Word Embeddings for Text Classification

An Overview of Word2Vec and Discussion of Tradeoffs

Tech Review - CS 410 - Fall 2020 - Amrutha Gujjar (Paper Source)

Intro:

Humans are much better equipped to gather meaning from natural language than computers are. Word embeddings transform natural language into a numerical form that is understandable by a machine. The challenge is to make this transformation meaningful enough to capture the syntactic structure and intent behind natural language. A good word embedding translates into better model performance.

The most naive way to compute a word embedding is to map each word in a document to a distinct number - independent of semantic similarity. This approach is sometimes called the 'one-hot' encoding vector. This word to integer mapping approach will be on the order of n, where n is the vocabulary size.

Word2Vec is an opportunity to (1) Capture contextual closeness of words that is lost in a naive embedding, and (2) reap some value from dimensionality reduction, thereby alleviating the computational cost associated with the model. Rather than mapping each word to a distinct index, if we cluster syntactically similar words together, we can capture semantic similarity between various phrases in a more abstract way. Word2Vec's true value lies in its ability to demonstrate the context of a word within a document.

In a Nutshell

How does the actual clustering of syntactically similar words work? Basically, the expectation is that if two words are syntactically similar, then the outputs are also similar. Therefore, the word2vec algorithm learns the similarity. The word2vec approach uses shallow neural networks to compute this. The underlying neural network can transform an entire corpus into a high

dimension vector space. Words with similar contexts are assigned vectors that are closer to one another. The model can be trained with negative sampling, the samples of which come from a unigram distribution model. This approach precludes the need to sample the entire vocabulary. Only the words that have a label of 1 in the one-hot word vector of the vocabulary have their weights updated.

Additionally, there are multiple possible similarity functions that can be used to compute the distance between vectors in the vector space for words that are similar, just as we saw for other models such as VSMs (Vector Space Model). Any kind of one-hot representation treats all words as fully independent of one another - However, cosine similarity, on the other hand, introduces dependence by correctly capturing that the existence of one word increases the probability of another word existing.

Computational Power

For any of these types of word embeddings tasks, it is worth thinking about the respective computational cost of different algorithms. This becomes important when working with massive data sets. Decisions like 'should I use n-gram vectors or a bag of words representation' can have massive implications. For example, the bag of words approach for word2vec yields is less computationally expensive, and faster. This is because the BOW representation effectively loses a lot of information about the surrounding context of a given word. However, the N-gram approach provides higher accuracy. There is no hard and fast rule on making a decision one way or another in a generalizable way, and often empirical methods may need to be used to determine the best way forward for a given use-case.

Word2Vec in the real world:

Translating the academically sound approach of word2vec into something that is usable and valuable in practice is always a challenge. Tuning the hyperparameters of the word2vec model is crucial to ensuring that the best results are achieved for a given use case. Tradeoffs are often made regarding accuracy or computational complexity, but for extremely large datasets, this is often necessary. Additionally, using/curating a large training set, and having a broad variety of words within the corpus / vocabulary can yield better results, but again, these additions add to the computational complexity that comes with a higher dimensional vector space.

Conclusion:

Ultimately, word embeddings are a powerful tool within text classification. By choosing the right number of dimensions and using a well-vetted similarity function to group words together, we can both improve the computational cost of text classification, while also making it generalizable enough to ascertain the meanings of semantically similar documents, phrases and sentiments.