DATA643 Final

V. Nguy

July 17, 2016

1 Data

http://eigentaste.berkeley.edu/dataset/

I will be using the data from Jester, an online Joke Recommender System.

1.1 Data Aquisition

There are three datasets from the Jester Online Joke Recommender System from (1999-2012). In the interest of time, only the third dataset will be used for the ratings information. The first dataset have data that is fairly unstructured and would require a lot of preprocessing. In addition to the preprocessing, the dataset only contains only 100 jokes. In contrast, both Dataset 2 and 3 contains ratings information for 150 jokes. Dataset 2 will be used because it contains the actual jokes. Dataset 3 contains ratings information in a format that is already in a user-item format.

Dataset 2
Dataset 3

The first step in to extract both dataset 2 and 3. In Dataset 2, we will read in the 'jester_items.dat' file. This file contains unstructured data representing the jokes that are being rated.

In Dataset 3, we will be using the 'jesterfinal151cols.xls' file. The data file was opened in Microsoft Excel file and saved as a csv file before being loaded.

```r
#Ratings information from Dataset3
jester <- read.csv("jester_dataset_3/jesterfinal151cols.csv", header = FALSE)
#jester <- read.csv("jester_dataset_3/jester.csv", header = FALSE)

#Adding Column and row names since the data file does not contain the information
#Columns are jokes and rows are users
colnames(jester) <- c("jokes_rated", 1:150)
rownames(jester) <- 1:nrow(jester)

#Jokes information from Dataset2
jokes <-  readLines(con="jester_dataset_2/jester_items.dat", n=-1)
```

1.2 Data Preparation

1.2.1 Jokes

The jokes data was unstructured data. Extra processing was needed to pull in the data in a format we can use. Lets take a look at the jokes data.

```r
head(jokes, n=10)
```

```
##  [1] "1:"
##  [2] "<p>"
##  [3] "A man visits the doctor. The doctor says, &quot;I have bad news for you. You have cancer and Al
##  [4] "<br />"
##  [5] "The man replies, &quot;Well, thank God I don&#039;t have cancer!&quot;"
##  [6] "</p>"
##  [7] ""
##  [8] "2:"
##  [9] "<p>"
## [10] "This couple had an excellent relationship going until one day he came home from work to find hi
```

We can see the jokes are seperated by numbered section, 1:, 2:, . . . , 150:
We will use that pattern as a seperator between jokes.

```r
#Create Empty vector to store jokes
jokes_parsed <- vector()

#Vector to temporarily store the pieces of the jokes
temp <- vector()
```

This pointer will be used to track our position while we iterate through the jokes dataset. Since we are iterating over the dataset 150 times, we want to pick up where we last left off from the previous iteration.

```r
pointer <- 1
```

For each 'N:' identifying tag, iterate and concatenate the pieces together until we see the next 'N:' tag Once we see marker, write the temp data into the container. Reset the temp vector and move the pointer to the next position.

```r
for(n in 1:150) {
  for(line in pointer:length(jokes)) {
    if (jokes[line] != paste0(n, ':')) {
      temp <- paste0(temp, jokes[line])
    }
    else {
      jokes_parsed <- c(jokes_parsed, temp)
      pointer <- line+1
      temp <- vector()
      break
    }
  }
}
#Get the last Joke
for(line in pointer:length(jokes)) {
  if (jokes[line] != paste0(n, ':')) {
    temp <- paste0(temp, jokes[line])
  }
}
jokes_parsed <- c(jokes_parsed, temp)
```

As we can see, there are a few html tags in the data. Lets clean up the data a bit.

```
jokes_parsed[1]
```

```
## [1] "<p>A man visits the doctor. The doctor says, &quot;I have bad news for you. You have cancer and
```

```
#Patterns we want to delete
del_1 <- '<p>'
del_2 <- '</p>'
del_3 <- '<br />'

jokes_parsed <- gsub(del_1, '', jokes_parsed)
jokes_parsed <- gsub(del_2, '', jokes_parsed)
jokes_parsed <- gsub(del_3, '', jokes_parsed)
```

There are still a few html codes in the document. Lets remove those.

```
#http://stackoverflow.com/questions/5060076/convert-html-character-entity-encoding-in-r
unescape_xml <- function(str){
  xml2::xml_text(xml2::read_xml(paste0("<x>", str, "</x>")))
}

for(i in 1:length(jokes_parsed)) {
  jokes_parsed[i] <- unescape_xml(jokes_parsed[i])
}

jokes_parsed[1]
```

```
## [1] "A man visits the doctor. The doctor says, \"I have bad news for you. You have cancer and Alzhei
```

1.2.2 Ratings

The first column of the data contains a count of how many jokes were rated by the user. The count is not being used so it will be removed. It will be saved just in case it is needed in the future.

```
jester_cnt <- jester[1]
jester <- jester[-1]

#Not sure why, but these two records were causing errors so I removed them.
jester <- jester[-1466,]
jester <- jester[-50691,]
```

The ratings are real values ranging from -10.00 to +10.00. A 99 represents a null rating. To create a sparse matrix, the null values need to be represented by 0. We convert 99 to 0's

```
library(Matrix)

jester[jester==99] <- 0
jester[jester==NA] <- 0

jester_sparse <- Matrix(as.matrix(jester, sparse=TRUE))
```

```r
dim(jester)
```

```
## [1] 50690   150
```

2 Dimensionality Reduction

```r
k<-50
svd <- svd(jester, nu=k, nv=k)
svd$d <- svd$d[1:k]

aveUser <- apply(jester_sparse, 1, mean, na.rm=TRUE)

pred <- aveUser + (svd$u %*% sqrt(diag(svd$d))) %*% (sqrt(diag(svd$d))%*%t(svd$v))

sqrt(mean((pred-jester_sparse)^2, na.rm=TRUE))
```

```
## [1] 1.631914
```

```r
pred[1:5, 1:5]
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]
## [1,]  0.8658333  0.8658333  0.8658333  0.8658333  0.6282497
## [2,]  0.8475000  0.8475000  0.8475000  0.8475000  0.7332906
## [3,] -0.8181250 -0.8181250 -0.8181250 -0.8181250 -0.8975135
## [4,]  1.4214583  1.4214583  1.4214583  1.4214583  1.3273533
## [5,] -0.3970833 -0.3970833 -0.3970833 -0.3970833 -0.4326306
```

```r
jester[1:5,1:5]
```

```
##   1 2 3 4        5
## 1 0 0 0 0  0.21875
## 2 0 0 0 0 -9.68750
## 3 0 0 0 0 -9.84375
## 4 0 0 0 0  6.90625
## 5 0 0 0 0 -0.03125
```

3 Topic Modeling

Topic Modeling will be used to automatically classify the jokes. The jokes will be preprocessed to remove unwanted characters such as punctuations and extra spaces.

```r
library(tm)
```

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

```
#Pre Processing
docs_jokes <- tolower(jokes_parsed)
docs_jokes <- gsub("'", "", docs_jokes)  # remove apostrophes
docs_jokes <- gsub("[[:cntrl:]]", " ", docs_jokes)  # replace control characters with space
docs_jokes <- gsub("\\s+", " ", docs_jokes) #remove extra spaces between words.
docs_jokes <- gsub("^[[:space:]]+", "", docs_jokes) # remove whitespace at beginning of documents
docs_jokes <- gsub("[[:space:]]+$", "", docs_jokes) # remove whitespace at end of documents
docs_jokes <- gsub("[[:punct:]]", " ", docs_jokes)
```

Stop words are common words that are filtered out before (or after) the text is processed.

```
#Create a Corpus from the vector of jokes
docs <- Corpus(VectorSource(docs_jokes))
docs <- tm_map(docs, removeWords, stopwords('SMART'))

dtm <- DocumentTermMatrix(docs)

#The jokes dont have any titles so we will just assgin them a number
rownames(dtm) <- 1:150
```

K is the number of topics. The number of topics have to be determined before LDA (Latent Dirichlet allocation) can be executed. Here, we arbitrarily chose 5.

```
library(topicmodels)
k<-5

lda <- LDA(dtm, k)

#Topic assignments for the jokes
lda_topics <- as.matrix(topics(lda))

#top 10 terms in each topic
lda_terms <- as.matrix(terms(lda,10))

#probabilities associated with each topic assignment
topic_probs <- as.data.frame(lda@gamma)
```

4 Topic Weighing

A topic matrix will be created which will remove the ratings for each joke. It will replace the ratings with the topic assignment from the topic modeling.

```
jester_topic <- jester

jester_topic[1:15, 1:15]
```

```
##    1 2 3 4        5 6        7        8 9 10 11 12        13 14       15
## 1  0 0 0 0  0.21875 0 -9.28125 -9.28125 0  0  0  0 -6.78125  0  0.87500
## 2  0 0 0 0 -9.68750 0  9.93750  9.53125 0  0  0  0  9.93750  0  0.40625
## 3  0 0 0 0 -9.84375 0 -9.84375 -7.21875 0  0  0  0 -2.03125  0 -9.93750
## 4  0 0 0 0  6.90625 0  4.75000 -5.90625 0  0  0  0 -0.40625  0 -4.03125
## 5  0 0 0 0 -0.03125 0 -9.09375 -0.40625 0  0  0  0  7.50000  0 -7.21875
## 6  0 0 0 0 -2.90625 0 -2.34375 -0.50000 0  0  0  0 -0.96875  0  2.25000
## 7  0 0 0 0  6.21875 0 -7.43750 -0.81250 0  0  0  0 -3.43750  0  0.53125
## 8  0 0 0 0  8.25000 0  9.00000  8.87500 0  0  0  0  9.75000  0  9.37500
## 9  0 0 0 0 -5.75000 0  0.28125  0.78125 0  0  0  0  8.28125  0  3.59375
## 10 0 0 0 0 -7.15625 0 -5.90625 -0.09375 0  0  0  0 -5.50000  0 -6.00000
## 11 0 0 0 0 -5.18750 0  1.50000  6.21875 0  0  0  0  2.87500  0 -2.71875
## 12 0 0 0 0 -9.31250 0 -9.37500 -9.43750 0  0  0  0 -1.87500  0 -2.15625
## 13 0 0 0 0 -5.00000 0 -3.90625 -3.71875 0  0  0  0  0.00000  0 -0.93750
## 14 0 0 0 0  0.46875 0 -9.93750  1.56250 0  0  0  0 -9.90625  0 -9.93750
## 15 0 0 0 0  6.28125 0  8.15625  9.93750 0  0  0  0 -8.96875  0 -8.37500
```

```
jester_topic[jester_topic!=0] <- 1

jester_topic[1:15, 1:15]
```

```
##    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
## 1  0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 2  0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 3  0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 4  0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 5  0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 6  0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 7  0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 8  0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 9  0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 10 0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 11 0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 12 0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 13 0 0 0 0 1 0 1 1 0  0  0  0  0  0  1
## 14 0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
## 15 0 0 0 0 1 0 1 1 0  0  0  0  1  0  1
```

For each user, the counts of the movies, by topic will be calculated. This will be used to determine what percentage of the jokes the user rated belonged to which topic.

```
#Get movies with different topics
user_tp1 <- jester_topic[,lda_topics==1]
user_tp2 <- jester_topic[,lda_topics==2]
user_tp3 <- jester_topic[,lda_topics==3]
user_tp4 <- jester_topic[,lda_topics==4]
user_tp5 <- jester_topic[,lda_topics==5]

user_sum_tp1 <- apply(user_tp1, 1, sum, na.rm=TRUE)
user_sum_tp2 <- apply(user_tp2, 1, sum, na.rm=TRUE)
user_sum_tp3 <- apply(user_tp3, 1, sum, na.rm=TRUE)
user_sum_tp4 <- apply(user_tp4, 1, sum, na.rm=TRUE)
user_sum_tp5 <- apply(user_tp5, 1, sum, na.rm=TRUE)

user_tp_total <- user_sum_tp1 + user_sum_tp2 + user_sum_tp3 + user_sum_tp4 + user_sum_tp5
user_perc_tp <- cbind(user_sum_tp1/user_tp_total,user_sum_tp2/user_tp_total,user_sum_tp3/user_tp_total,
```

The user topic percentage is a user preference matrix. How much each user prefer each topic. Once we know how much each user prefers a topic, the dot product is taken between the users preference and joke breakdown between of topics; how much each joke represent each topic. That weight is then added to the prediction matrix calculated earlier with SVD.

```
user_topic_pref <- user_perc_tp %*% t(topic_probs)
dim(user_topic_pref)
```

```
## [1] 50690   150
```

```
dim(pred)
```

```
## [1] 50690   150
```

```
dim(user_topic_pref)
```

```
## [1] 50690   150
```

```
pred_weighted <- pred+user_topic_pref

pred_weighted[1:10, 1:7]
```

```
##            [,1]       [,2]        [,3]       [,4]         [,5]       [,6]
## 1     1.0161808  1.0658333  1.14835806  1.0822699  0.910774454  1.0159292
## 2     0.9365121  1.0826563  1.16983078  1.0242954  1.055621400  1.0240157
## 3    -0.6999056 -0.6415626 -0.35016234 -0.8153641 -0.429550859 -0.5828986
## 4     1.5681727  1.6165993  1.73739552  1.6647547  1.643290513  1.5192157
## 5    -0.3221341 -0.2856252  0.04498921 -0.2116936  0.009441911 -0.2118697
## 6     0.4533396  0.5398110  0.60416865  0.4965455  0.543938137  0.4097041
## 7     1.3435188  1.3836174  1.50370024  1.4036206  1.415579236  1.3231115
## 8     0.5565984  0.6223955  1.15132704  0.5579692  1.263957474  0.8882860
## 9     1.4217498  1.4699975  1.54683493  1.4602882  1.388531875  1.3828134
## 10    0.1551400  0.3205468  0.69210140  0.2386938  0.565754772  0.3620674
##            [,7]
## 1     -8.257137
## 2     10.828855
## 3    -10.480301
## 4      6.388841
## 5     -9.382385
## 6     -1.957791
## 7     -6.134396
## 8     10.005598
## 9      1.669645
## 10    -5.535628
```

```
jester[1:10,1:7]
```

```
##    1 2 3 4        5 6        7
## 1  0 0 0 0  0.21875 0 -9.28125
## 2  0 0 0 0 -9.68750 0  9.93750
## 3  0 0 0 0 -9.84375 0 -9.84375
## 4  0 0 0 0  6.90625 0  4.75000
## 5  0 0 0 0 -0.03125 0 -9.09375
## 6  0 0 0 0 -2.90625 0 -2.34375
## 7  0 0 0 0  6.21875 0 -7.43750
## 8  0 0 0 0  8.25000 0  9.00000
## 9  0 0 0 0 -5.75000 0  0.28125
## 10 0 0 0 0 -7.15625 0 -5.90625
```

```
sqrt(mean((pred_weighted-jester_sparse)^2, na.rm=TRUE))
```

```
## [1] 1.699261
```

5 Conclusion

With dimentionality reduction the error rate was pretty low. Once we added the topic weights, the error rate went up. There are a few possible reasons for affect.

1) The number of topics (k) was chosen arbitrarily. Better choices for k could be possible.
2) The ratings matrix used 99 as a null indicator since valid ratings were from -10 to 10. This meant that 0 was a valid rating. When processing the ratings matrix to create a sparse matrix, I replaced the 99 with 0 and created a sparse matrix. This had the adverse effect of deleting some of the '0' ratings.
3) The jokes did not contain many words. This could be a problem for accurate topic modeling.

```
summary(rowSums(as.matrix(dtm)))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4.00   10.00   24.00   27.94   41.00   93.00
```

The amount of words in the jokes might not be enough to categorize the jokes into appropriate topics. There was an average of about 30 words per joke after processing.