COMP 551: Assignment 4

Omar Abdel Baky
David Carrier
Mounir Hamdar

December 14, 2019

Abstract

In 2014, Le and Mikolov[5] suggest the *Paragraph Vector* method for sentiment analysis, an unsupervised method meant to overcome the weaknesses of bag-of-word methods (BoW). We show that on a dataset of 25k IMDb reviews, the Paragraph Vector method performs significantly better than neural network-based methods (like LSTMs, RNNs and CNNs) but does not perform significantly better than some sophisticated BoW methods. Moreover, a simple modification of the original NB-SVM approach using word embeddings yields an error rate of 7.5%, which is within 0.1% of the Paragraph Vector result.

1 Introduction

In sentiment analysis, the objective is to detect what type of sentiment is expressed by a particular snippet of text. One of the subproblems of interest is binary sentiment classification, where the question is to tell whether a piece of text expresses negative or positive feelings towards a particular entity.

Sentiment analysis is a branch of natural language processing (NLP) and as such requires exploring the tools used in the discipline.

1.1 TF-IDF Encoding

All the baseline models evaluated in this paper used Bag-Of-Words representation of the data. Instead, we propose using TF-IDF weighting, which gives more importance to documents that appear rarely in the corpus, and downweights common terms. This makes the models less likely to mistake noise for patterns. More precisely, the TF-IDF statistic for each term t in a particular document d is computed as:

$$\text{TF-IDF} = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \cdot \log\left[\frac{N}{|\{d \in D : t \in d\}|}\right]$$

where f is a frequency, D is the corpus, and N is the total number of documents in the corpus ($|D|$).

1.2 Downsides of BoW approaches

Bag-of-Words approaches are vectorization methods that map a specific comment to a vector related to the presence of words or n-grams in a comment, regardless of their order within the comment. Common such approaches include *Count Vectorization*, where the count of a specific word or n-gram forms the vectorizaztion, or various scaling methods based on count such as TF-IDF. These methods have the merit of preserving a lot of the content of the comment and to be easy to compute. However, the vectors they produce have dimension of order $O(|v|^n)$ where v is the vocabulary and n is the length of the gram considered. They also ignore the ordering of the words in the comment: the sentences

This movie is a very necessary watch and is not boring.

and

This movie is not a necessary watch and is very boring.

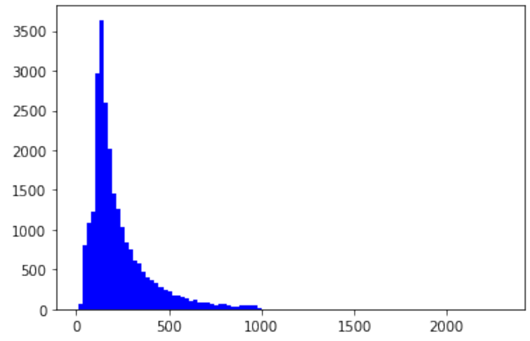
get mapped to the same vector under unigram BoW vectorizations since they contain the same words, even though they express very different messages. In practice, these types of scenarios do not occur frequently, and can be countered partially by increasing the length of the n-grams considered. However, this inability to read word orderings properly puts a theoretical cap on the accuracy that can be reached by a classifier, which is why approaches that are not reliant on BoW vectorizations are also studied.

2 Dataset

The dataset used for comparison purposes is Andrew Maas's *Large Movie Review Dataset*, available at `https://ai.stanford.edu/~amaas/data/sentiment/`. This dataset contains 50,000 movie reviews, including 25,000 training examples and 25,000 test examples. These comments are described as "highly polarized" movie reviews in English, with no more that 30 reviews coming from a specific movie. These reviews are then split into positive and negative categories, which we then have to classify in either of these categories.

The comments have a mean length of 228 words, with the shortest comment having 10 words and the longest one having 2308 words.

Figure 1: Histogram of comment lengths



Since its release, the dataset has been used in hundreds of papers, with state-of-the art approaches reaching accuracies of 97.42%[4].

3 Original results

3.1 Baseline Models

The authors report multiple baseline models evaluated on this dataset. We evaluate the following six baselines:

Baseline Model	Error rate
Unigram MNB	16.45%
Bigram MNB	13.41%
Unigram SVM	13.05%
Bigram SVM	10.84%
Unigram NB-SVM	11.71%
Bigram NB-SVM	8.78%

We first note that all the above models are trained on binary representation of document term presence rather than TF-IDF encoding. We will demonstrate that using TF-IDF encoding instead substantially improves accuracy. Additionally, hyperparameter tuning will yield positive results. Moreover, we will explore **four additional models, and demonstrate a modification of the original NB-SVM approach that achieves better results**.

3.2 NB-SVM Model

The authors discuss the NB-SVM model developed in *Baselines and bigrams: Simple, good sentiment and text classification*[1]. In this approach, we use Naive-Bayes features - log-odds ratios computed from term frequencies - as feature inputs used to train a Support Vector Machine (SVM). More formally, we let f_i be the feature vector of term counts for a given training case i, with label $y_i \in \{-1, 1\}$. We then define $\mathbf{p} = \alpha + \sum_{i:y_i=1} f_i$ and $\mathbf{q} = \alpha + \sum_{i:y_i=-1} f_i$, where α is a smoothing parameter. This gives us a log-count ratio $\mathbf{r} = \log(\frac{\mathbf{p}/\|\mathbf{p}\|_1}{\mathbf{q}/\|\mathbf{q}\|_1})$. Then, we represent a binarized rather than count representation of the data instead, with $\hat{f}_k = I\{f_k > 0\}$, I being the indicator function. We compute $\hat{\mathbf{p}}$, $\hat{\mathbf{q}}$, and $\hat{\mathbf{r}}$ as previously, but with $\hat{f}_i$ rather than f_i. The NB-SVM model solves the standard SVM minimization problem, but sets $x_k = \hat{r} \cdot \hat{f}_k$, the elementwise product. Le and Mikolov report an error rate of 8.78% with this model, rivalling state-of-the art results with a fraction of the computational cost. We will propose a modification to this method that reduces the error rate to 7.5%.

3.3 Paragraph Vector Model

A common method for NLP tasks is to learn vector representations for words, these vector representations can capture the semantics of words. They allow models to compare the meaning that words have to make predictions about the text. The paragraph vector method is inspired by the preceding method, the aim is to learn a unique vector representation for each word stored in a Matrix as well as unique vector representation for each paragraph. The current Paragraph vector and word vectors can be concatenated or averaged out to predict the next word in the context. The paragraph Vector representations are learned by training neural network to predict the paragraph vector using stochastic gradient descent. The predicted paragraph vector is then used to predict next words in randomly sampled sections of fixed length from the text. The paragraph vectors can be used as features for non-neural network machine learning techniques.

4 Proposed Models and Improvements

4.1 Hyperparameter tuning

We conduct extensive hyperparameter tuning using grid search. This results in increased accuracy in most baseline models.

4.2 Recursive Neural Networks

Recursive neural networks are a type of neural network where the computation graph contain cycles. They are suited to process sequential data where memory of the previous time steps is required in order to reach a satisfactory conclusion: examples of such scenarios include time series and text analysis. Indeed, we can use as a single time step a word in a phrase. We took a look at two different types of RNNs to use as a comparison base; unfortunately, these models did not perform very well. As overfitting was occuring early, we suspect that these approaches would yield an advantage in situations where more training data is available.

4.3 Convolutional Neural Networks (CNN)

Convolutional neural networks are a variant of a neural network where some of the layers are replaced with convolutional layers, which perform discrete convolutions with a learned convolution mask onto the data. Convolutional neural networks are space invariant, making them robust to variations in feature positions. This can be a useful property for the task at hand: since we are attempting to do sentiment classification, we might be more likely to find predictive power in the *set* of words present in a document rather than in their position in a sentence. Although convolutional neural networks are mostly known for their performance on image classification problems, they have often been shown to achieve state-of-the-art results on natural language processing problems [2]. We indeed demonstrate that a CNN is one of the best performing models on this task.

4.4 Modified NB-SVM Model (Word embeddings)

The original paper proposes an NB-SVM model in which log-odds ratios are used as features to train an SVM. We propose a slightly different approach using word embeddings, inspired by an article written by Arun S. Maiya[1]. We first represent movie reviews as a collection of words each represented by a unique ID, and for each word, we compute a Naive Bayes log-odds ratio. We then encode this as an embedding layer **R**. This is similar to the original NB-SVM approach, but using an embedding layer structure makes model training more computationally efficient. We then create a second embedding layer, **W**, storing weights associated with each word ID for any given document. These weights are initialized with a normal distribution. Similar to the original model, we then take the dot product of the two embedding layers, $\mathbf{R} \cdot \mathbf{W}$. We then use a sigmoid activation function to get a final prediction. This becomes a network where all weights can be learned using standard optimization algorithms such as gradient descent. As we will discuss in the **Results** section, this is our best performing model on this dataset. It is implemented in *nbsvm.ipynb.*

5 Results

5.1 Baseline Models

By using TF-IDF encoding and aggressively tuning the baseline model hyperparameters using grid search, we achieve results better than those reported by Le and Mikolov. All SVM models use the squared hinge loss function. Our error rates on the MNB and SVM models are:

Model	**Le and Mikolov Error Rate**	**Our Error Rate**	**Our Hyperparameters**
Unigram MNB	16.45%	16.82%	$\alpha = 2.0$
Bigram MNB	13.41%	**13.03%**	$\alpha = 0.43$
Unigram SVM	13.05%	**11.50%**	$C = 0.25$
Bigram SVM	10.84%	**9.76%**	$C = 0.5$

5.2 Modfied NB-SVM (Word embeddings)

We train this model by using the *Adam* optimization algorithm. We get best results by using trigram encoding of the original data. Impressively, this modified approach to NB-SVM only takes a few seconds to train - substantially better than the neural network methods tested - yet **achieves an error rate of 7.50%**, the smallest rate we obtained on this dataset. This rivals the authors' reported state-of-the-art result of 7.42% using paragraph vectors. Given the practical equality of these two error rates, the computational efficiency of our modified NB-SVM model might make it a better approach for sentiment classification tasks. Our final results are:

Baseline Model	**Le and Mikolov Error Rate**	**Our Error Rate**
Unigram NB-SVM	11.71%	**10.61%**
Bigram NB-SVM	8.78%	**7.97%**
Trigram NB-SVM	N/A	**7.50%**

5.3 Recursive Neural Networks

In the RNN-based models below, we truncated comment lengths to 384 words for the LSTM and 256 for the GRU. We padded the comments with zeros if they had insufficient length. This is to be able to

[1]https://medium.com/@asmaiya/a-neural-implementation-of-nbsvm-in-keras-d4ef8c96cb7c

efficiently use minibatch *Adam* optimization on GPU, as it requires a uniform length across a given batch. A binary crossentropy loss was used.

The models used an embedding layer containing 512 dimensions, learned from the 15000 top words in the training set. We then used a dense layer of width 128, whose output was then fed into RNN cells (LSTM or GRU). Its output was then followed by two dense layers of width 512 and 128, respectively, before a binary output. All activations were hyperbolic tangent, except for sigmoid activation at the last layer.

At every layer, we used a dropout factor of 0.3 for the LSTM model, except for the last layer that had a dropout of 0.5.

The attempts using recursive neural networks gave disappointing results. Models would quickly reach a state of overfitting after 5-10 epochs. Increasing regularization caused either model to not converge. We therefore suspect that the dataset had insufficient data in order for either of these approaches to be viable.

Attempts were made to avoid overfitting while keeping convergence by decreasing regularization (dropout), reducing the number off RNN cells, as well as reducing the width of the word embeddings and the cells. None of these approaches gave satisfactory results or even resulted in consistent convergence of the models beyond results attributable to chance.

Baseline Model	**Our Error Rate**	**Training Epochs**
LSTM	15.93%	13
GRU	15.61%	9

5.4 Convolutional Neural Networks

Convolutional neural networks performed well on this dataset. We notice that CNNs start overfitting quickly on this dataset, and that using very deep structures also led to overfitting. In the end, we implement a simple network with an embedding layer of the 5000 most common words, a single convolutional layer of width 64, a max pooling layer, and a dense layer. Most attempts at complexifying this structure led to worse results. In the end, this architecture yielded an error rate of 10.27%, which is competitive with the baseline models of the paper, and in fact beats them all except the Bigram NB-SVM model.

6 Conclusion

In conclusion, we matched or improved on all of the baselines presented in the Le and Mikolov paper[**?**]. Additionally we proposed several new methods, being the LSTM, GRU and the Tri-gram NB-SVM. The Tri-gram NB-SVM model reached an accuracy comparable to the paragraph vector method while taking only seconds to train. This suggests that non-neural network methods, particularly Bag-of-Words (BoW) approaches, may be viable in practice, especially when dealing with small datasets.

7 Statement of Contributions

1. Omar Abdel Baky: LSTM and GRU networks applied to word vector representations learned from a pretrained library.

2. David Carrier: LSTM and GRU networks, brief attempts at various bag-of-words approaches not specified here.

3. Mounir Hamdar: NBSVM, Convolutional Neural Network, SVM, and MNB models.

References

[1] https://www.aclweb.org/anthology/P12-2018.pdf

[2] https://arxiv.org/pdf/1702.01923.pdf

[3] https://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf

[4] https://www.aclweb.org/anthology/P19-2057/

[5] http://proceedings.mlr.press/v32/le14.pdf

1