Michael Ing October 29, 2020

Hwk 3 – Classification and Vector Semantics

**Baseline Classifier**

For Step 2 of the assignment I used the xlrd package to read the column data from the excel sheet, and then cleaned the data to get rid of punctuation, numbers, and excess symbols. I then split the data into training and test sets and used the scikit learn count-vectorizer to get word-count vectors, both base and with tfidf from the training set. I passed these vectors with their corresponding y-values to the logistic classifier model from scikit learn, which trained it. Finally, I used this fit logistic classification model to predict on the test data, and counted the number of correct predictions to determine accuracy. I repeated this process with different randomly generated test and training sets, using a test set that was a third the size of the total data each time.  
  
The majority-vote classifier was trivial to implement. I simply trained the data by keeping track of basic word-count vectors in the vocabulary of the training set, and determined if a word was found more in the “constructive” class or the “not constructive” class. In each comment on the test set, I then kept a running sum, adding 1 if the word was more “constructive” and subtracting 1 if the word was more “not constructive”. I then classified the sum based on whether it was positive or negative, and counted the number of correct responses to get the accuracy.

My base model ran with 3 cross-validation sets (and thus a test size of 1/3) and a window size of 5 for sparse vectors. My baseline data showed an accuracy of 53.16% for the majority classifier (slightly better than guessing, as expected), and 85.06% accuracy for the bag-of-words, which was surprisingly high. The tfidf for bag-of-words got an accuracy of 81.23% which is lower than the normal vectors. These reasonably high levels of accuracy show that using logistic regression is in fact a large improvement on a baseline majority classifier. The dip in tfidf accuracy could be due to the small amount of data, and overfitting caused by a large vocabulary.

**Two “Improvements”**

In Step 3 I introduced two more models, using sparse vectors and dense vectors. The sparse vectors were calculated for each comment by iterating over the training data and building context vectors for each word based around a certain window size (base of 5). Once these context vectors were built for the entire vocabulary, I combined and averaged the context vectors for all words in a single comment to train the logistic regression model. The test data was then similarly divided into context vectors, which were then summed and averaged to get test features on which the model could predict. The dense vector model was used in a similar way, except that instead of generating and using word-context vectors, I used pre-trained word embedding vectors that were more dense (size 300 compared to vocabulary size of >6000) from GoogleNews-vectors-negative300, a source trained on a corpus with size over 1 billion. I similarly applied a tfidf model to those vectors and found that sparse vector model had probabilities of 53.35% and 53.40% for the regular and tfidf vectors, respectively. The dense vector model on the other hand showed probabilities of 59.19% and 80.27% for the regular and tfidf vectors, respectively.   
  
It makes sense that the dense vector model performed well, as the large dataset on which it was trained allows for relatively accurate word embeddings. It also makes sense that there is the largest jump in accuracy from normal to tfidf models here, since these vectors are trained on the largest number of documents and could be most affected by presence in samples. The sparse vector representations on the other hand perform surprisingly poorly, on par if not worse than the majority-vote classifier which is essentially as good as flipping a coin. As a result, I will be exploring ways to improve the accuracy of this model. What is most surprising is that neither of these models perform as well as the simple bag of words model. This could be due to the small vocabulary size which could lead to overfitting, or to the small number of samples which doesn’t allow for a good set of summary vectors to be created, even with cross-validation.

**Potential Gains**

Based on my results, I wanted to explore why the sparse vector representation performed so poorly, as well as different ways to potentially improve the accuracy by altering parameters. The data is summarized as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Accuracy** | **Sp\_Vect reg** | **Sp\_Vec tfidf** | **Vocab Size** |
| **CV=3, W=5** | 53.35% | 53.40% | 6245 |
| **CV=3, W=10** | 54.12% | 52.78% | 6266 |
| **CV=3, W=2** | 53.07% | 52.20% | 6150 |
| **CV=3, W=100** | 53.07% | 52.20% | 6309 |
| **CV=5, W=5** | 46.27% | 54.50% | 7001 |
| **CV=2, W=5** | 45.69% | 52.01% | 5310 |
| **stopwords** | 57.33% | 53.64% | 6126 |
| **stemming** | 47.75% | 53.54% | 4483 |
| **both** | 57.57% | 53.35% | 4413 |

The first thing I tried was altering the cross-validation sets from a base of 3 to 2 and 5, which changed the size of the training data from 66% to 50% and 80%, respectively. As can be seen both had a negative effect on the regular accuracy, dipping below 50%. I then tried changing the window size from a base of 5 words on each side to 2, 10, and even 100. Over a couple of iterations this also proved to do very little for the accuracy, changing only slightly in either direction which could easily be caused by random variation.  
  
I then turned from parameter tuning to input text cleaning, as I had included a basic cleaning method that got rid of all non-letters and moved all letters to lower case. I found two pre-made packages, “stopwords” and “PorterStemmer”, both from nltk that could further clean the training text and potentially give some improvements. The first, stopwords, introduced some basic simple words that were excluded from the input to avoid oversaturation, including words like “a”, “the”, and “I”. As can be seen, this provided a small improvement over the baseline, from 53% accuracy to 57%. The other package, PorterStemmer, reduced words to their stems which reduced the vocabulary size and thus the dimensions of the context-vectors. I thought this would improve the results by reducing dimensionality, but in fact this had the opposite effect, causing a decrease in accuracy for the regular vector to push it below 50%. The final thing I tried was combining both tactics, which showed similarly small improvement to just using stopwords, but also decreased the vocabulary size. I included the vocabulary size column to show that it changed significantly over my experiments, but with little correlation to the accuracy. It seems as though the accuracy of the sparse-vector model remained relatively unchanged and poor, and although I believed reducing the vocabulary size would help by making the vectors more dense, there was in fact little to no impact. It could be that these changes weren’t significant enough, and that the context-vector size was just too big to be useful.  
  
It is also worth noting that the tfidf accuracy remained almost entirely uniform in spite of all of my changes and experiments, ranging from 52.2% to 54.5%. For this data, this vector regularization technique seems to be useless, likely due to the fact that the vocab size is too large for a very small number of samples and words, so the tfidf measure simply further smoothed the vectors into uniformity.

Overall, the accuracy of the sparse vector model did not change very much despite rather drastic changes in vocab size, initial parameters, and text cleaning methods. From this observation I can conclude that this dataset has too few samples and too large of a vocabulary for sparse vectors to be a viable classification technique. In order to see real accuracy gains in either category, I think many more samples would need to introduced, so that the number of samples drastically out scales the size of the vector, similar to how the dense vector Google-Net embeddings were trained on billions of samples with a vector size of only 300. Maybe then the changing of cleaning techniques like word-stemming and removing stopwords would have a more pronounced impact on overall accuracy.