**Project 3: What we have tried (I)**

#626

Project 3

**Notes:** For the Python code below, I'm not certain if it's error-free. Please inform me if you come across any issues.

I start with the first split.

**Step 1**: Load the training data and clean the html tags.

train = read.table("train.tsv",

stringsAsFactors = FALSE,

header = TRUE)

train$review = gsub('&lt;.\*?&gt;', ' ', train$review)

**Step 2**: Use R package text2vec to construct DT (DocumentTerm) matrix (maximum 4-grams). The default vocabulary size (i.e., # of columns of dtm\_train) is more than 30,000, bigger than the sample size n = 25,000.

I treat the following words as stop words: i, me, my, myself, we, our, ours, ourselves, you, your, yours, their, they, his, her, she, he, a, an, and, is, was, are, were, him, himself, has, have, it, its, the, us.

stop\_words = c("i", "me", "my", ...)

it\_train = itoken(train$review,

preprocessor = tolower,

tokenizer = word\_tokenizer)

tmp.vocab = create\_vocabulary(it\_train,

stopwords = stop\_words,

ngram = c(1L,4L))

tmp.vocab = prune\_vocabulary(tmp.vocab, term\_count\_min = 10,

doc\_proportion\_max = 0.5,

doc\_proportion\_min = 0.001)

dtm\_train = create\_dtm(it\_train, vocab\_vectorizer(tmp.vocab))

**Python** code for Step 1

import pandas as pd

train = pd.read\_csv("train.tsv", sep='\t', header=0, dtype=str)

train['review'] = train['review'].str.replace('&lt;.\*?&gt;', ' ', regex=True)

and for Step 2

from sklearn.feature\_extraction.text import CountVectorizer

stop\_words = ["i", "me", "my", "myself" ...]

vectorizer = CountVectorizer(

preprocessor=lambda x: x.lower(), # Convert to lowercase

stop\_words=stop\_words, # Remove stop words

ngram\_range=(1, 4), # Use 1- to 4-grams

min\_df=0.001, # Minimum term frequency

max\_df=0.5, # Maximum document frequency

token\_pattern=r'\b\w+\b' # Use word tokenizer

)

dtm\_train = vectorizer.fit\_transform(train['review'])

**Step 3**: use Lasso (with logistic regression) to trim the vocabulary size to 2K.

The output from glmnet comprises various sets of estimated beta values corresponding to different lambda values. I focused on the number of non-zero beta values (df) for each of these estimates. I picked the largest df among those less than 2,000, and stored the corresponding words in "myvocab".

Please note that in this context, a 'word' represents a term, which can include phrases involving multiple words, such as 'highly\_recommended' or 'most\_annoying,' as we use 1- to 4-grams.

Now, using this customized vocabulary, I proceeded with ridge regression on the five data splits. For each split, the initial part of my code resembled the following:

* **R**

vectorizer = vocab\_vectorizer(create\_vocabulary(myvocab,

ngram = c(1L, 2L)))

dtm\_train = create\_dtm(it\_train, vectorizer)

* **Python**

vectorizer = CountVectorizer(

ngram\_range=(1, 2) # Use 1- to 4-grams

)

vectorizer.fit(myvocab)

dtm\_train = vectorizer.transform(train['review'])

I experimented with Lasso, Ridge, and ElasticNet (with various alpha values). At times, I could achieved an accuracy of 0.96 for all five data splits.

I didn't try to reduce the vocab size further down.

I believe this strategy could yield better results if I construct the vocabulary using the entire dataset, including both the training and test data. Ideally, students should avoid using test data, but I acknowledge this might be difficult to enforce, so I've allowed it.

However, one aspect that has left me somewhat unsatisfied is that certain words from 'myvocab' do not appear to be relevant to sentiment analysis at all. Therefore, I am actively exploring alternative approaches to improve the relevance of the vocabulary.

**Project 3: What we have tried (II)**

#628

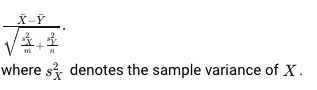
Project 3

As described at #626, I used a customized vocab of size 2K to achieve 0.96 AUC (for all five splits).

However, during my analysis, I noticed that some of the selected terms posed challenges for interpretation. To address this, I decided to filter the vocabulary to include only terms I could readily interpret. I employed a straightforward screening method: the **two-sample t-test**. This test compares one-dimensional observations from two groups, denoted as:



The goal is to determine whether the X population and the Y population share the same mean. The two-sample t-statistic is computed as:



Suppose we have m positive reviews and n negative reviews. For a given word, X\_i's represent the measurements associated with each of the positive reviews, and Y\_j's represent the measurements corresponding to each of the negative reviews. The goal is to assess the significance of differences in measurements between the two sentiment groups for each word via t-test.

I compute the t-statistics using the training data from the first split. Assume dtm\_train is the document\_term\_matrix (with over 30K cols) obtained as described in #626. Since dtm\_train is a large sparse matrix, in R, you can use commands from the R package slam to efficiently compute the mean and var for each column of dtm\_train.

I proceeded to rank the words based on the magnitude of their t-statistics and selected the top 2,000 words, categorizing them into two lists: positive words and negative words.

Top 50 **positive words**

[1] "great" "excellent" "best"

[4] "of\_best" "wonderful" "one\_of\_best"

[7] "perfect" "love" "superb"

[10] "amazing" "loved" "beautiful"

[13] "well" "favorite" "brilliant"

[16] "life" "must\_see" "highly"

[19] "also" "very" "fantastic"

[22] "one\_of" "performance" "beautifully"

[25] "both" "always" "enjoyed"

[28] "wonderfully" "very\_well" "well\_worth"

[31] "8\_10" "today" "highly\_recommend"

[34] "strong" "this\_great" "performances"

[37] "young" "touching" "10\_10"

[40] "years" "7\_10" "powerful"

[43] "highly\_recommended" "8" "perfectly"

[46] "definitely" "terrific" "moving"

[49] "well\_as" "gives"

Top 50 **negative words**

[1] "bad" "worst" "waste" "awful"

[5] "terrible" "worse" "waste\_of" "no"

[9] "boring" "of\_worst" "nothing" "poor"

[13] "minutes" "horrible" "stupid" "even"

[17] "acting" "so\_bad" "one\_of\_worst" "supposed"

[21] "at\_all" "poorly" "crap" "just"

[25] "why" "supposed\_to" "ridiculous" "avoid"

[29] "plot" "waste\_of\_time" "not\_even" "lame"

[33] "wasted" "don't" "script" "worst\_movie"

[37] "money" "cheap" "pointless" "thing"

[41] "waste\_time" "oh" "supposed\_to\_be" "annoying"

[45] "any" "dull" "laughable" "unless"

[49] "pathetic" "mess"

I also included words in the word list that never appeared in any of the positive reviews and words that never appeared in the negative reviews.

In my quest to further reduce the vocabulary size, I adopted a similar approach to what I outlined in #626. Specifically, I applied Lasso to the first split using this vocabulary and then aimed to reduce it to fewer than 1,000 terms—ultimately achieving a total of 980 terms. With this trimmed vocabulary, ridge regression consistently produced AUC scores above 0.96 for all five splits.

In prior semesters, some students reported difficulty in replicating the benchmark results strictly following the outlined procedure. I'd like to introduce an alternative approach that students might find valuable.

Lasso, especially when using lambda.min, can sometimes select spurious variables. To address this, you may want to explore an iterative technique. In this approach, the data is perturbed iteratively, and Lasso is applied. By doing so, we can identify which variables are consistently chosen. Below is the pseudocode outlining this approach:

T = 50;

for( i in 1:T){

# Randomly sample a subset of the data, e.g., 60%

# Apply Lasso on this subset

# Record selected variables using lambda.min

}

# Rank the 2K variables based on selection frequency

# Select the top M variables (where M is less than 1,000)

I developed this two-stage method several years ago when first introducing sentiment analysis as a course project for PSL. I later discovered that our approach bears similarities to the method outlined at

<http://statweb.stanford.edu/~tibs/superpc/tutorial.html>

In their method, they focus on regression, with the initial step being a univariate regression (Y ~ j-th feature), which is essentially akin to the t-test we employed, given that our response variable Y is binary.