Sentiment Analysis with Word2Vec and XGBoost

Sheng Chien

September 15 2018

1 Definition

1.1 Project Overview

Sentiment analysis (also called opinion mining) is a method of computationally identifying the underlying emotion from a piece of text. It's important in NLP field as to truly communicate with humans, computers would need to understand human emotions. It is also valuable for a business to know if their customers are satisfied.

Sentiment analysis has a long history dated back to the beginning of 20th century. Sentiment analysis is a fast growing research area in recent years. 99% of papers have been published after 2004 [13]. It shifted from analyzing online product reviews [14] to more social media texts from Twitter [6]. And many topics extend beyond by utilizing sentiment analysis such as stock prediction [12].

1.2 Problem Statement

The problem to solve is to computationally classify a polarized sentiment (positive or negative) behind the expressed words of a movie review. Our objective contains two parts. First is to map a review document d to a fixed-size vector representation of words ϕ_w using a word embedding converter function $g(d)$. The word vector is then mapped to a predicted sentiment label $\hat{y}$ by using a classifier function $f(x)$.

$$\phi_w = g(d). \tag{1}$$

$$\hat{y} = f(\phi_w). \tag{2}$$

It's noticeable that we can also improve word vector ϕ_w to achieve better prediction of the review sentiment.

1.3 Metrics

The evaluation metric used to quantify the result is accuracy. Basically, to compare which model can predict most correct sentiments on the test set. It's defined as:

$$accuracy(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} 1 \times (\hat{y} = y) \tag{3}$$

The n is the number of test samples. The y is the truth while $\hat{y}$ is the predicted value. Since the test set is split equally among positive and negative reviews, accuracy is an appropriate measurement as the random guess would only get us 50% accuracy. The model is expected to be a lot higher than the random prediction.

2 Analysis

2.1 Data Exploration

The dataset chosen is Large Movie Review Dataset which was originally collected by Maas, et al. [8]. It can be downloaded from [5]. The dataset contains large amount of movie reviews from the Internet Movie Database (IMDB). Since there are unbalanced reviews among movies, no more than 30 reviews are allowed per movie. They constructed a collection of 50,000 reviews which are equally split into training and test sets. Each has 25,000 reviews. The original star rating {1..10} are linearly mapped to $\{0, 1\}$ (bad or good). A negative review if $rating <= 4$, and a positive review if $rating >= 7$. In addition to the labeled data, an extra unlabeled data of size 50,000 are provided.

The first thing to explore is to find the most common words. They normally provide little information so that they are usually treated as stop words and filtered out in the preprocessing step. Table 1 shows the top 50 frequent words from the training set, both labeled and unlabeled.

50 Most Common Words									
the	br	was	but	he	one	who	there	out	good
and	it	as	film	are	all	an	her	has	can
of	in	with	you	his	at	from	or	if	more
to	this	for	on	have	they	so	just	what	when
is	that	movie	not	be	by	like	about	some	very

Table 1: Top frequent words ordered by count

The emoji is widely used nowadays to express the feeling. Part of the exploration is to identify if they could be used as a good indicator of the review expression. There are only 50 comments containing ":-)" found in the labeled training set. Among them, 29 are positive reviews while 21 are negative reviews.

2.2 Exploratory Visualization

In a natural language, there are few words used very frequently. A greater number of words are used with lower frequency. And many words are hardly used. They normally follow Zipf's Law. Not surprisingly, as shown in Figure 1, the movie reviews data also follow the Zipfian distribution.

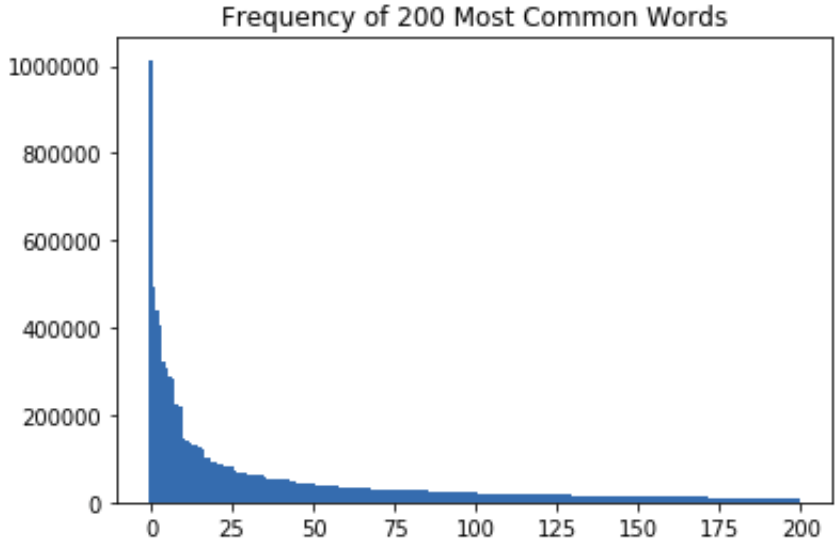

Figure 1: Word frequency distribution of top 200 words from the movie reviews

2.3 Algorithms and Techniques

2.3.1 Word2vec

Word2vec is a powerful model capable of learning the underlying vector representations of words called "word embeddings" that can be used to predict both semantic and syntactic relationships published by Mikolov et al. [9]. A famous example is "King - Man + Woman" results in a vector very close to "Queen" [11]. Another classic example is "Madrid - Spain + France" results in a vector closest to "Paris" [10]. There are two types of architecture: Continuous Bag-of-Words (CBOW) and Skip-gram (Figure 2).

2.3.2 CBOW

CBOW model predicts the current word $w(t)$ based on the context with the size c, c words before and c words after $w(t)$. The model is a neural network with only 1 hidden layer. Instead of a fully connected layer, the model uses a simplified layer called projection layer. The same input weights $WI_{d\times V}$ is used to do the

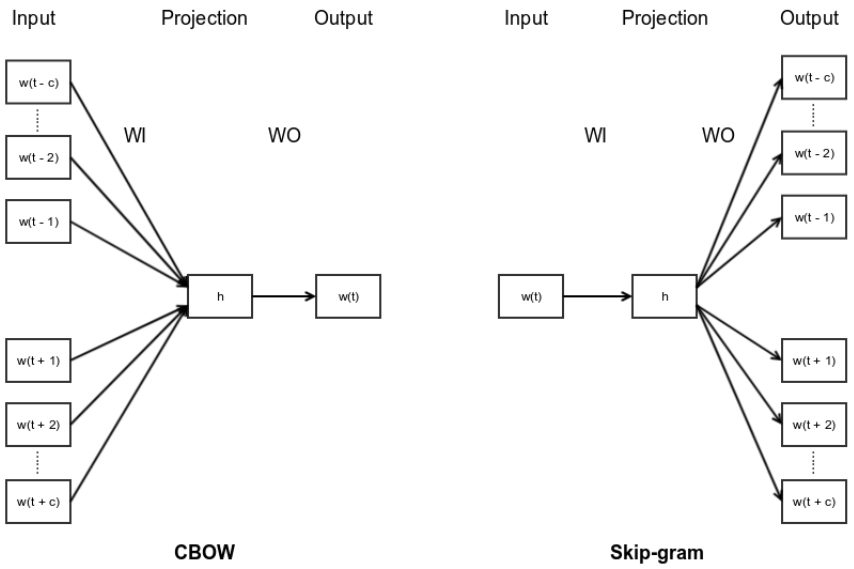

Figure 2: Word2vec model architectures

projection, where i-th column is a d-dimensional embedded vector for word w_i of input. The vectors of the context words are averaged into the hidden layer $h_{d\times 1}$. The output weights $WO_{V\times d}$, where j-th row is a d-dimensional embedded vector for word w_j of output. The output probability is calculated as followed [17]:

$$\begin{aligned} z &= WO\times h \\ \hat{y} &= softmax(z) \end{aligned} \tag{4}$$

The objective function is the cross entropy where the loss increases when the predicted probability diverges from the actual label. The gradient descent is then performed to update the corresponding weights.

$$H(\hat{y}, y) = -\sum_{i=1}^{|V|} y_i \ log(\hat{y_i}) \tag{5}$$

2.3.3 Skip-gram

Skip-gram model is similar to CBOW, but the process is reversed. Instead of predicting the current word based on the context, it is trying to predict the context based on the current word. The objective of Skip-gram is to maximize the average log probability where c is the context size for the centered word w_t given a sequence of words $w_1, w_2, ..., w_T$ [10].

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} log\ p(w_{t+j}|w_t) \tag{6}$$

Skip-gram defines the probability $p(w_{t+j}|w_t)$ using softmax function, where v_w and $v_w^{'}$ are the vector representations of word w for "input" and "output" respectively.

$$p(w_O|w_I) = \frac{exp({v_{W_O}^{'}}^{\top} v_{W_I})}{\sum_{i=1}^{|V|} exp({v_i^{'}}^{\top} v_{W_I})} \tag{7}$$

2.3.4 Negative Sampling

One bottleneck of Skip-gram model is that the computation increases linearly to the size of vocabulary. One optimization technique is called Negative Sampling. Instead of computing for every word during the gradient descent, it samples only few words drawn from the noise distribution $P(w_i)$. For Negative Sampling, Mikolov et al. [10] found that the unigram distribution raised to the 3/4rd power outperformed significantly the unigram and the uniform distributions.

$$P(w_i) = \frac{U(w_i)^{3/4}}{\sum_{j=1}^{n}(U(w_j)^{3/4})} \tag{8}$$

2.3.5 Boosting

Boosting is an ensemble technique to create a strong learner by combining a number of weak learners. Different from bagging, boosting adds a weak learner to ensemble sequentially. Each weak learner is trained to focus on correcting its predecessor.

$$\hat{y}^{(t)} = \hat{y}^{(t-1)} + f_t(x_i) \tag{9}$$

2.3.6 XGBoost

XGBoost is one of such boosting algorithms based on Gradient Boosting that performs gradient descent to minimize the objective [2]. That means at t-th iteration, we add the $f_t(x_i)$ to improve our model. And $\Omega(f_t)$ is the regularization to penalize more complex models in favor of simpler ones.

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}^{t-1} + f_t(x_i)) + \Omega(f_t) \tag{10}$$

By using the second-order Taylor approximation, the objective can be optimized into a general form [3].

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{t-1}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t(x_i)^2] + \Omega(f_t) \tag{11}$$

where $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{t-1})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{t-1})$ are the first and second gradient of the loss function [2]. This optimization is computation friendly as the terms $l(y_i, \hat{y}^{t-1})$, g_i, and h_i only need to be calculated once and reused as constant for a given iteration for all splits. A nice full detailed explanation can be found at [1].

Besides regularized gradient boosting and the optimization. There are two algorithmic features highlighted in XGBoost.

- **Sparsity-aware Split Finding** - This algorithm handles missing data values automatically by introducing the default direction. When a value is missing, the instance is classified into the default direction where the optimal default directions are learned from the data.
- **Weighted Quantile Sketch** - This algorithm handles weighted data that an existing quantile sketch algorithm [4] does not support. Weighted Quantile Sketch makes the approximate tree learning more efficiently especially when the data are too large and cannot fit into memory such that the exact greedy algorithm might become too expensive to enumerate over all possible splitting points.

XGBoost is more than a machine learning algorithm. It also incorporates several system designs to make the algorithm run more efficiently.

- **Block Structure** - The data are stored in a compressed column format where each column is sorted by the corresponding feature value. This makes the quantile search a linear problem when the exact greedy algorithm is used. This block structure also enables parallel learning.
- **Cache Optimization** - For the exact greedy algorithm, an internal buffer is allocated in each thread to fetch gradient statistics. For the approximate algorithm, the problem is solved by choosing a good block size to prevent cache miss or inefficient parallelization due to small workload.
- **Out-of-core Computing** - This feature helps when a very large dataset cannot fit into the memory. The first technique XGBoost uses is Block Compression. It trades decompression time with disk reading where the latter is more costly. The second technique is Block Sharding that divides the data onto multiple disks with a thread prefetching the data into a buffer for the training to read alternatively.

Together, they make XGBoost an efficient gradient boosting algorithm in terms of both model performance and execution speed. It's a popular algorithm has been recently dominating in Kaggle competitions.

2.4 Benchmark

One benchmark model compared in this project is the best model reported by Mass, et al. [8], that is, %88.89. Their best result was achieved by employing logistic regression algorithm to train on the word vectors learned by their probabilistic model combined with the original bag of words representation. The measurement for comparison is accuracy, that was used in the paper, to see which model results in more correctly predicted sentiments on the test set. In addition, logistic regression algorithm is included as benchmark in comparison to XGBoost. The reason is there are two parts in this sentiment analysis process, where the first part is to use word2vec model to learn vector representations of words. It will be interesting to just compare these two classifier algorithms. As logistic regression algorithm is often served as the baseline, we can observe how much better (or worse) performance XGBoost could achieve.

3 Methodology

3.1 Data Preprocessing

The only one data preprocessing done was to filter out the stop words. Inspired by [8], the most common 50 words were discarded. We also did not perform stemming as the word2vec model is good at learning different forms of a word including typos. However, we did not limit the vocabulary to only 5000 applied in the paper because their model is probability-based. But the word2vec model is prediction-based which learns the underlying relationship using neural network.

Another preprocessing was considered is the smiley emoji. That was originally thought to be useful to determine good reviews. However, during our data exploration, there are only 50 smiley emojis found out of 25,000 data points. And the reviews with smiley faces did not dominantly to be positive. Therefore, it did not justify a special rule to keep our process simple and generalized.

3.2 Implementation

There are two main parts of our solution. The first part is a unsupervised learning to build vector representations of words. The second part is a supervised learning to classify a sentimental movie review.

To be specific, the word2vec model published by Mikolov et al. [9] was used to perform the unsupervised part. All the reviews in the training set were fed into word2vec model to build the word vectors. Both labeled and unlabeled training data were included to make better model accuracy. A word vector is a fixed-size low dimensional vector, typically 100 to 200. The trained word2vec model was then used to build a document vector (or feature vector) by averaging of all vectors from each word in a review. The document vector along with the labeled sentiment become the input to the classification algorithm for training. The trained classifier were used to make predictions of the sentiments. XGBoost

was chosen for the binary classification task in addition to logistic regression for benchmarking.

3.3 Refinement

The refinement process started with using default parameters of the models involved - word2vec, logistic regression, and XGBoost. The train test split was performed on the training data where 80% are used for training and 20% are used for validation. The default logistic regression model already demonstrated a good accuracy. However, the default XGBoost model started much lower than the original expectation as shown in Table 2.

Model	Accuracy %
Default Word2vec + Default Logistic Regression	85.48
Default Word2vec + Default XGBoost	80.56

Table 2: Accuracy of default run validated using 20% split of the training data.

Since tuning word2vec model has direct impact on the classification accuracy, we first explored word2vec parameters by using the default classification model to observe the improvements. Some heuristics also help check if the relationship the model learned makes sense by listing similar words of some words. Both CBOW and Skip-gram models were tried as well as vector size, window size, minimal count, and negative sampling with different number of noise words drawn. The final parameters used were "size=200, window=12, min_count=2, sg=0, cbow_mean=1, hs=0, negative=5". The intermediate accuracy with default classifiers was recorded in Table 3.

Model	Accuracy %
Tuned Word2vec + Default Logistic Regression	86.60
Tuned Word2vec + Default XGBoost	83.22

Table 3: Accuracy of tuned word2vec with default classifiers validated using 20% split of the training data.

Once the word2vec model was finalized, the grid search with cross validation was applied to find the best model for our classifiers. For logistic regression, the optimal parameters found were "C=1, penalty=L1". For XGBoost, they were "max_depth=6, n_estimators=800, learning_rate=0.04, subsample=0.6". At this point, XGBoost was slightly better than logistic regression as shown in Table 4.

Model	Accuracy %
Tuned Word2vec + Grid Search Logistic Regression	86.52
Tuned Word2vec + Grid Search XGBoost	86.80

Table 4: Accuracy of tuned word2vec with best models from the grid search validated using 20% split of the training data.

4 Results

4.1 Model Evaluation and Validation

In the final testing, the optimal parameters learned during tuning were applied. The classifiers were trained with the full training set and validated with the test set. One thing to point out is although the default logistic regression classifier (C=1, penalty=L2) achieved little bit higher accuracy on the 20% split of the training data than the best model from the grid search, we still adopted the parameters learned by the grid search in the final testing. The reason is the grid search has gone through the 5-fold cross validation which generalized the model better for the unseen data.

As for word2vec model, Table 5 shows an example of the 8 most similar words for two sample words we checked heuristically. They demonstrated our word2vec model was capable of learning different forms of a word including typos and synonyms.

Sample Word	Most Similar Words
recommend	recommended, advise, suggest, reccomend, unrecommended, recomend, recommendable, reccomended
sucks	sucked, suck, stinks, crappy, horrible, terrible, bad, awful

Table 5: Top 8 most similar words

In the final result, our benchmarking logistic regression classifier outperformed xgboost slightly. However, both classifiers achieved higher accuracy on the testing data than the training data which suggests the models were generalized well to handle unseen data. The final accuracy comparison with benchmark models was shown in Table 6.

4.2 Justification

Our best result was in fact achieved by the benchmark model logistic regression instead of XGBoost. This might suggest the averaging part of the document vectorization is too lossy for XGBoost. Besides, the speed of logistic regression is clearly a win. Although our solution did not beat the best model from Mass, et al. [8], our word2vec model with logistic regression classifier performed little

Model	Accuracy %
Mass Full Model	87.99
Mass Full Model + Bag of Words	88.89
Final Word2vec + Logistic Regression	88.04
Final Word2vec + XGBoost	87.58

Table 6: Test accuracy of final models. The models compared here all include additional unlabeled dataset to train word representations. Mass models are the benchmarking models from Mass, et al. [8].

bit better (88.04% vs 87.99%) than their full model without using bags of words, which is more equivalent to our solution. Overall, both results are comparable to Mass, et al's models that demonstrated our solution is capable of solving this sentiment analysis problem.

5 Conclusion

5.1 Free-Form Visualization

XGBoost is an algorithm based on gradient boosting which is a generalized version of AdaBoost. One characteristic AdaBoost has is its resistance to overfitting [16]. It would be interesting to observe this behavior on XGBoost with this sentiment analysis problem. To do so, we fixed all tuned parameters learned earlier and increased only the number of rounds (n_estimators) from 100 to 5000. Each model was validated to get both train and test accuracy. The resistance to overfitting were indeed observed as shown in Figure 3 where the training accuracy reached 100% around 1000 rounds while test accuracy kept increasing without the sign of overfitting and reached 87.95% after 3000 rounds. Such resistance to overfitting makes XGBoost an attractive algorithm in addition to its strengths.

5.2 Reflection

We presented a solution to perform sentiment analysis using word2vec and XGBoost. The data exploration was first performed to observe some patterns of the movie reviews from word distribution to the most common words. The solution comes with two main parts - unsupervised learning and supervised learning.

The unsupervised learning part is to apply word2vec to learn both underlying semantic and syntactic relationships from the movie reviews to build vector representations of words. The supervised learning part is to transform the review documents into vectors by averaging and feed them to XGBoost classifier (and logistic regression for benchmarking) for training and prediction.

The most difficult aspect of this project is to understand the maths behind the models in order to build the intuition for model tuning. Furthermore, there are two parts of the tuning. Tuning word2vec model has a direct impact on

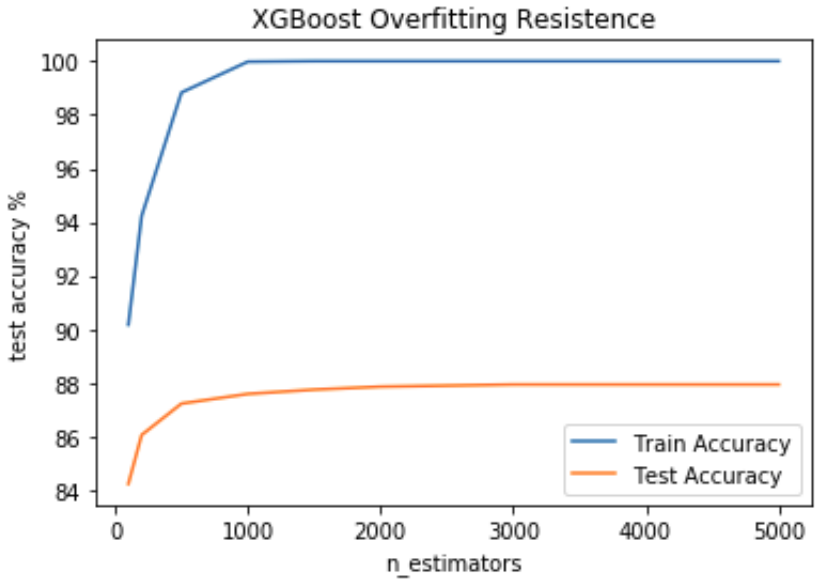

Figure 3: XGBoost showed the resistance to overfitting

the classifier accuracy. Second, the number of parameters available from both word2vec and XGBoost made it even more daunting. Due to time constraints, only a subset of the parameters were explored that seemed to make bigger contributions.

Although the best result was achieved by using logistic regression instead of XGBoost, it highlights the crucial role word2vec model plays in solving this problem. Its capability to learn the underlying semantic and syntactic relationship from a collection of texts is amazing. We proved again there is no one machine learning algorithm for every problem. Overall, this sentiment analysis problem can be solved with our solution.

5.3 Improvement

One promising way to beat our best model is to incorporate the bag of words. As demonstrated by Mass, et al. [8], that alone increased almost 1% in their test accuracy. The quality of data or feature engineering is not uncommon more important than picking the right machine learning algorithm.

A future direction is to compare more recent word embedding models like doc2vec [7] and ELMo [15]. Doc2vec is an adaptation of Word2Vec to learn richer representations of documents. ELMo is a LSTM-based model that learns high-quality deep context-dependent representations. By applying these state-of-the-art word embedding models, it is also possible to further increase the accuracy.

References

[1] Tianqi Chen. Introduction to boosted tree. https://homes.cs.washington.edu/ tqchen/pdf/BoostedTree.pdf.

[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[3] J. Friedman, T. Hastie, and R. Tibshirani. Additive Logistic Regression: a Statistical View of Boosting. *The Annals of Statistics*, 38(2), 2000.

[4] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In Sharad Mehrotra and Timos K. Sellis, editors, *SIGMOD Conference*, pages 58–66. ACM, 2001.

[5] Stanford AI Lab. Large movie review dataset. http://ai.stanford.edu/˜amaas/data/sentiment/, 2012.

[6] Patrick Lai. Extracting strong sentiment trends from twitter. http://nlp.stanford.edu/courses/cs224n/2011/reports/patlai.pdf, 2010.

[7] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, volume 14, pages 1188–1196, 2014.

[8] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *NIPS*, pages 3111–3119, 2013.

[11] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013.

[12] Anshul Mittal and Arpit Goel. Stock prediction using twitter sentiment analysis. http://cs229.stanford.edu/proj2011/GoelMittal-StockMarketPredictionUsingTwitterSentimentAnalysis.pdf, 2012.

[13] Mika Viking Mäntylä, Daniel Graziotin, and Miikka Kuutila. The evolution of sentiment analysis - a review of research topics, venues, and top cited papers. *CoRR*, abs/1612.01556, 2016.

[14] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 115–124, 2005.

[15] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[16] Robert E. Schapire. Explaining adaboost. In Bernhard Schölkopf, Zhiyuan Luo, and Vladimir Vovk, editors, *Empirical Inference*, pages 37–52. Springer, 2013.

[17] Stanford University. Cs 224d: Deep learning for nlp. https://cs224d.stanford.edu/lecture$_notes/notes1.pdf$.