Bin Dong
CS 2731
Homework 1
7 February 2019

Homework 1 Report

What I Done:

Before I begin pre-processing, I have split my dataset into 80/20. What I mean by 80/20 is that 80% of my dataset will be used for training and the remaining 20% will be used testing (this 20% is not included in my cross validation testing). Once I have split my dataset, I began to take out some common stop words as well as few punctuations like commas and periods. My stop word set are generally small so it shouldn't have a major impact. After tokenizing the dataset and getting rid of stopwords, I was able to build 2 sets of bag of words. One for Unigram Bag of Words (which uses vocab as features) and one for Bigram Bag of Words (which uses co-occurrence words as features). I modified the way I build my bigram bag of words. Instead of having the entire dataset and transforming it into a bigram dataset, I simply choose the top 22500 most frequently appeared bigrams out of the entire dataset and use that as my bigram bag of words (will talk about why I picked 22500 later). For each comment in the dataset I extract a binary feature (will talk about why I choose binary feature later). For each words in bag of words, I check if it exists in the comment. If it does, a Boolean value of 1 is appended to the list. If not, the Boolean value 0 is appended to the list. In the end, each comment will have a feature list of size 22500 for Bigram feature extraction and 9499 for Unigram feature extraction. Now, the Bigram bag of words isn't separate words in the bag but co-occurrence word. For example, if I have the sentence "This is an example", the bigram bag of words will contain [("This","is"), ("is", "an"), ("an", "example")].

Before the feature extraction, I also extracted the targeted toxicity level for my classifier. Since the data are extremely skewed (there are a lot more label 1 than there are for label 2, 3, and 4), I made a binary classifier which classifies if the comment is a 1 or not. (Basically toxic or not toxic where 1 = not toxic class and 2,3,4 are toxic class). This somewhat balances out the data being skewed. After targeted value extraction from the dataset and feature extraction from the dataset for each comments, I simply trained my logistic model on those data and compare it to Majority Baseline, Unigram Model, and Bigram Model.

The reason why I choose binary feature instead of word-count is because I have tried to use word-count. Instead of appending 1 or 0, I will append the number of times that specific word (or co-occurrence words) appeared in that specific comment. This, however, did not work very well at all and lowered my overall average accuracy by about 5-10% when doing cross-validation of 10 splits. When switching to binary, it increased my feature. My guess behind this is because since there are a lot of different features, having binary limits the total combination of features while having number count will increase the total combinations. Having word count may work for a much larger dataset but since our dataset only has about 1000 data, it does not work as well.

The reason why I choose the top 22500 instead of the any other number is because that number seems to work well for the given size of the dataset. If the numbers increases to 25000 or anything higher, the model will be very strict in its prediction and mostly always predict 1 (low or zero precision value and recall value). However, if we lower that number, the accuracy will be lowered as a result of overfitting because at this point, you may as well say that the model is simply guessing rather than making a good prediction.

What Is different between Unigram Model (vocab as feature) and My Modified Bigram Model?

The Unigram Model uses vocab as its feature. What this means is that it will tokenize the entire dataset and uses that as its feature. For each word in the tokenized dataset (bag of words), if the word contains in a specific comment, then 1 will be appended to the feature list that is associated with that comment. Otherwise, 0 will be appended to the feature list that is associated with that comment. For example, let's say the bag of words contain ["Hello", "world", "nice", "today"] and the comment is "Hello. Today is beautiful." We would tokenize the comment to become ["Hello", "Today" , "is", "beautiful"]. We first check if the string "hello" contains in the tokenized list. Since it is in there, a 1 is added to the list. Now we look at "world". Since "world" is not in the tokenized list, 0 will be added. This process keeps repeating until you have iterated through the bag of words for that sentence. Your feature will end up looking like this: [1,0,0,1]. This will be associated with the comment ["Hello", "Today" , "is", "beautiful"].

My modified model uses the same idea. As stated, my modified model is a bigram model. It will look at co-occurrence words, calculate the top 22500, and use that as its bag of words

(although it is not technically a bag of words but a bag of co-occurrence words). Once you have the bag of words, you can follow the similar procedure as you would in unigram. An example of this would be… let's say your bag of co-occurrence word is [("Hello", "world"), ("world" , "nice"), ("nice", "today")] and your comment is "Hello. Today is beautiful.". We will split that string into bigrams as well so the bigram will look like this: [("Hello", "Today"), ("Today", "is"), ("is", "beautiful")]. Since none of the words in bag of words are contained in this comment, the feature [0,0,0] is associated with it. Again, I have limited my bag of words size to be top 22500 of most frequent appeared bigram instead of everything in the vocab set like unigram model uses. An explanation to why I limited the size of the bag of words can be found in paragraph.

Things that I have tried in my modified mode that didn't improve the accuracy score:

I have tried several things in my modified model that didn't seem to work well. One thing that I stated was using frequency word count instead of 1's or 0's when creating the feature. That didn't seem to work. I then moved on to getting rid of bigrams completely and instead downloaded a sentimental lexicon. Each word in the lexicon has a value attached to it indicating if the word has a negative meaning or a positive meaning. I was able to create a dictionary using the words as keys and the associated value indicating whether its negative or positive as its value. Once I created that bag of words, for each word in the bag of words, if the comment contains the word that is in the bag of word, a number will be appended to the feature. A value "1" will indicate that the word has a positive meaning, a value "-1" will mean that the word has a negative meaning, and a value "0" if the word doesn't exist, if the word is neutral, or if the word is both positive and negative. That didn't seem to work and in fact lowered my average accuracy when doing cross validation by roughly 10%.

I have also tried to combine both bigram and sentimental lexicon. What I mean by this is that we will check for words in bigram, create the feature list. We will then append to that feature list by checking the words in sentimental lexicon. That didn't seem to work and I thought maybe it was because in my bigram, I am only using 1s or 0s and in my lexicon approach, I am using -1, 0, and 1. I decided to modify it so that instead of storing 1s or 0s, I will store the total number of appearance for bigram and the total number of appearance for lexicon (I.E., if a negative word

appear 5 times, then a -5 will be stored instead of -1). This gave me a slight improvement over the previous attempt but it was still not better than strictly using top 22500 bigrams.

Lastly, I have tried trigrams using the same feature building method as unigram and bigram with binary values. Trigrams performed better however the computational speed was really slow. As a result, I didn't bother with Trigrams and went with Bigrams instead since the difference wasn't too big.

Comparisons with values:

If you run my program the way it is right now, you would get details on the how both models (both Step 2 and Step 3 model) perform in cross validation, T-Test, Precision and Recall, Accuracy, Accuracy on test data, and Precision and Recall on test data. I have also provided how it would compare against the Majority Baseline Model (a model where you find the most frequent label and for every prediction, you would predict that label). Since the data are extremely skewed (there are a lot more label 1s than there are with labels 2, 3 , 4), the baseline model does extremely well and as a result, it is really hard to beat.

Comparing Results:

Models:	Majority Baseline	Vocabulary Feature (Unigram)	Bigram Feature
Score on Test Set:	0.7870370370370371	0.7453703703703703	0.7685185185185185
Score on Train Set:	0.7968561064087062	1.0	0.9975816203143894
Average score on Cross Validation:	0.7967381722009991	0.7375844842785776	0.7774169850132238
P-Value vs Majority	Not Tested	0.014447444542895463	0.019408503445230398
P-Value vs Vocab Feature (Unigram)	0.014447444542895463	Not Tested	0.04378039239290925

P-Value vs Bigram	0.019408503445230398	0.04378039239290925	Not Tested

Score on Test Set:

As you can see from the table above, it is hard to beat the majority baseline since the data are skewed. However, the modified model (bigram feature) comes slightly close to Majority Baseline. The vocab feature model performs really bad when compared to the majority model. This is because it is inaccurately predicting a lot of the 1's label into 2 and thus decreasing the performance. On the bigram model, you can see a slight improvement when compared to the vocab feature. This tells me that the model is predicting the correct label. However, in terms of which label it predicts correctly (either label 1 or label 2) will be hard to tell without precision and recall data. If you run my code, you will see a precision and recall data being printed out when doing k-splits. You can see that it is predicting labels to be 2 and it has been getting successes on doing so.

Score on Trained Set:

When performing on the train dataset, it is pretty obvious that both vocab model and bigram model will outperform the majority model. When comparing vocab feature to Bigram, I believe that the reason why bigram is not at a 1.0 is due to potentially overfitting when coming across similar feature set. For example, say you have a comment where the feature is [1,0,0] and it is labelled as 1. Now you have two more feature of [1,0,0] which is labelled as 2. Now, if you were to predict on feature [1,0,0], it will most likely predict that it is 2 although the correct result is 1.

Average Score on Cross Validation:

In terms of average score on cross validation where split = 10, the score is as expected. The explanation is similar to the explanation I gave on Score on Test Set.

T-Test and P-Value:

The purpose of T-Test is to show exactly how significantly different each model are. If the P-Value is lower than a certain threshold (typically 10%), you can reject the null hypothesis

which states that there are no difference between the two. As you can see, all of our P-Values are lower than 10% and thus we can reject the null hypothesis.

Posing a question:

This homework assignment has got me thinking of exactly what I could do to create a good model. There are a lot of techniques in terms of feature extraction that directly translates into the accuracy of your model. However, each technique produces a different accuracy rating and strictly aiming for the best accuracy rating may not be ideal (because if I were to adjust my number 22500 to 30000 or simply getting all bigrams, my model will be a lot more hesitant to predict a label other than the majority and there wouldn't be any difference between bigram model vs majority if your dataset is extremely skewed). You would have to take a look at other numbers that is embedded in accuracy value… such as precision and recall for each label.

My question would be… if given a much larger dataset, if we were to train a bunch of other models (say about 9 total models… odd number of models) each with having completely different feature extraction and then selecting the most frequent predicted value, how well will that perform? For example, let's say I have Unigram, Bigram, and another model that uses a completely different feature extraction to predict on a comment. Say Unigram and Bigram predicted label 1 while the other predicted label 2. Since two models say it is label 1 and one says it is label 2, I would predict that the label is 1.

This thought came to me when I was specifically looking at the recall and precision. Obviously majority has a precision and recall value for Label 1 at 100%. The precision and recall value for Label 2 is at 0%. Unigram is somewhat mixed in terms of precision and recall value for label 1 and label 2. Bigram tends to have a higher precision and recall value for label 1 and lower precision and recall value for label 2. This has gotten me thinking. What if we were to were to create models that has a high precision and recall value for label 2, another that has a high precision and recall value for label 1, and another having a mixed. Will our total accuracy improve now that we have one model that is more leaning towards predicting 1 and another more leaning towards predicting 2?

Another thought would be what would happen if we do something similar but instead of with different feature extractions, we would work on different sets of data? (Splitting the data set into N folds and we will have N logistic regression models each training on one of the folds).

This thought seems not as ideal as the first thought mainly because if we were to split the data set up and train N models, why don't we just train one model and come up with better features? Another error with this approach would be that since our datasets are small, cutting the dataset into N splits and having N models training on each of the splits might produce a logistic regression that has extremely low accuracy. Because the accuracies are low, even if you select majority vote, the accuracy might not improve at all (thinking of it as if I were to guess a label 20 times and selecting the most frequent appeared label, at the end of the day, it's still a guess). With that said, I will be performing my experiment on having different feature extraction (my first proposal).

Since I don't have much data for now, I can only use what is given to me. My experiment will also be limited as I will be using Majority Model, Vocab Feature Model, and my Modified Bigram Model as the three models that will be making a prediction. I will then find the most frequent predicted value and use that as the predicted value. I will then compare the result to the actual label. I will then calculate the accuracy by determining how many I predicted correctly over the total. My prediction for this experiment is that the overall accuracy will increase.

Results of experiment:

Accuracy with Majority Voting:	0.7870370370370371
Accuracy with Majority Model Alone:	0.7870370370370371
Accuracy with Vocab Feature Model Alone:	0.7453703703703703
Accuracy with Modified Bigram Model Alone:	0.7685185185185185

	Precision	Recall
Label 1	0.79	1.0
Label 2	0.0	0.0

As you can see, the majority voting and the majority model has the same exact accuracy when trained on the test data. This result may not be as accurate mainly because of the majority model. If there is another model that has similar accuracy as both vocab set model and bigram model, I strongly feel that the overall accuracy may improve. If you take a look at precision and recall for Label 2, it seems that there was never a case where the predicted label was a 2. In this

experiment, the only way for the majority vote to predict a 2 would be if vocab feature model and modified bigram model both predicts a 2 since majority voting always predict 1. However, it seems like with the given dataset, there was never a time where vocab feature predicted a 2 while modified bigram model also predicted a 2. Since Majority always predict 1, the experiment might not be as accurate. It would be much better if we were given a larger dataset as well as having a better model to replace majority model or simply creating 2 additional models that can be added to the list of models (since you want an odd number of models so that there are no ties).