Multi-Class Text Sentiment Analysis

Jihun Hong
Stanford University
hjihun@stanford.edu

Alex Nam
Stanford University
hjnam@stanford.edu

Austin Cai
Stanford University
acai21@stanford.edu

December 13, 2019

1 Introduction

Text sentiment analysis is an important research topic for its wide applicability in real-world applications, and recent breakthroughs in text embedding and classification models led to state-of-the-art results. This project aims to apply recent innovations in machine learning to fine-grained multi-class sentiment analysis of Amazon reviews, contrasting different models including Naive Bayes Classifiers, Textblob/Vader, ML Classifiers, Neural Networks, and Text CNN to identify the most performant ML techniques. The objective of this project is to: predict one of the three (positive, neutral, negative) sentiment classes given an Amazon text book review.

2 Related Work

Most professional literature on sentiment analysis focused on individual models, with few contrasting an ensemble of models as we do in this paper. Projects that do contrast multiple models have primarily focused on a Yelp review dataset[9], which is limited in scope and diversity compared to the Amazon dataset[6]. A paper by Tan, Wang and Xu does use the Amazon dataset to train a sentiment classifier[1], but their best model (LSTM with GloVe) achieved only 70% accuracy. The authors' also simplified the problem into one of binary classification and failed to account for category imbalance. This project aims to extend the work of Tan et. al. by performing multiclass classification, addressing category imbalance, and applying new NLP techniques developed since Tan et. al.'s project.

3 Dataset and Features

We used an Amazon review dataset[6] which associates textual reviews with labels 1 through 5. We preprocessed our data by only considering book reviews, and defining a 3-class classification problem with label 1 as negative, 3 as neutral, and 5 as positive. In order to maintain consistency with our later approaches involving text-CNN, which requires a fixed example size, we only included reviews of maximum 50 words. We used 40,000 examples in our training set and 5,000 examples in our validation and test sets.

3.1 Managing Skew

Class imbalance was a major issue within our dataset, with 81.1% of the training data labeled "positive" and 2.7% labeled "negative." This led to many false positive predictions for underrepresented examples, with the f1 score of 0.367 and 0.547 for "negative" and "neutral" examples, respectively, from the best performing Naive Bayes baseline, compared with 0.913 for the dominant class. After experimenting with oversampling, undersampling, and SMOTE[8], we decided to use undersampling to enforce comparable class sizes within our training set while maintaining balanced class representation in our training data.

3.2 One-Hot Encoding

For our naive Bayes implementation, we created a dictionary of every word that appears at least 5 times from the entire training dataset (of 40,000 reviews), which generated a dictionary of size 20497. We then used a sparse vector to represent the number of occurrences of every word per text review.

3.3 Word Embeddings

To create vector representations for reviews, we used 300-dimension word2vec[11] embeddings pretrained on Google's news dataset and 25-dimension Global Vector for Word Representation (GloVe)[12] embeddings pretrained on a Twitter dataset. To represent each review as a vector, we converted each token to a 300-dimensional vector then computed the average of token vectors to obtain a vector per review text. Figure 6 contains the t-SNE visualization of original review data converted to vectors using the same method.

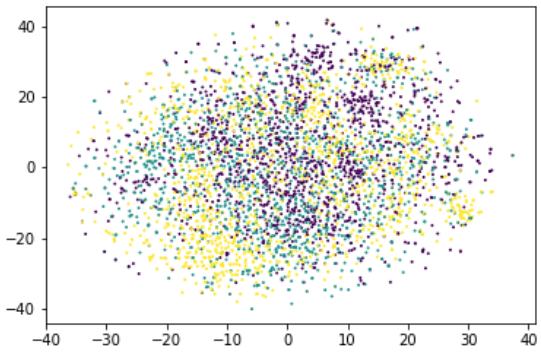

Figure 1: t-SNE visualization, colors represent different labels.

For our text CNN model, we stacked the token vectors within a sentence into a 50-by-300 matrix (where 300 is the dimensionality of word2vec embeddings), excluding reviews with more than 50 tokens. We also experimented with the learned embeddings by using the PyTorch Embedding layer, varying our embedding dimensions between 5 and 150.

4 Methods

We adopted multiple methods from pretrained NLP classifiers to text-based CNN to identify which best predicts the three class sentiment from a review text.

4.1 Baseline - TextBlob, Vader

To establish the baseline, we ran predictions on our testing set with pre-trained sentiment analysis tools available on Python: TextBlob[2] and Vader[3]. TextBlob outputs a score for 'polarity' and 'subjectivity' based on the words observed in the text input and Vader calculates a 'compound' scores based on the lexical features.

4.2 Baseline - Naive Bayes

Next, we implemented a Naive Bayes classifier with multinomial event model where each lowercased word in the training set is mapped to a distinct index in the dictionary, and the review is represented by a sparse vector of the number of occurrences for every word in the dictionary. In order to filter out specific product details (e.g. author names, brands), we used pyenchant[4] as a valid word check, excluded stopwords[5], and only included words that appear more than 5 times in the entire training set in our dictionary. For qualitative analysis, we examined the top twenty word indicators for each class.

4.3 ML Classifiers

We utilized a variety of ML Classifiers in the Scikit-Learn library within Python.

1. Support Vector Machine - discriminative classifier that finds a separating hyperplane between classes, optimized to maximize margin

$$\min_{w,b} \frac{1}{2}||w||^2$$
$$\text{s.t } y^{(i)}(w^T x^{(i)} + b) \geq 1$$

2. Decision Tree - continuously splits data according to certain parameters until examples are grouped by class

3. Random Forest - aggregates the results of a set of decision trees to predict classification

4. Multi-Layer Perceptron - deep neural network that is only feedforward

5. K-Neighbors Classifier - each example is assigned to the classification most common among its k nearest neighbors

6. Quadratic Discriminant Analysis - Generative algorithm that models each class as a normal distribution

$$p(x|y=i) = c \cdot \exp(-\frac{1}{2}(x-\mu_i)^T \Sigma^{-1}(x-\mu_i))$$
$$p(y) = \phi^y (1-\phi)^{1-y}$$
$$c = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}$$

7. Gaussian Naive Bayes - Naive Bayes algorithm applied on continuous data, assuming that the distribution of X given Y is Gaussian.

4.4 Linear Neural Net

We implemented several linear neural networks with a small number of hidden layers (1 3) on top of GloVe and Word2Vec embeddings. We experimented with different hyper parameters including the number and sizes of hidden layers, learning rates, activation (e.g. hardtanh, ReLU, Sigmoid) and optimizer types (e.g. SGD and Adam with learning rate .001) to find the best performing architecture.

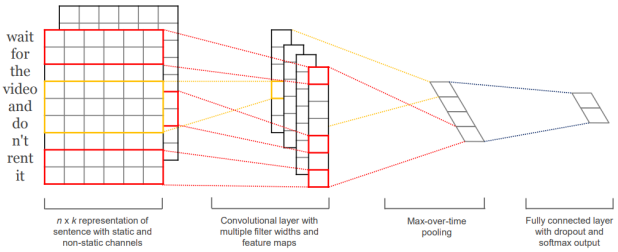

Figure 2: Text CNN model architecture

4.5 Text CNN

We implemented a Text CNN model based on Yoon Kim's architecture[10] (see Figure2), which has been shown to be effective in text sentiment analysis. The architecture includes a convolutional layer with varying number of channels, followed by ReLU and MaxPool activation layers; in the final layer, the model uses softmax function to output the probability predictions for each class. We adopted the open source Pytorch Text CNN implementation[13], on both learned embeddings of n-dimensions where the sparse vector representation of a review was mapped to n dimensions by the embedding layer trained concurrently with the convolutional layer, and static Word2Vec embeddings. In order to find the optimal hyper-parameters, we manipulated varied embedding schemes, filter sizes, and training epochs.

5 Results

5.1 Baseline - TextBlob, Vader, Naive Bayes

Table 1: Baseline Results.

Model	**Accuracy**
TextBlob	0.646
Vader	0.741
Naive Bayes (v.1)	0.838
Naive Bayes (v.2)	0.793
Naive Bayes (v.3)	0.764

Table 2: F1 Scores for v.1 Naive Bayes.

Negative	**Neutral**	**Positive**
0.367	0.547	0.913

The Naive Bayes classifier with the common words filter (v.1 in Table1) obtained 83.8% prediction accuracy on the testing set, while the same classifier without the filter (v.2) achieved 79.3% accuracy. The Naive Bayes classifier with the stop words filter on the undersampled training set (v.3) was least successful due to the misrepresentation of the prior distributions in the undersampled training set which did not match the actual skewed distribution in the testing set.

All three Naive Bayes classifiers outperformed the pretrained sentiment analysis tools. For qualitative analysis, we reviewed the top 20 indicator words for each class. The top 5 examples per class are listed as follows: Negative - ['waste', 'pointless', 'wasted', 'redeeming', 'puerile']; Neutral - ['slower', 'wordy', 'decent', 'somewhat', 'okay']; and Positive - ['awesome', 'beautifully', 'wonderful', 'loves', 'wait']. These indicators suggest that our Naive Bayes classifier is able to correctly identify sentiment associations for different words to output an overall score per review. However, even the best performing Naive Bayes classifier fell short in making predictions for under-represented classes (e.g. 'Negative' and 'Neutral') due to the significant influence of the prior distributions which favored the dominant class. The confusion matrix as well as the F1 scores[Table 2] clearly shows that the model outputs a significantly higher number of false positives (predicting a positive sentiment when the true label is negative or neutral) than false negatives to match the skewed prior distribution of the sentiment classes.

5.2 ML Classifiers

The best performing classifier was the SVM with rbf kernel trained on original distribution dataset, which achieved 83.8% test accuracy (Table3). This model performed significantly better than the same structured model trained on the undersampled set (accuracy 73.1%). Other methods, including SVM with Linear kernel, K-Neighbors, Quadratic Discriminant Analysis and MLP, also achieved comparable results, with all four classifiers achieving accuracy greater than 81%. As seen in Figure 1, the scattered nature of word representations make it difficult for traditional machine learning methods to obtain optimal accuracy on the classification task. However, the SVM did show significantly greater results on the test set than the baseline results. Although the SVM with rbf kernel improved the prediction accuracy marginally than the best performing Naive Bayes classifier, the SVM obtained F1 scores of 0.484, 0.321, and 0.911, for Negative, Neutral, and Positive, respectively, meaning the SVM is better at detecting Negative sentiments but shows a significant bias towards Positive class.

5.3 Linear Neural Net

The linear neural net with 3 hidden layers over Word2Vec (implemented with 3 ReLU activation, 0.1 dropout rate and final softmax layer using Adam optimizer with learning rate 0.001) obtained the highest testing accuracy of 69.2%. This significantly outperformed other models, including the same architecture over 100-dimensional GloVe embeddings (56.1%), over 10-dimensional GloVe (43.4%), and a single hidden layer model over Word2Vec (43.3%). We started with the simplest 1 hidden layer implementation as our baseline and gradually increased the complexity of the model to avoid overfitting. However, none of the linear neural net models obtained decent prediction accuracy over 70% without overfitting on the training set.

5.4 Text CNN

5.4.1 Learned Embeddings

We implemented a grid search of hyperparameters for Text CNN architecture based on learned embeddings varying embedding dimensions, learning rates, and the number of kernels. We started with the baseline architecture of 5 dimensional embedding for each word and three kernel CNN (which achieved 48.6% accuracy) and gradually increased the complexity of our model to avoid overfitting. Between 0.005 and 0.01 for learning rate, 0.005 consistenly performed better in different model architectures. The highest testing accuracy was achieved by CNN based on 55-dimensional learned embeddings with 5 convolutional kernels; 80.9% was the highest testing accuracy from 500 epochs. Despite decent performance, this model had its shortcomings as it initially had a high dev set variance and soon overfitted to the training data (see Figure3). Generally, regardless of the model complexity, Learned Embeddings-CNN showed high dev set variance compared to Word2Vec models (See Figure4 for another example of high dev set variance).

5.4.2 Word2Vec

The Text CNN model with word2vec embeddings performed the best among all other models, therefore achieving state of the art results on the sentiment classification task. The model achieved 91.46% accuracy on the test dataset, with high F1 scores across all three classes. The training loss and validation loss both steadily decreases over training, as seen in figure 3. Also, training accuracy and validation accuracy both increase over 150 epochs, showing no signs of model overfitting. Since the model is heavy and we had limited computing resources, we could only train the model for up to 150 epochs even though training the validation accuracy was marginally decreasing. The validation accuracy and loss seems to start at a better point that the training accuracy and loss, only because the model was outputting only positive labels as predictions. Text CNN with 10 convolutional kernels performed better than Text CNN with 5 convolutional kernels, which achieved a test accuracy of 87.1%.

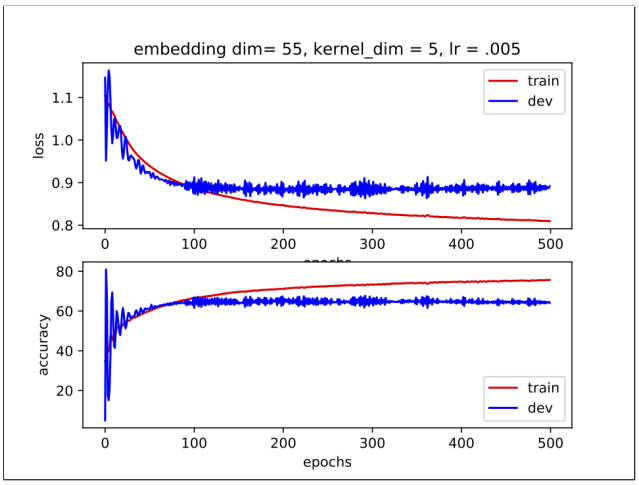

Figure 3: Training and Validation Loss and Accuracy

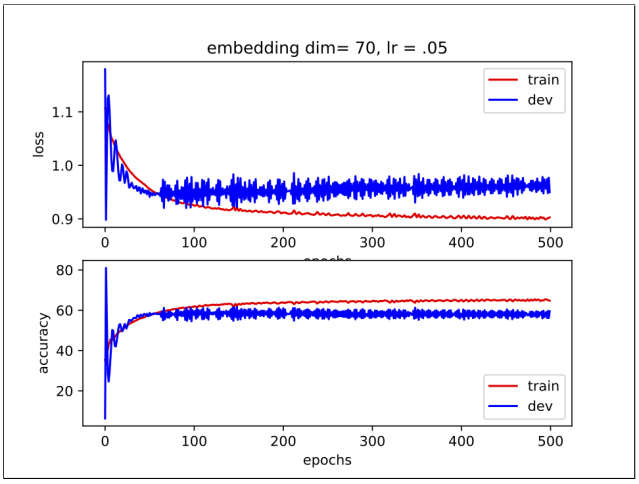

Figure 4: Training and Validation Loss and Accuracy

6 Discussion

In this project, we implemented an ensemble of ML techniques for fine-grained text sentiment classification and evaluated them against themselves and each other. We experimented with different sampling and embedding techniques, and utilized various visualization frameworks to inform our hyperparameter search. By achieving 91.46% classification accuracy with text CNN, we replicated the findings of Kim[10] while also

Table 3: Experiment Results (display highest accuracy from 500 epochs for NN/CNN).

Model	Accuracy
SVM (Rbf)	0.837
SVM (Linear)	0.833
QDA	0.816
KNN-5	0.816
MLP	0.811
Random Forest	0.795
Decision Tree	0.720
NN + 10-dim GloVe	0.434
NN + word2vec	0.692
TextCNN + 5-dim	0.486
TextCNN + 20-dim	0.576
TextCNN + 45-dim	0.742
TextCNN + 55-dim	0.809
TextCNN + 100-dim	0.748
TextCNN + 150-dim	0.675
TextCNN + word2vec	0.914

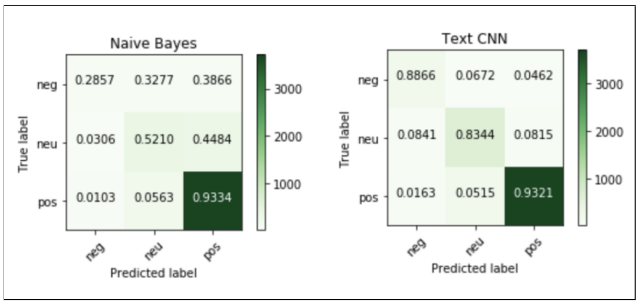

Figure 5: Confusion Matrix (normalized)

Table 4: F1 Scores for Text CNN

Negative	Neutral	Positive
0.727	0.788	0.955

showing Text CNN can work on paragraphs of input text of an arbitrary length.

Interesting extensions include generalizing our findings by classifying review text to all five review categories. Since the model was only trained and tested on the Amazon Book Reviews dataset, we can extend the sentiment classification model to text data from other domains. Other future projects might support CNN input text of arbitrary length and experiment with Transformer based models such as BERT or XL-NET.

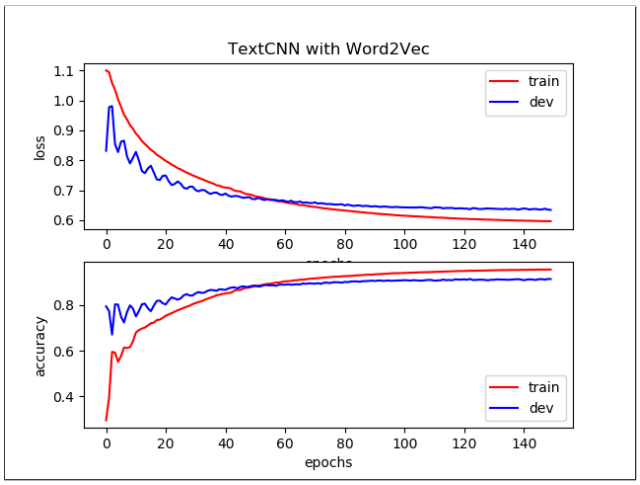

Figure 6: Training and Validation Loss and Accuracy

7 Contributions

Alex tested on TextBlob (as baseline), implemented naive Bayes model (as baseline), the linear NN model over Glove and Word2Vec embeddings, one hot encoding CNN model over raw text input, and tuned the CNN model on learned embeddings. Austin implemented under-, over-, and SMOTE sampling techniques, hyperparameter grid search, and contributed to the CNN model. Jihun tested on Vader (as baseline), implemented ML classifiers, generated GloVe and Word2Vec embeddings, implemented Word2Vec-input CNN model, and tuned the CNN model on Word2Vec embeddings.

References

[1] Tan, W., Wang, X. and Xu, X. (2018). Sentiment Analysis for Amazon Reviews. *CS 229, Stanford University*

[2] TextBlob Python Library: https://textblob.readthedocs.io/en/dev/

[3] VADER (Valence Aware Dictionary and sEntiment Reasoner) Analysis https://pypi.org/project/vaderSentiment/

[4] pyenchant 2.0.0 https://pypi.org/project/pyenchant/

[5] nltk 3.4.5 stopwords https://pypi.org/project/nltk/

[6] J. McAuley, C. Targett, J. Shi, A. van den Hengel (2015). Image-based recommendations on styles and substitutes. *SIGIR*

[7] Devlin, J., Change, M., Lee, K., Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

[8] Chawla, N., Bowyer, K., Hall, L., Kegelmeyer., W. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*

[9] Yelp. (2019). Yelp Dataset. *Kaggle* https://www.kaggle.com/yelp-dataset/yelp-dataset

[10] K., Yoon. (2014). Convolutional Neural Networks for Sentence Classification *EMNLP*

[11] T., Mikolov, H., Sutskever, K., Chen, G., Corrado, J., Dean. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781

[12] J., Pennington, R., Socher, C., Manning. (2014). GloVe: Global Vectors for Word Representation.

[13] Open source CNN for Sentence Classification Code in Pytorch https://github.com/Shawn1993/cnn-text-classification-pytorch

[14] Project Github Link: https://github.com/jihunhong/amazon-review.git