Natural Language Processing

MInDS @ Mines

Natural Language Processing involves the various applications that intersect human languages and computer science. Machine learning applied to natural language applications has been shown to produce useful products due to its impressive results. In this lecture we focus on the numeric representation of natural language and common features used when applying machine learning to natural language.

Natural Language Processing (NLP) is the intersection of linguistics and computer science among many other fields. Our focus with respect to NLP is to develop an accurate numeric representation of text to apply machine learning techniques in language-based applications. Applications to language include sentiment analysis, entity recognition, and topic modeling. For all these applications, we need a good feature representation of text and that will be the topic of discussion here.

Background & Definitions

First, let's examine some details and definitions that are shared with / borrowed from linguistics. We refer to a sample of text data as a *document*. We refer to a group of documents as a *corpus*. A *vocabulary* is the set of words that belong to a language. A *grammar* defines all the possible (syntactically correct) formations of sentences in a language. For the purposes of NLP, we focus on context free grammars. A *parser* follows a grammar to determine the possible (syntactically correct) parse-trees for a particular sentence. When defining a grammar, we utilize *parts of speech* as the possible symbols. Words in the language's vocabulary are also assigned possible parts of speech. Example parts of speech are noun, verb, and adjective. A grammar focuses on possible structures of a sentence but not on whether they have any real meaning.

Context free grammars are grammars defined by $A \rightarrow \alpha$, where α is a set of symbols. Further details about grammars are beyond the scope of this course.

Semantics focus on the meaning of the text and pragmatics focus on including context in determining the meaning of text.

So far we've looked at the high level perspective on text. Next, we'll look at individual words in a sentence. A word is a variation of a *root* which is the portion of a word that cannot be reduced any further. Words are formed by adding a *prefix*, characters before the root, or *suffix*, characters after the root. Prefixes and suffixes create an *inflectional* morph when they maintain the same core meaning and part of speech and a *derivational* morph when they change the meaning and/or part of speech.

Given a sentence, our goal is to reduce it to its root form. This allows us to more easily determine the meaning of text and collapse multiple representations of the same intention into one representation. *Tokenization* is the process of splitting up a sentence into its pieces. We refer to each piece as a *token*. You can think of a token as a word with a root meaning. Following tokenization, we attempt to determine the base form of each token.

There are two main approaches to determine a word's base form; stemming and lemmatization. Stemming is a heuristic based approach that removes commonly found prefixes and suffixes to reduce a word to its lowest level word stem. Lemmatization uses the language's vocabulary to determine the lemma of the word which is the base form of a word used in the dictionary. Stemming used to be very common due to the computational complexity and lack of data for lemmatization but these days we have very powerful and efficient lemmatizers. A sequence of n tokens is called an n-gram and we refer to special cases where $n = 1, 2, 3$ as uni-grams, bi-grams, and tri-grams. n-grams are useful when we use multiple words to refer to an entity, ex. "machine learning" is a meaningful bi-gram and "natural language processing" is a meaningful tri-gram. An n-gram is simply a sequence of words and does not have to be meaningful. Our goal is usually to determine if they are meaningful or have a different meaning compared to unigrams. After applying tokenization, common practice is to remove words that add little value to the text called *stop words*. We can determine stop words by having a list of words for a language that are commonly used and considered filler words with respect to the task at hand such as "the", "a", and "an", or by determining words in a corpus that have a comparatively high frequency in all documents.

The Porter Stemmer and WordNet Stemmer are common stemmers used in NLP.

Natural Language Features

After applying tokenization, stop word removal, and lemmatization, we can now begin to extract features from our text. We will cover two simple approaches that do not utilize any machine learning techniques, and more complex machine learning based approaches for feature extraction of text.

Simple Approaches

The simplest feature that is commonly used in NLP and is often considered the baseline feature for text is the *bag of words*(BoW) representation. A bag of words representation of a document can be a $1 * m$ vector where m is the number of unique terms in the **corpus** and the i^{th} element represents the number of occurrences[1] of the i^{th} term in that document. BoW representations do not store any information about the order of the text, they only care about the number of occurrences. This approach is considered careless by linguists, since order is a significant part of language, yet it manages to produce good results in machine learning applications.

We can represent our entire dataset of documents as an $n \times m$ matrix, **X** often referred to as a *document term matrix*. The document term matrix naming convention is independent of the feature actually stored in the matrix but is used to denote the structure of the matrix of n documents and m terms.

An extension of BoW is to normalize the frequency of the words that

[1] A variation of BoW exists where instead of storing the number of occurences we only store a boolean value denoting whether that word occurred at all or whether it occurred more than k times where k is a hyperparameter.

appear based on the document length. If we simply use BoW, longer documents will have larger values overall and therefore produce a higher weight to infrequently occurring words. A simple approach to weighting the term count is to normalize by dividing by the total number of words in each document. Another approach is to use logarithmic scaling. If we denote the raw count of a term t in document d as $c_{t,d}$ we can represent the various term frequencies of term t in document$d$, $tf_{t,d}$ as,

$$\underset{BoW}{tf_{t,d}} = c_{t,d}, \tag{1}$$

$$\underset{Normalized}{tf_{t,d}} = \frac{c_{t,d}}{\sum_{i \in \mathcal{V}} c_{i,d}}, \tag{2}$$

$$\underset{LogScaled}{tf_{t,d}} = \log\left(1 + c_{t,d}\right). \tag{3}$$

In practice when we refer to term frequency we usually refer to the raw count or BoW. This may be confusing to get used to but is the current standard practice.

Term frequency as a representation can be effective but it results in all terms being weighted equally. This is not the case in practice, as words that appear infrequently across the corpus might possess superior discriminant ability and words that appear frequently may possess little value. We discussed words that are too frequently used being removed as stop words so we should incorporate this frequency into our feature representation. Document frequency of a term t which we represent as df_t is the number of documents in which a term t appears. Instead of using the term frequency for feature representation, we can inversely weight the term frequency by the document frequency such that words that appear frequently get penalized and words that appear infrequently get relatively boosted. This representation is called term frequency inverse document frequency (TF-IDF). The inverse document frequency of term t in a corpus can be calculated as,

$$idf_t = \log \frac{n}{df_t} \approx log \frac{n}{df_t + 1}. \tag{4}$$

We add one to the denominator to ensure that we do not divide by 0. We then calculate the term frequency inverse document frequency as,

$$tfidf_{t,d} = tf_{t,d} \times idf_t. \tag{5}$$

This approach is generally good if we assume that we already know all possible words in a language. If we learn a representation based on some training data and our test data contains new terms there are several ways we can work with this. One way is to ignore terms that were not in the training data. This is why spammers were able to get through spamming filters by writing the same words but using different characters in some words for example ”1” instead of ”i” and ”0” instead of ”o”. Another approach is to use the hashing trick. Instead of storing our data as a vector of length equal to the number of terms in the vocabulary, we use a hashing function that converts a term into a number that we use as the index for that word. This allows us to keep track of words that are not in the original training data since the hashing function forces the word to an index within the range limit. A classifier trained on the original data might still have issues with these words but at least our feature mapping is incorporating them.

Learned Embeddings

Another more complex approach that utilizes machine learning is to learn an embedding for every word that is somewhat representative of the relative meaning or relationship between it and other words. By learning an embedding, we force the dimensionality of the text to be consistent. We can pre-train the model on a much larger corpus that is likely to have any words that people use and then we can use that model on our particular dataset to determine the feature representation.

Despite the popularity of both word2vec and GloVe, there are newer superior word embeddings and many are targetted to specific applications. An example of that is sentence embeddings as opposed to word embeddings.

There are two commonly used approaches for learning embeddings for text; word2vec[2] and GloVe[3]. word2vec uses a deep learning model to learn the word embedding for text. GloVe uses matrix factorization techniques with word-word occurrence tracking to learn the word embeddings. Both these approaches are often trained on a large dataset and we then we can simply use the vectors from the pre-trained model via a lookup for each word in our corpus.

One benefit of these word embeddings is that they learn common relationships between words which lead to linguistic arithmetic such as,

$$\text{King} - \text{Man} + \text{Woman} \approx \text{Queen}. \tag{6}$$

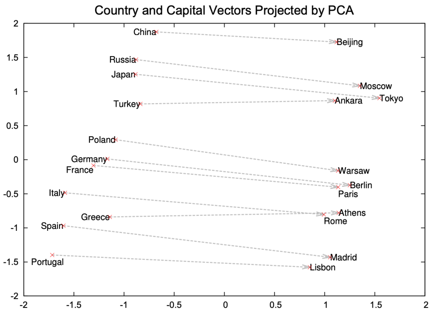

Figure 1: Example learned relationships between countries and their capital cities in their word2vec embeddings.

A good demo of the learned relationships from word embeddings is available at https://lamyiowce.github.io/word2viz/.

Finally, another method used for feature representation of text is latent Dirichlet allocation (LDA). LDA is a topic modelling method that assumes that every document belongs to multiple topics and learns those topics based on the distribution of words across the corpus. Based on that, the feature representation of the text is the number of unsupervised topics that exist in the corpus and the values for each document are binary based on whether it belongs to each topic. LDA will not be covered in detail due to time limitations but you should be aware of its application to topic modeling.

[2]

[3]