Sentiment Analysis on Movie Reviews

Nicolas Angelard-Gontier
260532513
McGill University
nicolas.angelard-gontier@mail.mcgill.ca

Abstract

In this paper we explore and compare the accuracy of two different types of algorithms on two movie review sentiment classification tasks. The first algorithm that we try is a Random Forest classifier with bag-of-word features and word-to-vec features. The second is a Long Short-Term Memory (LSTM) Recurrent Neural Netowrk (RNN). We first classify IMDB movie reviews in a binary format: positive (1) or negative (0) and observe a final accuracy of 87%. The second task consists of classifying Rotten-Tomatoes reviews into five sentiment classes: positive (4), somewhat positive (3), neutral (2), somewhat negative (1), negative (0). For this much harder task we will observe a final accuracy score of 61%. We conclude by observing that the LSTM network only slightly improves the Random Forest classifier accuracy, and that combining both IMDB and Rotten-Tomatoes data sets is usually better.

1 Introduction

People express their emotions with language that is often obscured by sarcasm, ambiguity, and sometimes inside jokes. The sentiment analysis challenge consists of capturing the author's emotion in a text by classifying its sentences into positive or negative statements. More recently the challenge has been to also classify neutral sentiments. Natural language being so subtle, it makes sentiment analysis a challenging task in machine learning and can even also be very misleading for humans.

In this project, we dig into sentiment analysis of movie reviews. The work of this project is inspired by two Kaggle competitions (Kaggle, 2014a; Kaggle, 2014b) closed in 2015. The idea is to collect data from both competitions to have a big training corpus, and experiment with a Random Forest and an LSTM Recursive Neural Network (RNN) algorithm to classify each competition's test data set. Our first hypothesis is that extending each training set with the other should yield better results (ie: more data is better). Furthermore we would like to test how well does the LSTM network performs compared to a simpler classification algorithm: Random Forest. To do so we will make prediction submissions to both of the Kaggle competitions and see how well our algorithms performs on each test set. Note that even if a Kaggle competition is closed we can still submit prediction files. Our score will not be added to the leader board but we are able to get accuracy feedback.

2 Method

In this section we describe the data set we use, and the models that we implement.

2.1 Data set

In this project we used two publicly availabe movie reviews data sets. One is made of IMDB reviews[1] and the other is made of Rotten-Tomatoes reviews[2]. In the rest of this paper we will refer to these two data sets as the 'IMDB' data and the 'Rot-Tom' data respectively.

The IMDB data is composed of $25,000$ binary

[1] https://www.kaggle.com/c/word2vec-nlp-tutorial/data

[2] https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data

labeled training reviews, $50,000$ unlabeled training reviews, and $25,000$ unlabeled test reviews. The goal of the competition is to use the $75,000$ training reviews to make some predictions on the $25,000$ test reviews. Each review in this data set is a full length string written by a user. A label in this case is a binary flag indicating if the review is positive (1) or negative (0). Note that we have the same amount of positive and negative reviews in the labeled training set (ie: $12,500$ for each label).

The Rot-Tom data is composed of $8,544$ reviews divided into $156,060$ labeled training samples and $3,310$ reviews divided into $66,291$ unlabeled test samples. The goal of the competition is to use the training samples to make some predictions on the test set. Sochet et al. (2013) used Amazon's Mechanical Turks to create fine-grained labels for all parsed phrases in the corpus. A label is an integer between 0 and 4 indicating if the review is positive (4), somewhat positive (3), neutral (2), somewhat negative (1) or negative (0). In this data set, each review has been truncated into smaller parts. Thus, each sample is either a full sentence or a few words that was in a sentence. For example we can find "`A series of escapades demonstrating the adage that what is good for the goose is also good for the gander , some of which occasionally amuses but none of which amounts to much of a story.`" with a score of 1 as well as "`good for the goose`" with a score of 3. Note that this data suffers from the class imbalance problem since we have $7,072$ negative, $27,272$ somewhat negative, $79,556$ neutral, $32,927$ somewhat positive and $9,206$ positive samples.

The first thing we did in this project is to expand each data set with the other. We did so by taking positive (4) and somewhat positive (3) **sentences** (and not part of sentences) from the Rot-Tom data and included those into the IMDB data as positive reviews. Similarly, we took negative (0) and somewhat negative (1) **sentences** from the Rot-Tom data and included those into the IMDB data as negative reviews. This resulted into $15,771$ negative and $16,102$ positive IMDB reviews.

When extending the Rot-Tom data we took all positive IMDB reviews and marked them as positive (4) Rot-Tom samples. Similarly we included all negative IMDB reviews as negative (0) Rot-Tom samples. This resulted in $19,572$ negative, $27,272$ somewhat negative, $79,556$ neutral, $32,927$ somewhat positive, and $21,706$ positive samples. Note that we didn't add any IMDB reviews to the somewhat positive or somewhat negative classes as they already had a lot of samples. Thus we tried to minimize the class imbalance problem.

Eventually, we preprocessed each review from both data sets by removing punctuation, putting every word to lower case, removing hyperlinks or anything that wasn't word or numbers. We sometimes removed stop words from the reviews as well.

2.2 Models

In this section we present the two classification algorithms we implemented to solve the two proposed Movie Review Sentiment Analysis tasks.

2.2.1 Random Forest

We first explored the Random Forest algorithm on the two classification tasks. The first step was to represent each review as a set of features. We considered two different sets of features: bag-of-words (BoW) features and word-to-vec (W2V) features.

In the BoW case, we first learn a vocabulary of words from all train reviews. We then model each review as a vector of word counts for each vocabulary word. Note that in order to limit the vocabulary size we removed stop words, and took the $10,000$ most frequent words from the corpus.

In the W2V case, we used the python library 'gensim' [3] to learn word embeddings for our corpus. A word embedding is a parametrized function that maps words to high dimensional vector space. To learn such a function we used the skip-gram model described by Mikolov et al.(2013) with 10 surrounding words. In our case we learned word embeddings of dimension 500. We used both labeled and unlabeled (only available for IMDB data) training reviews without removing stop words to learn the embeddings. Once we have a vector representation for each word in our vocabulary, we created a feature vector for each review by averaging all word embeddings present in that review. Note that at test time, if a word doesn't have any embedding (ie: it wasn't seen during training time) we simply ignore

[3] https://pypi.python.org/pypi/gensim

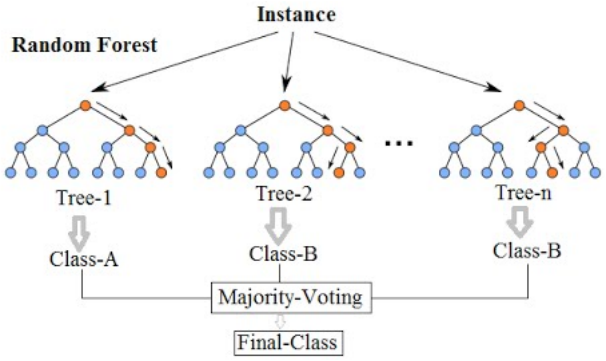

Figure 1: Random Forest Simplified from (Be Expert in Minutes, 2016)

this word when computing the average word embedding for that review.

Eventually, once we defined the features we will use, we started to look at the Random Forest algorithm using the python library 'scikit-learn' [4]. This is an ensemble method consisting of a large collection of decision trees (500 in our case). Each decision tree is trained on multiple random subsets of training reviews. As shown in Figure 1, once each tree has been trained, the Random Forest algorithm classify an input review according to each of its trees by outputting the majority class for which that review has been classified.

2.2.2 LSTM RNN

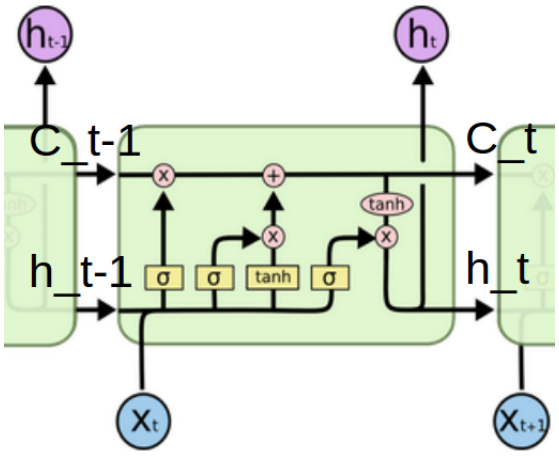

Figure 2: LSTM unit details from (Olah, 2015)

The second algorithm we implemented is an LSTM recurrent neural network using the python library 'theano' [5]. The advantage of using LSTM hidden units instead of regular $tanh$ units is to better capture long-term dependencies in each review. The idea is to use previous events (words in our case) in order to inform later ones, allowing information to persist over time. Here each review represent a time series of event where each event is "observing a word". As described in Figure 2, in our model an LSTM unit at time t is made of a forget gate f_t, an input gate i_t, a cell state C_t, and an output gate o_t, each defined like so:

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f)$$

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$$

$$C_t = tanh(W_C \cdot x_t + U_C \cdot h_{t-1} + b_C)$$

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o)$$

where $W_{f,i,C,o}$ and $U_{f,i,C,o}$ are weight matrices and $b_{f,i,C,o}$ are bias vectors, x_t is the observed word at time step t, and h_{t-1} is the representation of the review from the previous LSTM unit. The forget gate first decides what to forget in the cell state by outputting a 0 or a 1 for each value in the cell state C_{t-1}, the input gate then decides what to update in C_{t-1} while the cell state at time t will output a set of values to write. Eventually, the output gate decides what information to output from our new cell state. The actual output of the LSTM unit at time t is:

$$h_t = o_t \times tanh(C_t)$$

Our model is composed of a single LSTM layer (of 300 units), followed by an average layer and a logistic regression layer as illustrated in Figure 3. From an input review with words $x_1, x_2, ..., x_n$ each memory cell in the LSTM layer will produce a representation $h_1, h_2, ..., h_n$. These representations are then averaged to get the final review representation h, which will be fed to a logistic regression classifier whose target is the class label associated with that input review. The loss function that we try to minimize is the average negative log probability of the target class defined like so:

$$cost = -\frac{1}{n}\sum_{i=1}^{n} log(p_{targetclass} + \epsilon)$$

[4] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[5] http://deeplearning.net/software/theano/

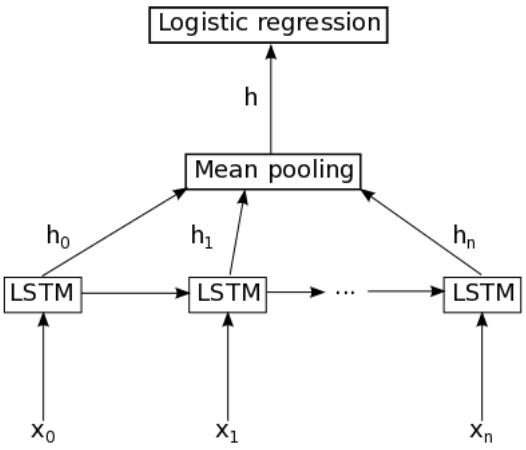

Figure 3: LSTM Network model

with ϵ being a small constant to avoid having infinite cost. We use ADAM as our optimizer since it is an improved version of Stochastic Gradient Descent (Kingma and Ba, 2014). The training of the model was done on 90% of the training data, keeping 10% for the validation set. Training and validation scores were measured after each epoch and the best parameters were saved. If no improvement was done on the validation score for five consecutive epochs we stopped training and produced test set predictions with our best model parameters.

3 Results

When submitting our predictions to each competitions we first noticed that the Rot-Tom data set was much harder to predict than the IMDB one. Indeed, having multiple sentiment classes increases the difficulty of the task. More over the Rot-Tom data is composed of small parts of reviews of only a few words which are very difficult to classify by themselves. In fact, complex time-series related models like Recurrent Neural Networks (RNNs) are useless for these kind of short reviews since the main idea of RNNs is to capture dependencies on long time-series. As we can see from Table 1 extending (e) one data set with the other is usually better except when using the LSTM network classifier where the original data set by itself (o) produces better accuracy scores. This suggests that the LSTM model is more sensitive to training data, which may be explained by the big amount of parameters to tune (tens of thousand in our case). On the other hand we can see that the LSTM model outperforms the Random Forest classifier, even if that baseline algorithm produced quite strong accuracies. This is probably due to the fact that both BoW and W2V features loose the word order of a review in contrary to the LSTM model which is based on that order. Eventually we can say that in general BoW features seems to be better than W2V features. This is mostly due to the fact that averaging word vectors to represent a review loses the word order, making it very similar to BoW features. In addition, W2V features were originally trained on a much bigger data set than this one (Mikolov et al., 2013)

	IMDB score	**Rot-Tom score**
RF+BoW(o)	0.85904	0.58497
RF+BoW(e)	**0.86088**	**0.59054**
RF+W2V(o)	0.83492	0.570901
RF+W2V(e)	**0.83704**	**0.59509**
LSTM(o)	**0.87379**	**0.61193**
LSTM(e)	0.87376	0.59117

Table 1: Test accuracy for Random Forest, and LSTM network

4 Discussion & Conclusion

We conclude by mitigating our initial hypothesis since combining both IMDB and Rot-Tom data is not always the best solution, especially for the LSTM network. Further more, we can accept the hypothesis that the LSTM network model performs better than the Random Forest baseline, even if the difference is small. One extension for the Random Forest algorithm would be to use word2doc features instead of averaging word2vec embeddings to try keeping the words order. We were surprised to see how small the difference was between the Random Forest classifier and the LSTM network accuracies. One extension to make our LSTM classifier more robust would be to use a more complex classifier than a logistic regression at the top layer. One can also think about stacking multiple LSTM layers or having a recursive tree-structured network rather than a recurrent network. This seems of particular interest in the case of the Rotten Tomatoes data where we sometimes have very short snippets of movie reviews to classify.

Acknowledgments

Thank you to Professor Jackie Chi Kit Cheung for the great class.

References

[Be Expert in Minutes2016] Youtube Be Expert in Minutes. 2016. Random forest based classification. [Online; accessed 10-November-2016].

[Kaggle2014a] Kaggle. 2014a. Bag of words meets bags of popcorn. [Online; accessed 10-November-2016].

[Kaggle2014b] Kaggle. 2014b. Sentiment analysis on movie reviews. [Online; accessed 10-November-2016].

[Kingma and Ba2014] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

[Mikolov et al.2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.

[Olah2015] Christopher Olah. 2015. Understanding lstm networks. [Online; accessed 10-November-2016].

[Socher et al.2013] Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. *EMNLP 2013*.