A sentiment analysis on hotel reviews

December 13, 2020

1 Overview of the project

In this project we explore some datasets containing hotel reviews and build two models predicting their ratings by extracting relevant information about them.

We start by tokenizing the corpus, extracting subjective words by marking negations and part of speech tags (POS). Secondly, we train a skip gram model to represent each word-token by a vector, hopefully capturing its semantics and hence the sentiment it was meant to convey.

Then, we train two models: a Naive Bayes classifier as a baseline model, and a linear kernel Support vector machine (SVM) as an upgrade, to classify the embedded word-vectors into their proper categories (i.e. ratings) on the training set, and evaluate them on the "dev" set.

Finally, using the (already trained) SVM classifier, we attempt to predict the ratings of the reviews in the "test" set.

2 Pre-processing: tokenization

The task of word tokenization consists of breaking the totality of the text corpora into individual tokens which can be normalized and treated individually. For semantic analysis, it is important to differentiate between the *objective* and *subjective* words, since the latter are those that most likely contain sentiment (Jurafsky and Martin, 2020).

To this end, after breaking each review into a collection of words, we use *nltk*'s POS tagger to identify *verbs, adverbs* and *adjectives* and extract them while discarding the rest. While a more fine grained approach is likely to lead to an improvement, for simplicity sake we stick with this approach. Furthermore, we also mark negation on words succeeding negating tokens like "not" or "didn't" since those alter the meaning of subsequent words (e.g. the verb "like" takes the opposite semantic meaning when coming after negation).

3 Word embeddings via Word2Vec

A common approach to infer the meaning of a word is to look at its context, so if two words arise in the same context they should be closer in meaning. A neat way to visualize this is to think of the totality of words as points a in vector space where each dimension captures a specific meaning.

To do this, we assume a probability distribution on the number of occurrences of each word w_i within a given context of fixed radius $\{w_{i-r}, ..w_{i-1}, w_{i+1}, ..w_{i+r}\}$. If two words have similar probability distributions we say that they are similar.
In the simple case we take 2 documents and count word occurrences for each, and for each word we get 2 probability distributions, one for each context (document). Over 2 large corpora we should get very similar distributions for a given word. The Word2Vec models does just that, and comes in two variants:

- In the *Continuous bag of words*, for each word in the training set, we associate a single context and we estimate a function (in this case a neural network) which maps each context to its word.
- While the *Skip-gram* model takes a word as input and predicts its context, the CBOW model does the inverse.

For tasks heavily depending on capturing the semantic meaning of words, such as sentiment analysis, the skip gram model is known to perform better. While for other tasks such as creating a generative language model, e.g. for a chatbot, the CBOW is more accurate since it is better at capturing the syntactic (grammatical) information of the training data.

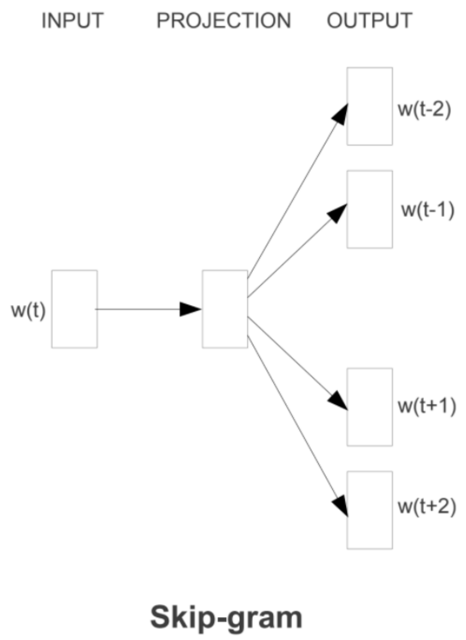

Figure 1: The computational graph of the skip-gram variant of Word2vec.

For a given context, we create a matrix of columns one-hot encoding each of the

context words and we feed this matrix the first layer.
Since the neural network has word-embeddings as parameters, the 0-1 elements of the one-hot encoding copies the relevant word-embedding which we update by using gradient descent.

3.1 Model parameters

We used the default parameters of the gensim library for Word2Vec:

- **size**: (default 100) The number of dimensions of the embedding, e.g. the length of the dense vector to represent each token (word).
- **window**: (default 5) The maximum distance between a target word and words around the target word.
- **min count**: (default 5) The minimum count of words to consider when training the model; words with an occurrence less than this count will be ignored.
- **workers**: (default 3) The number of threads to use while training.
- **sg**: that we set to one, i.e. using the Skip gram rather than the default CBOW model.

4 Support Vector Machines for classification

Perhaps the most commercially important task in NLP is to estimate customer satisfaction with the product and/or other *affective* information in general such as mood, personality and attitude. In this case we interested in inferring the 1-5 ratings of hotel customers from their reviews.

Fortunately, in the word embedding approach, the direction of a word vector captures its semantic (hence its sentiment), such that two similar words have small cosine distance. Therefore, we can represent a given review by taking the sum of the vectors corresponding to the words contained in said reviews, such that for each review we have a pair of points: a 100 dimensional vector encoding its sentiment and the rating of said review which is read from the end of each line in the training data.

On this set of vectors and their corresponding categories (i.e. ratings), we train two classifiers:

- A *Naive Bayes* model, where the naive quality comes from the assumption that each entry of the vector representation is probabilistically independent of all others.
- And a *Support Vector machine* which consists of estimating the coefficient of vectors supporting hyperplanes that classify the data points (vectors) into their 5 distinct categories.

As expected the SVM performs better, the resulting error rates are as follows:

SVM error rate	**Naive Bayes error rate**
0.411	0.598

Predicting a 1-5 five rating is a known hard classification problem (Fang X. and Zhang J., 2015), it is interesting however that when we shift to a less fine grained rating of tolerating error within a one point difference between the predicted rating and the ground truth, we have a much improved classification result:

SVM error rate	**Naive Bayes error rate**
0.031	0.175

5 References

Fang X.; Zhang J., 2015. Sentiment analysis of product review data, Journal of big data.

Jurafsky D.; Martin J.H., 2020. Speech and Language Processing, (see Jurafsky's personal web page for the latest edition).