CSC 791 Natural Language Processing

P1: Word Vectors

Poorvi Rai (prai) – 200263486

For this assignment, we try to classify event phrases into 3 classes using word embeddings. This is done by calculating the vector of each word in a phrase and then taking their average to obtain a single vector for each phrase. The vectors of the two phrases are then concatenated to get a single vector for each pair. We then train a classifier to classify the vectors into 3 classes based on the ordering of the phrases.

The basic idea behind word vectors is that semantic vectors should preserve most of the relevant information about a text while having relatively low dimensionality which allows better machine learning treatment than straight one-hot encoding of words. The given dataset is a small one and one of the advantages of a topic model is that they are unsupervised so they can help when labeled data is scarce. A high-quality topic model can be trained on the full set of one million. Through transfer learning, we can leverage existing language models that have been trained on much larger corpora for our own use.

Preprocessing Data:

Data processing is one of the vital and must to do step in exploratory data analysis in any data science project. If the project is related to raw text data, the cleaning and processing are musts.

First, I import all the necessary libraries. I used Scikit-learn and Keras libraries to make the models. ‘nltk’ library was used to tokenize the sentence and remove stop-words. The ‘re’ library was used for regex.

I read the necessary data from the csv files as training and test dataframes. Then the data must be cleaned. Regex becomes the vital part of this step. Regex can find a pattern in the raw, messy text and perform actions accordingly. The function removes all the punctuations, then converts all the words in lower case. I used the ‘nltk’ stop-word list to remove them from the text. Later, the function performs some regex operations to clean the unnecessary part of the text. Finally, I used ‘SnowballStemmer’ to stem the words. Stemming is also another important part of NLP.

This data preprocessing step is common to all following models.

Baseline Models:

1. GloVe (Global Vectors for word representation):

The first word embedding technique I used for classification is GloVe. It is an unsupervised learning algorithm that generates word embeddings by aggregating global word-word co-occurrence matrix from a corpus.

Here, I use word embeddings from pre-trained GloVe. It was trained on a dataset (‘glove.6B.100d.txt’) containing one billion tokens (words) with a vocabulary of 400 thousand words. The glove has embedding vector sizes: 50, 100, 200 and 300 dimensions. I chose the 100-dimensional one. I intentionally keep the “trainable” parameter as ‘False’ to see if the model improves while keeping the word embeddings fixed.

After cleaning and preprocessing the data, I tokenize the event phrases and begin creating sequences. Tokenization of sentences is one of the essential parts in natural language processing. Tokenization simply divides a sentence into a list of words. I used Keras tokenizer function to tokenize the strings and then used another important function ‘texts_to_sequences’ to make sequences of words. Because of the computational expenses, I used only the top 20000 unique words from the dataset ‘glove.6B.100d.txt’.

After this I create a weight matrix and develop a classifier model. I used the Keras library to build a neural network classifier. The network starts with an embedding layer. The layer lets the system expand each token to a more massive vector, allowing the network to represent a word in a meaningful way. The layer takes 20000 as the first argument, which is the size of our vocabulary, and 100 as the second input parameter, which is the dimension of the embedding. The third parameter is the input_length of 200, which is the length of the concatenated vector of two event phrases.

Result: Although the training accuracy steadily increases over 100 epochs to almost 82%, the accuracy with the test set is only 41%.

2. SpaCy word2vec:

Spacy is a free, open source python framework to support Natural Language Processing tasks. Spacy comes with a pre-trained word2vec model with a vocabulary of more than a million words.

Using pre-trained models in Spacy is incredible convenient, given that they come built in. Simply download the core English model of your choosing. Spacy has a number of different models of different sizes available for use, with models in 7 different languages (include English, Polish, German, Spanish, Portuguese, French, Italian, and Dutch), and of different sizes to suit your requirements.

I have used the larger-than-standard en_core_web_sm library, which includes 20000 unique vectors with 300 dimensions. The vectors in Spacy were used by first loading the model, and then processing text. The vectors can be accessed directly using the .vector attribute of each processed token (word). The mean vector for the entire sentence is also calculated simply using .vector, providing a very convenient input for our machine learning model based on event phrases.

I used the MLPClassifier from the scikit-learn library for classification of the sentences. MLPClassifier trains iteratively since at each time step the parameters are updated. The implementation works well with dense numpy arrays of floating-point values. The optimizer specified was ‘adam’ and the model was run over 100 max iterations. Everything else was kept similar to the previous model.

Result: This model gave a testing accuracy of nearly 50%, which is marginally better than the previous model.

MY Approach:

Average word vectors do not consider the sequential order of the tokens ("I bought an apple and then ate it" and "I ate an apple and then bought it" have the exact same vectors). I therefore used the Universal Sentence Encoder.

The Universal Sentence Encoder encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering and other natural language tasks. The model is trained and optimized for greater-than-word length text, such as sentences, phrases or short paragraphs. It is trained on a variety of data sources and a variety of tasks with the aim of dynamically accommodating a wide variety of natural language understanding tasks. The input is variable length English text and the output is a 512-dimensional vector.

After preprocessing the data, I load the Universal Sentence Encoder and compute the embeddings for the ordered event phrases. First time loading the module takes a while since it will download the weights files. The value of the embedding for each phrase is an array of 512 floating point numbers which forms the vectors for the sentences.

I then used the Convolutional Neural Network (CNN) for classification. Using CNN, the result of each convolution will fire when a special pattern is detected. By varying the size of the kernels and concatenating their outputs, we can detect patterns of multiples sizes (2, 3, or 5 adjacent words). Patterns could be expressions and therefore CNNs can identify them in the sentence regardless of their position. I used the Keras library to build the CNN. Keras supports two main types of models namely, the Sequential model API which I have used and the functional API which can do everything of the Sequential model but it can be also used for advanced models with complex network architectures.

The Sequential model is a linear stack of layers, where you can use the large variety of available layers in Keras. The model starts with an embedding layer. The layer lets the system expand each token to a more massive vector, allowing the network to represent a word in a meaningful way. The layer takes 20000 as the first argument, which is the size of our vocabulary, and 512 as the second input parameter, which is the output dimension. The third parameter is the input_length of 512, which is the length of the vectors. The next layers are convolution and max pooling. I have used 64 filters of size 5 in the convolution layer and pool size of 4. The next is the Dense layer which is your regular densely connected neural network layer with an activation function of softmax.

Result: Unlike previous models, the training accuracy remains almost constant throughout the 100 epochs. The accuracy with the test set is 44%. It is only just better than the GloVe model but not as much as the Spacy model.

Comparisons:

The GloVe word embeddings vector turns a word to 50-dimensional vector. The Universal Sentence Encoder is much more powerful, and it is able to embed not only words but phrases and sentences. That is, it takes variable length English text as input and outputs a 512-dimensional vector. Handling variable length text input sounds great, but the catch is as a sentence gets longer counted by words, the more diluted embedding results could be. And since the model was trained at the word level, it will likely find typos and difficult words challenging to process. This why the model performs better than GloVe but not as good as Spacy.

Spacy uses the latest and best algorithms, its performance is usually good as compared to NLTK. The other models, especially GloVe, attempt to split the text into sentences. In contrast, Spacy constructs a syntactic tree for each sentence, a more robust method that yields much more information about the text.