus_review <- VCorpus(VectorSource(reviews_raw$processed_text))

# Creates the document term matrix that will be passed to the LDA function
dtm <- DocumentTermMatrix(corpus_review,
                          control = list(stemming = FALSE,  # Which is also the default
                                         language = "english",
                                         removePunctuation = TRUE,
                                         removeNumbers = TRUE,
                                         dictionary = as.character(lexicon_content)) )

# Inpsecting the structure of the DocumentTermMatrix
str(dtm)
```

```
## List of 6
##  $ i       : int [1:15936] 1 1 1 1 1 1 1 1 1 1 ...
##  $ j       : int [1:15936] 12 14 22 23 32 36 39 40 43 48 ...
##  $ v       : num [1:15936] 1 1 1 1 1 2 1 1 4 1 ...
##  $ nrow    : int 1000
##  $ ncol    : int 100
##  $ dimnames:List of 2
##   ..$ Docs : chr [1:1000] "1" "2" "3" "4" ...
##   ..$ Terms: chr [1:100] "action" "also" "anatomy" "argument" ...
##  - attr(*, "class")= chr [1:2] "DocumentTermMatrix" "simple_triplet_matrix"
##  - attr(*, "weighting")= chr [1:2] "term frequency" "tf"
```

```
# Useful info on the DocumentTermMatrix and its parameters
?DocumentTermMatrix()
```

```
## starting httpd help server ... done
```

```
?termFreq()   # more info on the control settings used within the dtm
```

Next, we will set the LDA parameters. k is the number of topics we ask LDA to estimate. In supervised learning, we set that to 3. In this example, we arbitrarily set k at 10. Seed is for replicability (i.e., obtain the same random number every time the code is run). Burn-in and number of iterations are for convergence of the Markov chains in the Gibbs sampler (MCMC-based inference is outside of the scope of this course and not required for the assignment - just use the default values below.) In the unlikely case you have warnings re: not convergence, you can increase ITER to 2k or 4k.

```
# LDA parameters
seed <- 20080809
burnin <-  1000
iter <- 1000
k <- 10
```

Tip: choosing which k to use in LDA is a model selection problem. Typically, the best approach is to compute a model for each level of k, save the model log-likelihood (produced by applying the function logLik() to the model, stored in model_lda) and choosing the k that produced the highest log-likelihood.

Next, we will run the LDA and save the model. The model produced by LDA() is an object of class LDA (page 11 of https://cran.r-project.org/web/packages/topicmodels/topicmodels.pdf). This class includes the topics, the log-likelihood, and a lot more. To extract this information, it is needed to use functions listed in page 2 of the said pdf, as shown in the code below.

```
#Create an LDA model using GIBBS sampling
model_lda <- LDA(dtm, k, method = "Gibbs", control = list(seed = seed, burnin = burnin, iter = iter) , r
save(model_lda , file = paste("LDA_model_" ,k,".RData" ,sep=""))
```

Inspect posteriors.

```
#posterior probabilities per document by topic
posteriors_lda <- posterior(model_lda)$topics
str(posteriors_lda)
```

```
##  num [1:1000, 1:10] 0.1392 0.1158 0.0545 0.0515 0.0909 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:1000] "1" "2" "3" "4" ...
##   ..$ : chr [1:10] "1" "2" "3" "4" ...
```

```
posteriors_lda[999,]
```

```
##          1          2          3          4          5          6          7
## 0.07692308 0.10769231 0.09230769 0.10769231 0.15384615 0.07692308 0.10769231
##          8          9         10
## 0.10769231 0.07692308 0.09230769
```

4. Unsupervised Learning: word embeddings

Our word embedding example has three steps. First, run word2vec to train a model using the training data split. Second, it uses the trained model to analyze the prediction data split. Third, it uses the constructed variables to forecast box office.

Step 1 - Training Step

```
# Obtain the column with the reviews and convert it to lower case
x <-  reviews_raw$full_text
x <- tolower(x)

# TODO: use a split of the data here (say 50%) instead of the entire dataset
```

```r
# number of topics in Word2Vec
total_topics_word2vec <- 10

# Train
model <- word2vec(x = x, type = "cbow", dim = total_topics_word2vec, iter = 20)
embedding <- as.matrix(model)
```

Step 2 - Construct variables from word embeddings

Similar to tutorial 1.1, we loop over all reviews. For a refresher on for loops in R you can visit: https://www.dataquest.io/blog/for-loop-in-r/. If you'd like more practice with for loops, try exercises 1, 3, 6, 9, 14, and 18 from https://www.w3resource.com/r-programming-exercises/control-structure/index.php.

```r
# TODO: Use the other split of the data here (say 50%) instead of the entire dataset

# Create an empty matrix to store the posteriors
posteriors_w2v <- matrix(0, nc = total_topics_word2vec, nr = total_reviews)

# Loop over all reviews
for (k in 1:total_reviews )
{
   # 2.1 get a review and tokenize it - identify the words, separately
   tokenized_review <- unlist(strsplit(reviews_raw$full_text[[k]],"[^a-zA-Z0-9]+"))

   # 2.2 get the word vectors per review using predict()
   embedding_review <- predict(model, tokenized_review, type = "embedding")

  # 2.3 compute mean across all words in the review
   posteriors_w2v[k,] <- apply(embedding_review, 2, mean, na.rm=TRUE)
}
```

Tip: for the above data splits, mind the time. Best to train in a split that temporarily precedes the prediction split

Step 3 - Use the constructed variables to forecast

```r
# prepare the constructed variables for analysis
log_BO   <- log(as.numeric(gsub(",", "",reviews_raw$first_week_box_office)))
data_reg <- cbind(c(log_BO), posteriors_w2v)
colnames(data_reg) <- c("LogBoxOffice", paste("w2v_", as.character(1:total_topics_word2vec), sep=""))

# forecast
w2v_BO_lm<- lm(LogBoxOffice ~ posteriors_w2v,  data=as.data.frame(data_reg))
summary(w2v_BO_lm)
```

```
##
## Call:
## lm(formula = LogBoxOffice ~ posteriors_w2v, data = as.data.frame(data_reg))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.7862 -0.9225 -0.1424  0.9144  4.4688
##
```

```
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)         5.3964     1.3601   3.968 7.79e-05 ***
## posteriors_w2v1     8.5180     1.0582   8.050 2.37e-15 ***
## posteriors_w2v2    -5.4055     1.2933  -4.180 3.18e-05 ***
## posteriors_w2v3    -0.3667     0.3794  -0.967 0.334020
## posteriors_w2v4    -5.9063     0.4099 -14.408  < 2e-16 ***
## posteriors_w2v5    -4.4571     1.3177  -3.382 0.000747 ***
## posteriors_w2v6     7.3408     0.6210  11.821  < 2e-16 ***
## posteriors_w2v7    -1.9601     0.3486  -5.624 2.43e-08 ***
## posteriors_w2v8     8.4111     0.9909   8.488  < 2e-16 ***
## posteriors_w2v9    -4.3701     0.9273  -4.713 2.80e-06 ***
## posteriors_w2v10   -2.1034     0.4963  -4.238 2.46e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.382 on 989 degrees of freedom
## Multiple R-squared:  0.5496, Adjusted R-squared:  0.545
## F-statistic: 120.7 on 10 and 989 DF,  p-value: < 2.2e-16
```

```
# write.csv(data_reg, "data_reg.csv")
```