NLP Coursework 2

Samuel Zureick x83005sz

Part 1: Clulstering

Method Description

My text cleaning and pre-processing steps for this application were much more involved than previous projects due to the presence of many noisy tags indicating different parts of the reviews. My first steps were to remove all these tags with regular expressions, change it to lowercase, and split on sentence boundaries. I then remove all non-alpha-numeric characters and regularize whitespace to a maximum length of one. I then replace each sentence with a copy of itself with stop words removed, and then use NLTK's word tokenize to tokenize the sentence. Following this, I use the porter stemmer to get the stem of each word in the sentence. I then remove any remaining sentences that are empty after pre-processing.

To construct my distributional semantic representation, I use gensim's Word2Vec, with a window of 3 and vector size of 72, alongside relatively high alpha values and training iterations, being .15 and 32, respectively. This generates a feature vector for each word in my vocabulary based on a continuous-bag-of-words model.

Result Analysis

After 10 iterations of clustering, I get an average correct classification of 78.4% with a standard deviation of 4.71. I believe that with a large review set or a set with more detailed reviews, I could have gotten much higher clustering results, and I think that one of the limitations of this model is the appearance of multiple clusters of size > 2. There exist 50 clusters and there are no empty clusters, but my model is not amazing at distinguishing between certain key tokens.

For my word vectorization with Word2Vec, I settled on a window size of 3 and a vector size of 72. With smaller vector sizes, the model was not making large enough distinctions between similar words- words that were particularly tricky were router, ipod, and phone, which all were often placed in the same cluster and were ranked most similar to each other, instead of their inverse. I chose a smaller window size of 3 because for this application the most immediate context had the most impact on the feature vector. Chose a high alpha value of .15 for training for this reason as well- my vectors were not distinct enough, and with default hyperparameters I was getting many clusters with over 6 items in it, which greatly degrades accuracy because it leaves many clusters with only one item.

I used KMeans from SKlearn for my clustering algorithm. While this algorithm is not naturally deterministic, I chose to use this for its simplicity. By default it uses Lloyd as the K-Means algorithm, and after a grid search for the best performing algorithm, Lloyd won.

Part 2: Neural Network Classification

Method Description

Preprocessing included removing unnecessary tags such as [t], removing newlines, and lemmatization with the NLTK's WordNetLemmatizer. In order to get an accurate class label, I averaged the sentiment provided by the text tags, as some reviews included multiple sentiment tags. I did not preserve any document IDs and instead just stored the reviews as one single list, as the document ID did not have any impact on the sentiment of the review.

To transform my reviews to sentence embeddings, I use Word2Vec's skipgram model trained on the vocabulary in the reviews to generate a vector representation of each word and average this vector to use as the representation of each word in the feature vector. I use the skipgram model for my word embeddings

My model is composed on the inputs as the word embeddings, followed by a 64-unit LSTM with dropout of .3, then followed by a single unit dense output layer. Because the task is binary classification, the output layer used a sigmoid activation function, and the model loss is calculated with binary cross entropy. I use a 10% validation set when fitting the model, as anything over this was reducing my training set too much.

Experiment and Result Analysis

Overall, after 5-fold cross validation I tend to get results of about 68% classification accuracy with a standard deviation of around 12% with a BCE loss of around .67. I believe that one of the reasons that my classification accuracy was so low was that after filtering out only reviews that contain a sentiment, and then splitting these sets into train, test, and validate, I was not left with many feature vectors. Additionally, the data was quite noisy, which was something I attempted to correct with the LSTM dropout. While the dropout helped reduce the loss, it did not perform amazingly. I got better results by reducing the dimensionality of the feature vectors by eliminating extremely long reviews, but this also hurt my performance by reducing the number of features. I use 64 LSTM units because my model was underfitting even more with lower numbers of units, but I did not see any noticeable improvement after 64 units, which makes me believe that my feature vectors are not descriptive or clean enough. In order to prevent the occurrence of bias, I generated class weights using SKlearn's class_weight with the balanced hyperparameter, and this appears to have improved performance slightly.