Natural language processing

Natural language processing is a branch of machine learning that deals with how machines understand human languages. Text data is a widely available problem domain for NLP tasks.

In order to work with text data, it is important to transform the raw text into a form that can be understood and used by machine learning algorithms, this is called text pre-processing. We have various techniques for text pre-processing like stemming, lemmatization, POS tagging, and dependency parsing.



**Part-of-speech (POS) tagging:**

Part-of-speech tagging is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)). The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.

Part-of-speech tags describe the characteristic structure of lexical terms within a sentence or text, therefore, we can use them for making assumptions about semantics. Other applications of POS tagging include:

* Named Entity Recognition
* Co-reference Resolution
* Speech Recognition

When we perform POS tagging, it’s often the case that our tagger will encounter words that were not within the vocabulary that was used. Consequently, augmenting your dataset to include unknown word tokens will aid the tagger in selecting appropriate tags for those words.

**Default tagging**is a basic step for the part-of-speech tagging. It is performed using the DefaultTagger class. The DefaultTagger class takes ‘tag’ as a single argument. **NN** is the tag for a singular noun. DefaultTagger is most useful when it gets to work with most common part-of-speech tag. that’s why a noun tag is recommended.

bag-of-words

The bag-of-words model is simple to understand and implement and has seen great success in problems such as language modeling and document classification.

Machine learning algorithms cannot work with raw text directly; the text must be converted into numbers. Specifically, vectors of numbers.

A bag of words is a representation of text that describes the occurrence of words within a document. We just keep track of word counts and disregard the grammatical details and the word order. It is called a “bag” of words because any information about the order or structure of words in the document is discarded. The model is only concerned with whether known words occur in the document, not where in the document.

**Use of Bag-of-Words algorithm**

One of the biggest problems with text is that it is messy and unstructured, and [machine learning](https://www.mygreatlearning.com/blog/what-is-machine-learning/?highlight=what%20is%20machine%20learning) algorithms prefer structured, well defined fixed-length inputs and by using the Bag-of-Words technique we can convert variable-length texts into a fixed-length **vector.**

Also, at a much granular level, the machine learning models work with numerical data rather than textual data. So to be more specific, by using the bag-of-words (BoW) technique, we convert a text into its equivalent vector of numbers.

**Managing Vocabulary**

For a very large corpus, such as thousands of books, that the length of the vector might be thousands or millions of positions. Further, each document may contain very few of the known words in the vocabulary. This results in a vector with lots of zero scores, called a sparse vector or sparse representation.

Sparse vectors require more memory and computational resources when modelling and the vast number of positions or dimensions can make the modelling process very challenging for traditional algorithms.

As such, there is pressure to decrease the size of the vocabulary when using a bag-of-words model.

There are simple text cleaning techniques that can be used as a first step, such as:

* Ignoring case
* Ignoring punctuation
* Ignoring frequent words that don’t contain much information, called stop words, like “a,” “of,” etc.
* Fixing misspelled words.
* Reducing words to their stem (e.g. “play” from “playing”) using stemming algorithms.

A more sophisticated approach is to create a vocabulary of grouped words. This both changes the scope of the vocabulary and allows the bag-of-words to capture a little bit more meaning from the document.

In this approach, each word or token is called a “gram”. Creating a vocabulary of two-word pairs is, in turn, called a bigram model. Again, only the bigrams that appear in the corpus are modeled, not all possible bigrams.

**Scoring Words**

Once a vocabulary has been chosen, the occurrence of words in example documents needs to be scored.

In the worked example, we have already seen one very simple approach to scoring: a binary scoring of the presence or absence of words.

Some additional simple scoring methods include:

* **Counts**: Count the number of times each word appears in a document.
* **Frequencies**: Calculate the frequency that each word appears in a document out of all the words in the document.

### **Word Hashing**

You may remember from computer science that a [hash function](https://en.wikipedia.org/wiki/Hash_function) is a bit of math that maps data to a fixed size set of numbers. For example, we use them in hash tables when programming where perhaps names are converted to numbers for fast lookup.

We can use a hash representation of known words in our vocabulary. This addresses the problem of having a very large vocabulary for a large text corpus because we can choose the size of the hash space, which is in turn the size of the vector representation of the document.

Words are hashed deterministically to the same integer index in the target hash space. A binary score or count can then be used to score the word.

This is called the *“*hash trick*” or “*feature hashing*”.*

The challenge is to choose a hash space to accommodate the chosen vocabulary size to minimize the probability of collisions and trade-off sparsity.

### TF-IDF

A problem with scoring word frequency is that highly frequent words start to dominate in the document (e.g. larger score), but may not contain as much “informational content” to the model as rarer but perhaps domain specific words.

One approach is to rescale the frequency of words by how often they appear in all documents, so that the scores for frequent words like “the” that are also frequent across all documents are penalized.

This approach to scoring is called Term Frequency – Inverse Document Frequency, or TF-IDF for short, where:

* **Term Frequency**: is a scoring of the frequency of the word in the current document.
* **Inverse Document Frequency**: is a scoring of how rare the word is across documents.

The scores are a weighting where not all words are equally as important or interesting.

The scores have the effect of highlighting words that are distinct (contain useful information) in a given document.

### **Drawbacks of using a Bag-of-Words (BoW) Model**

1. If the new sentences contain new words, then our vocabulary size would increase and thereby, the length of the vectors would increase too.
2. Additionally, the vectors would also contain many 0s, thereby resulting in a sparse matrix (which is what we would like to avoid)
3. We are retaining no information on the grammar of the sentences nor
4. on the ordering of the words in the text.

Some of the most significant recent advances in machine learning and artificial intelligence have come in natural language processing (NLP). Most of the advanced neural architectures in NLP use word embeddings. A word embedding is a representation of a word as a vector of numeric values. For example, the word "night" might be represented as (-0.076, 0.031, -0.024, 0.022, 0.035).

**Need of Word Embeddings**

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation.  
Neural networks accept only numeric input, if you're working with NLP, words must be converted into numeric values of some sort. Theoretically you could represent words as single integer values, for example, "hello" = 1, "world" = 2, "cat" = 3, and so on, but there are two problems with this approach.

First, because of the way neural networks compute their output, words that are numerically close are indirectly processed as being semantically close. In the example above, "world" and "cat" would be processed as if they were very similar in some way.

The second problem is that even if you could somehow represent words that are close together in meaning with numeric values that are close, you could only represent similarity in one dimension. For example, the words "man" and "boy" can be compared on the dimension of gender (the same), or on age (a bit different), or on likelihood of being a software engineer (much different).