Sentiment Analysis with Logistic Regression Variants and Ensembled Neural-Net-Based Feature Engineering

Daniel Amin
The Graduate Center, City University of New York
Advisor: Sos Agaian

ABSTRACT

Sentiment analysis is a popular Natural Language Processing problem which seeks to classify text into categories (sentiments in this case). We explore multiple approaches to improving classification performance by utilizing many machine learning algorithms such as Logistic Regression and its extensions, Dense and LSTM neural networks. We also explore multiple feature engineering approaches using Word2Vec, TFIDF, Dense and LSTM neural networks by ensembling them, combining as much feature information as possible. We find that feature engineering is a more important factor for improving performance than learning algorithms, with ensembled Word2Vec and TFIDF features working the best.

1 INTRODUCTION

Sentiment Analysis is a classic Natural Language Processing problem (text classification) where certain sentences are classified into groups of sentiments. This problem is important since it could be used to help many industries where reading/ understanding intent or mood of some sentences is needed. This project, for example, uses the Amazon Reviews dataset to classify whether customer reviews of certain products are positive or negative, which could be very useful for vendors. Apart from sentiment analysis, text classification has many applications such as chatbot or customer service bot, which require topic classification and/or intent classification.

Many machine learning algorithms have been used as approaches to solve this problem, along with some well-known feature engineering methods such as TFIDF [12] (term frequency inverse document frequency), Word2Vec [6], etc. This project explores further the possible improvement to the solutions by utilizing Logistic Regression variants with different solver methods and addition of penalty functions. Along with that, we build many neural network architectures from Dense networks to LSTM for classification modeling and also feature engineering. Our result shows that feature engineering might be a more effective factor in improving the accuracy of the classification, compared to modeling algorithms.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND ALGORITHMS

2.1 Logistic Regression

Logistic regression [7] is a machine learning algorithm that maps independent variables into binary outputs. The heart of this algorithm lies in the logistic function and logits. Logit is defined as follows:

$$l = \log_b \frac{p}{(1-p)} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ...$$

with β_i representing function parameters and x_i representing independent variables. Logistic function is not other than a sigmoid function that maps to 0 or 1 defined as:

$$\sigma(l) = \frac{e^l}{e^l + 1}$$

β_is are then learned through data iteratively by minimizing a cost function, which is a negative log-likelihood function of $\theta(\beta_i)$:

$$min(-L(\theta|y;x)) = min(-Pr(Y|X;\theta))$$

with y representing dependent variable. This is also known as cross-entropy loss function.

2.2 Dense Neural Net

Dense neural network [11] is a fully-connected network of neurons in which each neuron contains weights, bias parameters and activation function. These parameters are trained from data by using gradient descent and back propagation, which minimize a loss function, in the case of this project, cross-entropy loss. In other words, each neuron receives input of numbers from other neurons or input variables, then outputs a result of matrix multiplication and addition of parameters along with activation function. Gradient descent and back propagation [11] make sure that updated weights and bias parameters move to the right direction in terms of minimizing loss function. A fully connected neural network is similar to connected logistic regression units, however, with variable or no activation functions, not just sigmoid.

2.3 LSTM

LSTM (long short term memory) [10] is a form of recurrent neural network, which is a neural network that takes multiple timesteps in each neuron as part of minimization of loss function in each step. LSTM, however, has three gates that enable the ability of having long term memory and short term memory properties. As any RNN (recurrent neural network), the resulting state from previous timestep is used as input along with current input variables, however LSTM regulates the flow rate of these processes. The three gates called input, output and forget gates regulate: how much of saved state is used into the input, how much current state is used for saved state (forget), and proportion of output in terms

of combination of current state and saved state. These gates have parameters that are also trained with gradient descent and back propagation. This network architecture allows the model to learn sequences in variables better, highlighting important and forgetting less important sequences, while helping to reduce the vanishing gradient problem that is prevalent in vanilla RNNs.

2.4 TFIDF

TFIDF [12] is a feature engineering algorithm typically for text data, where it combines the bag of words concept (one hot encoding vector based on whether or not certain word is present in text, disregarding order) and scales those one-hot-encoded vectors by term frequency and document frequency. By definition, TF (term frequency) is defined by:

$$TF = \frac{\text{number of word w appearances in a document}}{\text{total number of words in document}}$$

and IDF (inverse document frequency):

$$IDF = \log_e \frac{\text{total number of documents}}{\text{number of documents with w in it}}$$

and so TFIDF scaling:

$$TFIDF = TF * IDF$$

2.5 Word2Vec

Word2Vec [6] is a neural net based word embedding algorithm that uses 2 layers of dense neural network architecture. This project focuses on one variant of Word2Vec which is the CBOW (continuous bag of words) model. This model is trained by predicting the current word given a window of neighboring context words using the dense network architecture. The last layer, however, is cutoff and therefore the result is a hidden layer that maps each word into a vector space. This model embed each word into a fixed dimension vector that points to a vector space where similar words reside based on the contextual statistical occurrences in the training corpus.

3 METHODOLOGY/DESIGN

3.1 Logistic Regression: Penalty Functions

In order to improve logistic regression result, we modify some logistic regression loss functions to include extra penalty terms. As discussed above, cross entropy loss function can be defined in detail as follows:

$$\min_{w,c} C \sum_{i=1}^{n} log(exp(-y_i(X_i^T w + c)) + 1)$$

Adding L2 penalty terms to the loss function results in:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} log(exp(-y_i(X_i^T w + c)) + 1)$$

Other than L2 penalty terms we also try adding L1 penalty terms which is defined as:

$$\min_{w,c} ||w||_1 + C \sum_{i=1}^{n} log(exp(-y_i(X_i^T w + c)) + 1)$$

As seen, L2 penalty terms quadratically increase loss as weights increase. With this put in place, we hope to reduce overfitting by keeping weight parameters small without making them completely zero. L1 on the other hand, has a tendency to prefer zeroing weight parameters that are not important to preserve small loss. To fine tune in between L2 and L1 penalty terms, we add Elastic-Net penalty terms which are defined as follow:

$$\min_{w,c} \frac{1-\rho}{2} w^T w + \rho||w||_1 + C \sum_{i=1}^{n} log(exp(-y_i(X_i^T w + c)) + 1)$$

This allows us to fine tune how much L2 and L1 regularization applied using the parameter ρ.

3.2 Logistic Regression: Optimizers

The typical logistic regression optimizer is a Newton method, also known as gradient descent [7], where minimizing loss function is done through first and second partial derivatives. LBFGS [5] is a high scale optimizer where Newton method is estimated, hence speeds up the calculations. Along with LBFGS, we use SAG [9] and SAGA [2] solvers which are gradient descent variants to support all three types of regularizations we use as discussed before.

3.3 Feature Engineering: Word2Vec + TFIDF

As discussed in section 2.5, Word2Vec embeds each word into a fixed dimension size vector, which we determine to be 100. This is a reasonable minimum size since the corpus of Amazon reviews is not very large, hence smaller dimensions help avoid overfitting the embedding. After training the Word2Vec model, we average all embedding vectors resulting from all words in each review to map each review into a vector space. The idea is that there is a separation in the vector space between negative reviews and positive reviews.

TFIDF is a scaled bag of words approach to text feature engineering. In other words, TFIDF captures the presence of certain words and their importance in each document, which is very important considering positive and negative sentiments have unique words whose presence could be a determining factor in classification.

In order to capture both contextual and keyword features, we combine 100 dimensions from Word2Vec with 1900 dimensions or more TFIDF features. In our observation, there is an impact in result as to how many dimensions of TFIDF we include. At a later stage, we then perform Singular Value Decomposition (dimension reduction algorithm) [4] across these features to combine them with other features.

3.4 Dense Neural Network Architecture

The architecture used for Dense neural network is shown in Figure 1. Batch normalization is added on each layer and dropout regularization at 40% rate is used as well to reduce overfitting. Adam optimizer works well in the training process. In later stage, we upscale this network from having 32, 64 neurons in each layer, to 64 and 128, 128 and 256 neurons each. The small scale network is used for classification modeling, since bigger networks tend to overfit. The upscaled network is used for feature engineering to increase dimensionality of the resulting features.

As a feature engineering tool, the last layer of the trained network is cutoff, which in turn outputs 128 or 256 dimensions of features which are then used as input to other machine learning algorithms. This process focuses on extracting high level features

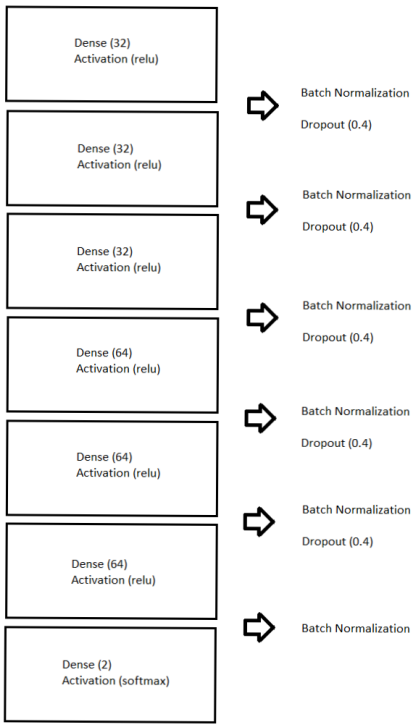

Figure 1: Fully connected network architecture

from the dense neural network which then are added with the base Word2Vec + TFIDF features in hope of catching more separation for classification.

3.5 LSTM Architecture

All feature engineering techniques above have been implementing bag of words model, in which word order does not matter at all. LSTM is used to enrich the features with sequence/order sensitive engineering. As a classification model, LSTM takes sequences of words as input. This is achieved through the Embedding layer that embeds each word into an integer/vector, which then appends these word vectors together forming compelete sentences in a document. Using Adam optimizer again, the network classifies negative and positive reviews based on these sequences of vectors. Since we have Word2Vec word embedding learned before, we use the embeddings as an initializer for the embedding layer, which we set as trainable (embeddings change overtime as optimizer minimize loss). This model encapsulates contextual features from Word2Vec embedding and sequential features from tokenized words. The architecture can be seen at Figure 2.

As a feature engineering tool, again, we remove the last Dense layer, which then results in the network outputting 64 dimension feature vectors to be used in conjunction with other features mentioned before. This encapsulates all three different approaches: Word2Vec for contextual features, TFIDF for presence of semantics features, and LSTM for sequential features.

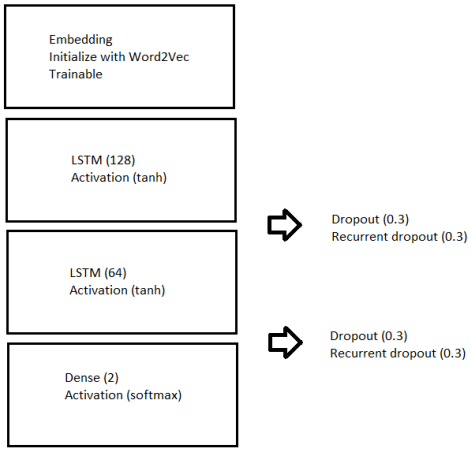

Figure 2: LSTM network architecture

4 EVALUATION

We sample 120,000 examples for training data and equally sized validation and test set at 40,000 samples consisting of text in each review along with a positive or negative label. The dataset has equal proportion of binary classes (half positive half negative) and so accuracy is an appropriate metric to evaluate our result.

4.1 Baseline

We set our baseline results to the classic algorithms such as Naive Bayes [8], default Logistic Regression, SVM [3], Random Forest [1], Dense Neural Network with only utilizing Word2Vec Features. These results can be seen at Table 1. As seen, the results between algorithms are quite similar except for Naive Bayes being significantly worse than others. In this baseline model, the deep neural network has the same architecture shown in Figure 1. We can conclude that the Word2Vec embeddings produce separation in the vector space between negative and positive reviews quite well.

Algorithm	*Naive Bayes*	*Logistic Regression*	*SVM*	*Random Forest*	*Deep Neural Net*
Accuracy	72.5%	85.0%	84.9%	83.2%	86.2%

Table 1: Baseline results

4.2 Logistic Regression Extensions

The results of Logistic Regression with different optimizers and loss functions can be seen in Table 2. These are obtained using Word2Vec features only. After trying many different hyperparameters, we conclude that no configuration significantly improve or worsen the accuracy. This shows us that complex loss functions including ElasticNet that combines both L1 and L2 regularizations do not factor in much into the performance of the model. With that in mind, we focus into better feature engineering, capturing more patterns in the data.

Loss function	L2	L2	ElasticNet	ElasticNet
L1 ratio (ρ)	-	-	0.6	0.4
C	1.0	0.5	1.618	0.618
Optimizer	LBFGS	SAG	SAGA	SAGA
Accuracy	84.93% (+/- 0.45%)	85.04%	85.26%	85.25%

Table 2: Logistic Regression Extension Results

4.3 Word2Vec + TFIDF

We capture single word and bigrams for TFIDF to ensure we also catch 2-word semantics. We use Logistic Regression with L2 penalty, C = 1.0, and LBFGS optimizer as modeling algorithm for these features. We believe there is an impact in how much TFIDF features we add in along with Word2Vec, to performance of the model, so we limit TFIDF features to some number of dimensions which correspond to the most occurring word or bigrams (most important). The results can be seen in Table 3. TFIDF alone gives a higher accuracy at 88% which shows that TFIDF carries information that Word2Vec lacks. When combining Word2Vec+TFIDF progressively, we can see the accuracy significantly improves as new information is added. However, we find that there is a limit to how many TFIDF features we can add before the model gets worse. In our experiment, we find that 29900 TFIDF dimensions yield the maximum accuracy. As shown, when using full TFIDF on top of Word2Vec features, the result gets worse.

Word2Vec Dimension	0	100	100	100	100	100	100	100	100	100
TFIDF Dimension	Full	1900	2900	4900	7400	9900	14900	19900	Full	29900
Accuracy	88.89%	87.92% +/-0.23	88.31% +/-0.31	88.73% +/-0.22	88.90% +/-0.26	89.00% +/-0.27	89.09% +/-0.20	89.12% +/-0.23	88.10% +/-0.29	89.15% +/-0.16

Table 3: TFIDF + Word2Vec Results

There is obviously an imbalance in the dimensionality between the two features. We suspect that 29900 most important TFIDF features contain most of the information, as seen, when the less important features get added up, the model disregards Word2Vec information by overfitting to the TFIDF features. We can also argue that Word2Vec features overlap much with TFIDF since TFIDF alone achieves 88%, compared to 85%, and those 100 Word2Vec dimensions only contribute to the extra 1% accuracy improvement.

4.4 Dense Neural Network

We use the optimum feature combination of TFIDF+Word2Vec at 30,000 total dimensions (100 Word2Vec + 29900 TFIDF) into the Dense network architecture. As seen in Table 4 , the accuracy is only a tad bit better than using Logistic Regression with the same features. Based on this observation, we can see that the machine learning algorithm matters less compared to the feature input.

As a feature engineering tool, after outputting the layer before classification, we input those vectors and combine them with original 30,000 Word2Vec+TFIDF features into a Logistic Regression and Random Forest model. The upscaled networks with 128 and 256 neuron outputs are combined with SVD'ed original 30,000 dimension vectors. The results are seen in Table 4.

We can conclude that these observations are not significantly different to each other since any combination of SVD dimensions and Neural Network output dimensions does not factor much into the performance of the model. We suspect that these are peak performances for these sets of features (Bag of Words), in other words, all information in the features are already extracted and there is nothing more these Dense net outputs could add to the model.

Features	30,000 W2V+TFIDF	200 SVD + 128 NN Output	500 SVD + 128 NN Output	200 SVD + 256 NN Output	25 SVD + 256 NN Output	50 SVD + 256 NN Output	100 SVD + 256 NN Output	200 SVD + 256 NN Output	500 SVD + 256 NN Output
Algorithm	Dense Network	Logistic Regression	Logistic Regression	Logistic Regression	Random Forest	Random Forest	Random Forest	Random Forest	Random Forest
Accuracy	89.65%	89.61%	89.51%	89.62%	89.91%	89.92%	89.91%	89.92%	89.92%

Table 4: Dense network Results

4.5 LSTM

As classification model, with embedded sequences features as input, LSTM performs really well, on par with Word2Vec+TFIDF combined. This brings high hopes that LSTM might add some value information to the previous bag of words models, which could increase the performance of the model. The results can be seen in Table 5.

Features	Embedded Sequences with W2V Embedding	200 SVD + RNN output	200 SVD + RNN output	100 SVD + RNN output	100 SVD + RNN output	256 NN output + RNN output	32 NN output + RNN output
Algorithm	LSTM	Logistic Regression	Random Forest	Logistic Regression	Random Forest	Logistic Regression	Logistic Regression
Accuracy	89.42%	89.27%	89.33%	89.12%	89.18%	89.17%	89.17%

Table 5: LSTM Results

LSTM brings new set of information to the feature sets since it focuses more on sequences, however, the results are rather disappointing. Combining 200 dimension SVD'ed Word2Vec+TFIDF with last layer LSTM output results in worse accuracy compared to LSTM with sequences features alone. With 200 SVD dimensions compared to 64 LSTM output dimensions, we suspect that these modeling algorithms favor SVD features more, hence, having results closer to Word2Vec+TFIDF results instead of getting better performance. However, after checking with smaller 100 SVD dimensions + 64 LSTM outputs, the results come worse.

We also try combining dense network output (as done in previous section) of 256 dimensions with 64 RNN outputs, with the results coming out worse as well. Even when downscaling the dense net to 32 outputs + 64 LSTM outputs, the results do not change at all. This brings us to the conclusion that the LSTM outputs prefer to have more Word2Vec+TFIDF features. As seen in

previous experiments, the highest accuracy is achieved based on TFIDF+Word2Vec features alone. LSTM itself, as a classifier, somewhat already implements the bag of words model along with sequences features by using its Embedding layer. Its embedding layer uses Word2Vec embedding on top of timesteps/sequences processing which could be why the LSTM model on its own performs on par with Word2Vec+TFIDF. Adding them together, however, only outbalance some feature groups or the others.

5 CONCLUSION

In this project, we explore many approaches to improving sentiment analysis classifier models. As seen in baseline models, TFIDF+Word2VEC models, the machine learning algorithms matter less compared to what features are going into the models. While Word2Vec and TFIDF work well combined, they both have strong correlation that they do not increase much information to each other. LSTM, while having a sequence attention features, already performs embedding on its own, hence adding its output to other bag of words features does not improve much.

REFERENCES

[1] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (01 Oct 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[2] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. 2014. SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. (2014). arXiv:cs.LG/1407.0202

[3] Theodoros Evgeniou and Massimiliano Pontil. 2001. Support Vector Machines: Theory and Applications, Vol. 2049. 249–257. https://doi.org/10.1007/3-540-44673-7_12

[4] Virginia C. Klema, Alan, and J. Laub. 1980. The singular value decomposition: its computation and some applications. *IEEE Trans. Autom. Control AC-25* (1980).

[5] Dong C. Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Mathematical Programming* 45, 1 (01 Aug 1989), 503–528. https://doi.org/10.1007/BF01589116

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. (2013). arXiv:cs.CL/1301.3781

[7] Joanne Peng, Kuk Lee, and Gary Ingersoll. 2002. An Introduction to Logistic Regression Analysis and Reporting. *Journal of Educational Research - J EDUC RES* 96 (09 2002), 3–14. https://doi.org/10.1080/00220670209598786

[8] Irina Rish. 2001. An Empirical Study of the Naïve Bayes Classifier. *IJCAI 2001 Work Empir Methods Artif Intell* 3 (01 2001).

[9] Mark Schmidt, Nicolas Le Roux, and Francis Bach. 2017. Minimizing Finite Sums with the Stochastic Average Gradient. *Mathematical Programming B* 162, 1-2 (2017), 83–112. https://doi.org/10.1007/s10107-016-1030-6 Revision from January 2015 submission. Major changes: updated literature follow and discussion of subsequent work, additional Lemma showing the validity of one of the formulas, somewhat simplified presentation of Lyapunov bound, included code needed for checking proofs rather than the polynomials generated by the code, added error regions to the numerical experiments.

[10] Ralf C. Staudemeyer and Eric Rothstein Morris. 2019. Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. (2019). arXiv:cs.NE/1909.09586

[11] Haohan Wang and Bhiksha Raj. 2017. On the Origin of Deep Learning. (2017). arXiv:cs.LG/1702.07800

[12] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. 2008. Interpreting TF-IDF Term Weights as Making Relevance Decisions. *ACM Trans. Inf. Syst.* 26, 3, Article 13 (June 2008), 37 pages. https://doi.org/10.1145/1361684.1361686