Note for Reinforcement Learning 2nd Edition

August 7, 2018

Text Classification

Text can be treated as a sequence of characters, words, phrases, named entities, sentences or paragraphs etc.

Tokenization is a process that splits input into useful unit(token) for the task at hand. (Can be word, sentence, paragraph). Example tokenizers are WhiteSpaceTokenizer, WordPunctTokenizer, TreebankWordTokenizer.

Token normalization operates on individual token. There are two main types of normalizations: stemming and lemmatization. **Stemming** uses simple rules and heuristics to remove/replace suffixes (e.g. Porter's Stemmer). **Lemmatization** use more advnace tehcnique such as vocabulary/morphological analysis (e.g. WordNet lemmatizer). Further normalization includes normalizing capital letters and acronyms.

Classical approach

Bag of Words(BOW) is a feature representation of text. For a set of text samples $\{s_1, \ldots, s_n\}$, we can extract a set of distinct tokens ("today", "a", "nice") or token pairs/triplets ("nice weather") $\{t_1, \ldots, t_m\}$. We define the bag of word representation to be an $n \times m$ frequency matrix B where

$$B_{ij} = \text{ \# of times } t_j \text{ appears in } s_i$$

n-grams consecutive tokens extracted from text. An example of bigram representation of a sentence, "Today is sunny" would be ("today is", "is sunny"). The problem is this would cause B to have too many columns. We should remove the high frequency n-grams (e.g. stop words such as "a", "the") which are not useful, and low frequency n-grams which are consisted of typos, rare n-grams (prevent overfitting). We should keep the medium frequency n-grams, they are the most representative of the sample set. To filter the n-grams with high and low frequency, we use two measures: **Term frequency(TF)** and **Inverse document frequency(IDF)**.

$$\text{TF: } td(t, d) = \text{Frequency of n-gram } t \text{ in document } d$$

We can use the following ways to calculate TF: binary (0, 1), raw count $f_{t,d}$, term frequency $f_{t,d} / \left(\sum_{t' \in d} f_{t',d}\right)$ and log normalization $1 + log(f_{t,d})$.

$$\text{IDF: } idf(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

Where D is the set of all documents in corpus. $|\{d \in D : t \in d\}|$ is the set of document where the term t appears.

$$\text{TF-IDF : } tdidf(t, d, D) = td(t, d) idf(t, D)$$

is a useful quantity to rank the terms (n-grams). Large td-idf value gives terms that are abundunt in a small number of documents.

We can get a better BOW by using TD-IDF for B_{ij} and normalize each row with L_2-norm.

Logistic regression can be used to do sentiment classification

$$p(y = 1 \mid x) = \sigma(w^T x)$$

Where x is rows of BOW matrix.

In the case of a large dataset distribute across machines, we need to map n-gram to column index of the BOW matrix. It is conveninent to use hash mapping to column indices.

$$ngram \to hash(ngram) \mod 2^{20}$$

Hash function can be defined as

$$hash(s) = \sum_{i=0}^{n} s[i] p^i$$

where s is a string, p a given prime and $s[i]$ is the ith charCode.

In the example of spam filtering, we might want to customize for each user. So the term t might be a spam word for user A but not other users. To do this, we change the hash function to

$$hash_u(s) = hash(u + \text{"\_"} + s) \mod 2^b$$

Where u is the user id string and $+$ is string concatenation. In this way, "userA_spamword" and "userB_spamword" are basically different words customized for A and B.

Deep Learning approach

BOW matrix is very sparse and high dimensional, we use Word2Vec embeddedings which are dense vectors in a much lower dimension. Analogous to ngram, we use 1d convolution to achieve the same thing. Given a sequence of tokens $s_0, \ldots, s_n$ and embeddings dimension of k, we compute the $n \times k$ embedding matrix for the sequence where the ith row is the embedding vector for s_i.