Project 4 – Movie Review Sentiment Analysis

Dataset: Project4_data.tsv from CS598 resource page on Piazza
Code reference: https://cran.r-project.org/web/packages/text2vec/vignettes/text-vectorization.html
Machine: Azure Linux VM (Standard DS3 v2 Promo (4 vcpus, 14 GB memory))

Summary

In Project 4, we were asked to build a binary classification model to predict sentiments of movie reviews. The dataset is subset of IMDB movie reviews from Bag of Words Meets Bags of Popcorn totaling 50,000 rows. Each row contains 3 columns: 1) ***new_id*** being the ID of the review, 2) ***sentiment*** value of 1 (positive) and 0 (negative) and 3) ***review*** contains the review text.

text2vec and ***tokenizers*** packages were the primary text mining and processing packages used to stem, tokenize, vectorize review texts. The glmnet package was used to fit a logistic regression with L1 (lasso) penalty to build a custom bag of n-grams after cleaning, stemming and pruning the entire set of review texts. The size of **custom vocabulary** is **2943**. A logistic regression with L2 (ridge) penalty was leveraged as the sentiment predictive model to evaluate against all 3 test sets (based on Project4_splits.csv). The model produced AUC of **0.9828 for split 1, 0.9829 for split 2 and 0.9832 for split 3**. Note that, as part of the submission collateral, along with mymain.R, I have submitted the R code (createVocab.R) showing the routine to create the custom vocabulary. The myVocab.txt file contains the custom vocabulary that will be consumed by mymain.R. Result_1.txt, Result_2.txt and Result_3.txt contain the id and probability (positive sentiment) of each split. The average runtime of the evaluation code (cv.glmnet and predict) per split is 17.22 seconds on an Azure Linux VM (Standard DS3 v2 Promo (4 vcpus, 14 GB memory)).

Building custom vocabulary

Building the custom vocabulary of n-grams is a manual and repetitive process as the goal is to have terms that are highly effective in predicting the outcome. The objective was to have the vocabulary size less than 3000 while producing an AUC of higher than 0.96. To find the most important terms, I leveraged cv.glmnet to fit a logistic regression with L1 penalty. The intuition here is that Lasso will zero out any terms (variables) that are not contributing to the prediction. Hence, terms (variables) with non-zero coefficient would be useful to the predictive model; these terms would be preserved as the vocabulary. To get to the desired vocabulary size, I repeatedly added new stop words and pruned rare and highly frequent words. Below are the key steps to construct the custom vocabulary.

1. Review texts from the whole dataset were first cleaned by removing special symbols and punctuations.

```
all$review <- gsub('<.*?>', ' ', all$review)
all$review <- tolower(gsub('[[:punct:]]', '', all$review))
```

2. Review texts were transformed into lowercasing. Then, stop words were removed; the texts were stemmed and tokenized using ***tokenize_word_stems*** function from ***tokenizers*** package.
3. Next, **n-grams up to 4-gram**s were generated from each of the review text using the ***create_vocabulary*** function from ***text2vec*** package.

```
vocab.all <- create_vocabulary(itr.all, ngram=c(1L,4L), stopwords = stop_words)
```

4. The vocabulary was then pruned to remove highly infrequent and frequent words. As shown below, ***prune_vocabulary*** function from ***text2vec*** package was used to remove any terms lower than 10 counts, keeping terms with 99.9% to 50.0% of document frequency.

```
vocab.pruned <- prune_vocabulary(vocab.all, term_count_min=10, doc_proportion_max = 0.5, doc_proportion_min =
0.001)
```

5. Now, we have an initial set of vocabulary to create the document-term matrix using ***create_dtm.***

```
pruned.vectorizer <- vocab_vectorizer(vocab.pruned)
dtm_all <- create_dtm(itr.all, pruned.vectorizer)
```

6. Next, cv.glmnet was used to fit a logistic regression with L1 penalty and 4-fold cross validations, i.e. NFOLDS=4. The threshold was set to 0.001 and max iteration set to 1000.

```
glmnet_classifier = cv.glmnet(x = dtm_all, y = all[['sentiment']],
                              family = 'binomial',
                              # L1 penalty
                              alpha = 1,
                              # use area under ROC curve as the measure
                              type.measure = "auc",
                              nfolds = NFOLDS,
                              thresh = 1e-3,
                              maxit = 1e3)
```

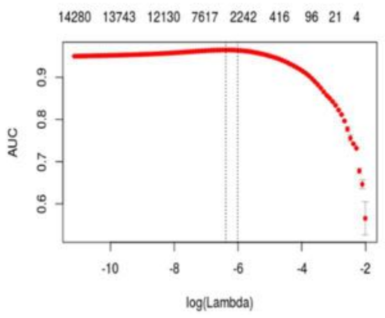
7. The terms with non-zero coefficient were inspected. To avoid overfitting, coefficients for lambda.1se was used. As mentioned above, terms (variables) with non-zero coefficient would be useful to the predictive model; these terms would be preserved as the vocabulary.

```
tmp_coeffs <- coef(glmnet_classifier, s="lambda.1se")
words_flg <- as.matrix(tmp_coeffs[-1,1] != 0)
words <- vocab.pruned$term[words_flg]
```

8. I manually inspected the 'effective' terms particularly the ones with high term counts. I discovered that they were additional stop words such as *"and", "to","be","that","this", "so", "a"* that could be removed. These words were then added to the stop words set and step 2 to 7 were repeated until I get the desired vocabulary size. Below shows the final set of stop words and the glmnet curve. Note that the vocabulary size is **2943**.

```
stop_words = c("i", "me", "my", "myself",
          "we", "our", "ours", "ourselves",
          "you", "your", "yours",
          "their", "they", "his", "her",
          "she", "he",
          "is", "was", "are", "were", "and", "to","be","that","this", "so","a")
```

```
> plot(glmnet_classifier)
> length(words)
[1] 2943
```



Model and Performance

In mymain.R, the methods described above were used to remove special symbols and punctuations from the review texts in the test and train sets. Similar approach (as noted above) was also taken to get rid of stop words, stem and tokenize. The custom vocabulary file, myvocab.txt, containing the 2943 terms was read in to create the document-term matrix for train and test set respectively. I decided to use a logistic regression with L2 (ridge) penalty as the sentiment predictive model. The rational here is that all terms form the custom vocabulary were useful and important to the prediction model. Below shows the cv.glmnet logistic regression with L1 penalty and 4-fold cross validations, i.e. NFOLDS=4. The threshold was set to 0.001 and max iteration set to 1000.

```
glmnet_classifier1 <- cv.glmnet(x = dtm_tr, y = train[['sentiment']],
                        family = 'binomial',
                        alpha = 0,
                        type.measure = "auc",
                        nfolds = NFOLDS,
                        thresh = 1e-3,
                        maxit = 1e3)
```

Lambda.min was used to perform the prediction.

```
preds <-predict(glmnet_classifier1, dtm_tt, s="lambda.min", type = 'response')[,1]
```

Below shows the AUC for the respective test splits and the vocabulary size.

Split	**AUC**
Split 1	**0.982793119999999**
Split 2	**0.98293651504002**
Split 3	**0.983192898948973**
Vocab Size	**2943**

Runtime

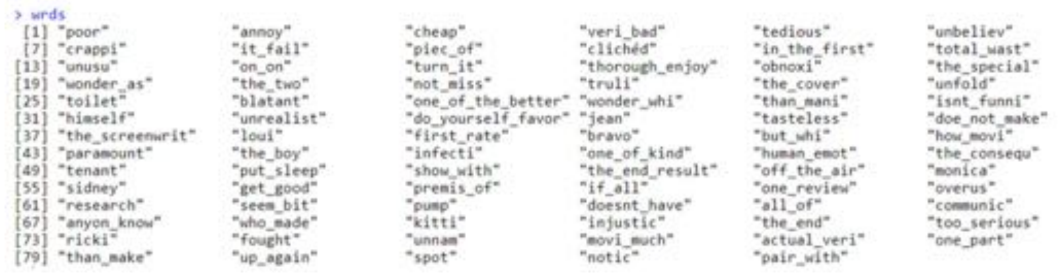
The cross validation and prediction runtime for the 3 models averaged 17.22 secs on an Azure Linux VM (Standard DS3 v2 Promo (4 vcpus, 14 GB memory)).

	Runtime (secs)
Test 1	16.87589
Test 2	17.75806
Test 3	17.01346
Average	**17.2158**

Further Analysis and Next Steps

The custom vocabulary created is a finite bag of n-grams that is applied to all 3 train and test sets. It is intuitive that not all n-grams would be applicable and relevant in terms of sentiment context for each document(record) and each split set. Similarly, the predictive power of each n-gram differs among reviews and splits. This is a major factor contributing to misclassifications.

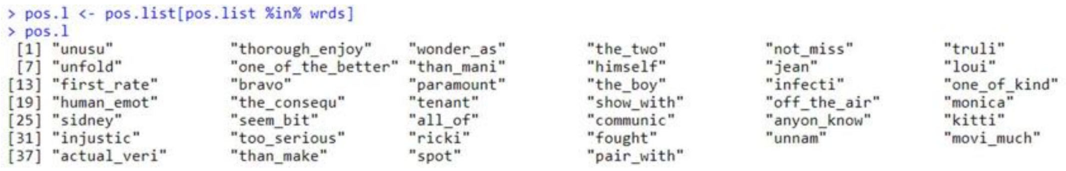
Below shows the n-grams applied to one of the misclassified test document in split# 3; the true sentiment for the document was 0 but the model predicted 1. A glance of the ngrams wouldn't allow us to understand why this document was misclassified.



```
> wrds
 [1] "poor"            "annoy"       "cheap"             "veri_bad"         "tedious"       "unbeliev"
 [7] "crappi"          "it_fail"     "piec_of"           "cliched"          "in_the_first"  "total_wast"
[13] "unusu"           "on_on"       "turn_it"           "thorough_enjoy"   "obnoxi"        "the_special"
[19] "wonder_as"       "the_two"     "not_miss"          "truli"            "the_cover"     "unfold"
[25] "toilet"          "blatant"     "one_of_the_better" "wonder_whi"       "than_mani"     "isnt_funni"
[31] "himself"         "unrealist"   "do_yourself_favor" "jean"             "tasteless"     "doe_not_make"
[37] "the_screenwrit"  "loui"        "first_rate"        "bravo"            "but_whi"       "how_movi"
[43] "paramount"       "the_boy"     "infecti"           "one_of_kind"      "human_emot"    "the_consequ"
[49] "tenant"          "put_sleep"   "show_with"         "the_end_result"   "off_the_air"   "monica"
[55] "sidney"          "get_good"    "premis_of"         "if_all"           "one_review"    "overus"
[61] "research"        "seem_bit"    "pump"              "doesnt_have"      "all_of"        "communic"
[67] "anyon_know"      "who_made"    "kitti"             "injustic"         "the_end"       "too_serious"
[73] "ricki"           "fought"      "unnam"             "movi_much"        "actual_veri"   "one_part"
[79] "than_make"       "up_again"    "spot"              "notic"            "pair_with"
```

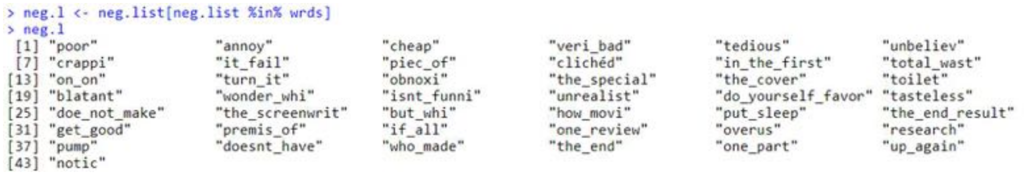
Using two-sample T-statistic, I segmented the ngrams for the split# 3 train set into positive and negative ngrams. Now, mapping the list above to the positive and negative n-gram list, I get the following breakdown for the misclassified document.

Positive n-grams:



```
> pos.l <- pos.list[pos.list %in% wrds]
> pos.l
 [1] "unusu"         "thorough_enjoy"    "wonder_as"  "the_two"    "not_miss"    "truli"
 [7] "unfold"        "one_of_the_better" "than_mani"  "himself"    "jean"        "loui"
[13] "first_rate"    "bravo"             "paramount"  "the_boy"    "infecti"     "one_of_kind"
[19] "human_emot"    "the_consequ"       "tenant"     "show_with"  "off_the_air" "monica"
[25] "sidney"        "seem_bit"          "all_of"     "communic"   "anyon_know"  "kitti"
[31] "injustic"      "too_serious"       "ricki"      "fought"     "unnam"       "movi_much"
[37] "actual_veri"   "than_make"         "spot"       "pair_with"
```

Negative n-grams:



```
> neg.l <- neg.list[neg.list %in% wrds]
> neg.l
 [1] "poor"          "annoy"          "cheap"      "veri_bad"       "tedious"           "unbeliev"
 [7] "crappi"        "it_fail"        "piec_of"    "cliched"        "in_the_first"      "total_wast"
[13] "on_on"         "turn_it"        "obnoxi"     "the_special"    "the_cover"         "toilet"
[19] "blatant"       "wonder_whi"     "isnt_funni" "unrealist"      "do_yourself_favor" "tasteless"
[25] "doe_not_make"  "the_screenwrit" "but_whi"    "how_movi"       "put_sleep"         "the_end_result"
[31] "get_good"      "premis_of"      "if_all"     "one_review"     "overus"            "research"
[37] "pump"          "doesnt_have"    "who_made"   "the_end"        "one_part"          "up_again"
[43] "notic"
```

It is quite clear that the terms in negative n-grams should have higher predictive power than the positive terms. Furthermore, some of the terms in both lists didn't seem to carry strong positive or negative context. Aside from removing more stop words or words that doesn't carry contextual meaning, one immediate next step to improve the prediction model is to leverage TF-IDF transformation that would increase the weights of terms that are specific to a document or set of documents and decrease the weights of terms that are highly frequent in most documents.

Ordering and distance of terms in sentences bring forth a more complete contextual meaning of the author. Bag-of-ngrams only considers word order in short context; it loses ordering in higher dimension and thus causes context sparsity. Furthermore, Bag-of-ngrams does not provide semantic distance between terms. One approach to solve this deficiency is to leverage paragraph vector based on the paper by Le and Mikolov (https://cs.stanford.edu/~quocle/paragraph_vector.pdf). Paragraph vector keeps the representation of sentences, paragraphs and documents that embody contextual meaning. As discussed in the paper, paragraph vector out performs bag-of-ngrams models for text representation.