Comparison of Sentiment Analysis using Traditional Machine Learning Methods and Recurrent Neural Networks

Nate C.-M. Bartlett

1 Domain Background

Sentiment analysis is the extraction and classification of opinion or emotion from text, and is a rapidly growing area in artificial intelligence. A recent review [1] traced the earliest applications of sentiment analysis to research on public opinion analysis in the early 1900s [2], but it wasn't until roughly a century later that sentiment analysis using natural language processing (NLP) and machine learning techniques really began to take off, owing to the exponentially growing amount of online text. To illustrate just how much activity has been generated in this field of research, the same review found that 99% of papers on the topic have been published after 2004. Some of the most relevant modern applications of sentiment analysis fall under marketing and online advertising [3, 4], and it isn't difficult to imagine why research in this area has been so hot. However, there are other applications in which reliably and accurately extracting opinions from text is of huge importance, e.g. determining public opinions regarding political or social issues by mining text obtained on social media [5, 6].

In 2002, Pang et al. [7] demonstrated that three machine learning techniques, Naive Bayes (NB), maximum entropy classification, and support vector machines (SVMs), achieved better than human-level performance on a binary sentiment analysis task applied to movie reviews, with the SVM model achieving around 83% accuracy. In the decade and a half since then, there have been remarkable advancements in machine learning that are particularly well suited to handling sequential (e.g., text-based) data and NLP tasks, in particular word embeddings, such as word2vec [8] and GloVe [9], and recurrent neural networks (RNNs) [10, 11]. Moreover, the number and size of publicly available datasets has grown as well, and is worth noting that the Pang et al. paper used a dataset of 1400 instances (700 positive and 700 negative), which by today's standards is quite small. By taking advantage of these improved methods for handling text-based data, as well as larger publicly available datasets, the aim of the present study is to investigate how well these more modern techniques perform on the sentiment analysis task, and compare their performance to that of the more traditional methods.

2 Problem Statement

Sentiment analysis is a classification task, and as such can be defined generically as the goal of obtaining a mapping $f : X \to Y$, where X is the set of text to be analyzed (sometimes called a corpus) and $Y = \{y_1, y_2, ..., y_k\}$ is the set of classes that each $x \in X$ can belong to. In less abstract terms, some examples of possible sets X include {movie reviews}, {hotel reviews}, {customer feedback} , and {tweets}, and some corresponding examples of possible sets Y include $\{+, -\}$, $\{\star, \star\star, \star\star\star, \star\star\star\star\}$, {☺, 😐, ☹ }, and {Democrat, Republican}. Thus, we would like to construct a model that can predict such class labels from unlabeled text data.

3 Dataset and Inputs

The primary dataset that will be used in the proposed study is the Large Movie Review Dataset (LMRD) [12], also known as the Internet Movie Database (IMDB) review dataset. This dataset was constructed to serve as a benchmark for binary sentiment classification and contains 50,000 IMDB movie reviews labeled as positive or negative. The dataset is split into 25,000 training and 25,000 test examples, with 50% of the examples belonging to the positive class and 50% belonging to the negative class. Other important characteristics of the dataset are as follows:

- The training and test sets are disjoint, meaning that no reviews associated with a single film exist in both the training set and the test set. This is important because it prevents inflated performance on the test set due to memorization of terms associated with a particular film.
- The number of reviews for a given film is capped at 30 to reduce correlation (reviews associated with a given film have been found to be correlated).
- A positive review corresponds to an IMDB score of ≥ 7, while a negative review corresponds to an IMDB score of ≤ 4. Note that IMDB scores are on a scale of 1–10. Since no reviews with scores of 5 and 6 are included in the dataset, neutral reviews are omitted, and the dataset can be considered as highly polar. Thus, it is well-suited to the study of binary sentiment classification.

LMRD is available online as both raw text and in a bag-of-words (BoW) representation, and the latter is also included in the Keras framework as a built-in dataset. In the BoW representation, each input x is represented as an n-dimensional vector indexed by the n unique words in X, where the value of the ith vector component is the number of times that word i occurs in x. Thus, the bag-of-words representation does not encode anything except word count (or sometimes just word presence), and in particular does not encode word order. For the sake of convenience, the Keras version of LMRD will be utilized in the proposed study since it includes a somewhat more extensive set of preprocessing functions, such as the ability to skip the top n frequently occurring words, which will typically be stop words, and to set the maximum sequence length. The words in the corpus are indexed by frequency, where an index of 1 represents the most frequent word in the corpus, and index of 2 represents the second most frequent word, and so on. The index 0 is used to encode <UNK>, or an unknown word, i.e. a word that does not appear in the corpus. This dataset can be used essentially as is to train a NB classifier (and others) in Scikit-learn.

For use with RNNs, a slightly different representation will be required because of the sequential nature of the neural network architecture. Here, each word in a review will be represented as a one-hot vector and will either be fed into the model as such, or converted to a word embedding vector using a pre-trained embedding matrix such as GloVe or word2vec, which are widely available online. Unlike BoW representations, word embedding vectors can encode complex relationships such as analogy relationships between words. To allow for parallelization, all input sequences will need to be the same length, and since reviews will in general differ in length, the longest review can serve as the maximum length, with other reviews padded to the maximum length.

Time permitting, it would be interesting to look at multiclass sentiment analysis, and the SAR14 dataset [13] is a good candidate dataset to use for this task. SAR14 is very similar to the LMRD, so the details will not be repeated here. The key difference between SAR14 and LMRD is that the former contains 233,600 IMDB movie reviews and their raw ratings of 1–4 (66,222 reviews) and 7–10 (167,378 reviews). Thus, SAR14 is a class-imbalanced dataset. Note that, like LMRD, neutral reviews (ratings of 5 and 6) are not included in SAR14.

4 Solution Statement

A number of machine learning classifiers, such as NB, SVM, logistic regression, and neural networks (particularly, RNNs) can be used to solve the sentiment analysis problem as a standard supervised learning task. In the proposed study, the classification accuracies and computation times of a NB classifier and one or two (time permitting) other traditional classifiers will be compared against those of an RNN-based model. The NB classifier will be implemented in Scikit-learn using unigrams and both word count and presence, and the RNN will be implemented in Keras. At a minimum, a unidirectional RNN with long-short term memory (LSTM) units will be implemented, since these are known to perform well with long sequences of text [14]. However, time permitting, it would be interesting to also attempt to solve the sentiment analysis task using bidirectional RNNs, RNNs with gated recurrent units (GRUs), and vanilla RNNs.

5 Benchmark Model

The benchmark model that will be used to establish baseline performance will be the NB classifier. Since it is easy to implement as well as interpret and fast to train, it is widely used as a baseline model in text classification problems. The NB classifier calculates the probabilities that document $x \in X$ has label $y \in Y$, and returns the highest-probability label as the predicted class $\hat{y}$, i.e.,

$$\hat{y} = \text{argmax}_{y \in Y} P(y|x). \tag{1}$$

The probabilities of the labels are calculated using Bayes' rule: [15]

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}. \tag{2}$$

$P(y|x)$ can be read as, "the probability of class y given document x", $P(x|y)$ can be read as "the probability of document x given class y", and $P(x)$ and $P(y)$ are the respective probabilities of document x and class y. However, note that because the purpose of the classifier is to return the argmax of $P(y|x)$ over $y \in Y$, it is unnecessary to include $P(x)$ in the denominator because it is a constant independent of y. Thus, the same y that maximizes $\frac{P(x|y)P(y)}{P(x)}$ will maximize $P(x|y)P(y)$, and $\hat{y}$ can be rewritten as

$$\hat{y} = \text{argmax}_{y \in Y} P(x|y)P(y). \tag{3}$$

The $P(y)$ can be calculated easily—it is the fraction of documents in X with label y. Calculation of $P(x|y)$ is done by assuming that the occurrences of the words in x are all independent events, and this is called the naive Bayes assumption. In other words, for words $w_1, ..., w_n$ in x:

$$P(w_1, w_2, ..., w_n|y) = \prod_i P(w_i|w_1, w_2, ..., w_{i-1}, y) = \prod_i P(w_i|y), \tag{4}$$

where the first equality follows from the chain rule and the second equality follows from the naive Bayes assumption. Thus, $\hat{y}$ can be finally expressed as:

$$\hat{y} = \text{argmax}_{y \in Y} P(y) \prod_i P(w_i|y). \tag{5}$$

Each $P(w_i|y)$ is just the fraction of words in all documents with label y that are word w_i. In the proposed study, Laplace smoothing (the default smoothing method in Scikit-learn) will be used to avoid the problem of words appearing in documents of some classes but not others. Without Laplace smoothing, this situation inevitably leads to $\hat{P}(y|x) = 0$ because the product term in Eq. (5) will contain some $P(w_i|y) = 0$. The performance of a NB classifier can be evaluated using the metrics described in the next section.

6 Evaluation Metrics

The primary evaluation metric that will be used to quantify the performance of both the benchmark model and the solution model will be accuracy. It is important to note that accuracy is often avoided as the evaluation metric in favor of precision and/or recall in classification problems because it does not work well in cases of class imbalance. However, as described in the Dataset and Inputs section, for LMRD, the classes are completely balanced. For completeness, precision and recall will be evaluated as well (and this will be useful/necessary if the SAR14 dataset is used). Below are the definitions of accuracy, precision, and recall:

$$accuracy = \frac{true\,positives + true\,negatives}{true\,positives + false\,positives + true\,negatives + false\,negatives} \tag{6}$$

$$precision = \frac{true\,positives}{true\,positives + false\,positives} \tag{7}$$

$$recall = \frac{true\,positives}{true\,positives + false\,negatives} \tag{8}$$

From the equations, we can see that accuracy expresses the fraction of correctly classified documents, precision expresses the fraction of documents correctly classified to be in the positive class, and recall expresses the fraction of positive-class documents that were correctly classified. A confusion matrix—a table in which the columns are the actual classes, the rows are the predicted classes, and the ij^{th} entry of the table is the number of examples in the dataset with predicted class i and actual class j—is useful for visualizing where the model has performed well and where it has made mistakes, particularly when there are more than two classes.

7 Project Design

The workflow of the proposed study will be as follows:

1. Establish baseline accuracy using a NB model in Scikit-learn and the Keras LMRD dataset. Scikit-learn has three variants of NB, but the two that are relevant here are the MultinomialNB and BernoulliNB classes. The former is for use with occurence count vectors and the latter is for use with binary (word present/word not present) vectors. A rough outline of what this will look like in Python is:

```
from keras.datasets import imdb
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.metrics import accuracy_score

(X_train, y_train), (X_test, y_test) = imdb.load_data(args)

mnb_clf = MultinomialNB(args)
mnb_clf.fit(X_train, y_train)
predictions = mnb_clf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)

bnb_clf = BernoulliNB(args)
bnb_clf.fit(np.where(X_train > 0, 1, 0), y_train)  # binarize vector
predictions = bnb_clf.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
```

2. Evaluate the accuracy of one other traditional model, such as a SVM model, also using Scikit-learn. In addition to utilizing the occurence count or binary vectors used for NB, it might be interesting to try out a pre-trained word embedding matrix to generate the input feature vectors. Optimizing the SVM will certainly require a bit of grid search, which can be implemented using Scikit-learn's GridSearchCV class. Again, a rough outline in Python would look like:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

params = {grid search parameters}
svm_clf = SVC(args)
grid = GridSearchCV(svm_clf, params).fit(X_train, y_train)
predictions = grid.predict(X_test)
accuracy = accuracy_score(y_test, predictions)

# repeat with binary count feature vectors and word embedding vectors.
```

3. Build the RNN model in Keras and optimize it by tuning the network architecture and hyperparameters (layer sizes, number of layers, unidirectional/bidirectional, word embedding size, activations, regularization, etc.). As an example, a very basic RNN model can be trained and evaluated in Keras with the code:

```
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense

model = Sequential()
model.add(Embedding(args))  # word embedding matrix goes here
model.add(LSTM(args))
model.add(Dense(args))
model.compile(args)

model.fit(X_train, y_train, args)
predictions = model.predict(X_test)
accuracy = accuracy_score(y_test, predictions)
```

Note that, as described in the Dataset and Inputs section, the dataset needs to be represented differently with RNNs, namely as a sequence of one-hot vectors. The LMRD feature vectors can be converted to one-hot vectors in Keras using the keras.utils.to_categorical function, and the sequences can be padded to the same length using the keras.preprocessing.sequence.pad_sequences function.

4. Once the models are optimized, compare their performances and try to understand where each of the models performed well and did not perform well by manually looking at some examples of correctly classified and misclassified reviews.

5. Time permitting, repeat the above steps for the SAR14 dataset. SAR14 does not come preprocessed, but can be processed without too much effort using the CountVectorizer class in Scikit-learn (or similar in NLTK). It would also be interesting to train a word embedding matrix from scratch.

References

[1] Mika V. Mantyla, Daniel Graziotin, Miikka Kuutila. *The Evolution of Sentiment Analysis - A Review of Research Topics, Venues, and Top Cited Papers.* Computer Science Review, **27**, 16–32, (2018).

[2] D.D. Droba. *Methods Used for Measuring Public Opinion.* American Journal of Sociology, **37**(3), 410, (1937).

[3] *Online Sentiment Analysis in Marketing Research: A Review.* Journal of Research in Interactive Marketing, **12**(2), 146, (2018).

[4] A.-M. Popescu, O. Etzioni. *Extracting Product Features and Opinions from Reviews* in Natural Language Processing and Text Mining. Springer, 9–28, (2007).

[5] A. Pak, P. Paroubek. *Twitter as a Corpus for Sentiment Analysis and Opinion Mining.* LREc, **10**, (2010).

[6] A. Tumasjan, T.O. Sprenger, P.G Sandner, I.M. Welpe. *Predicting Elections with Twitter: What 140 Characters Reveal about Political Statement.* Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media, (2010).

[7] B. Pang, L. Lee, S. Vaithyanathan. *Thumbs up? Sentiment Classification using Machine Learning Techniques.* Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 79-86, (2002).

[8] T. Mikolov, K. Chen, G. Corrado, J. Dean. *Efficient Estimation of Word Representations in Vector Space.* arXiv:1301.3781 [cs.CL]

[9] J. Pennington, R. Socher, C.D. Manning. *GloVe: Global Vectors for Word Representation.* `https://nlp.stanford.edu/pubs/glove.pdf`

[10] Z.C. Lipton, J. Berkowitz. *A Critical Review of Recurrent Neural Networks for Sequence Learning.* arXiv:1506.00019 [cs.LG]

[11] T. Young, D. Hazarika, S. Poria, E. Cambria. *Recent Trends in Deep Learning Based Natural Language Processing.* arXiv:1708.02709v6 [cs.CL]

[12] `http://ai.stanford.edu/~amaas/data/sentiment/`

[13] `http://tabilab.cmpe.boun.edu.tr/datasets/`

[14] S. Hochreiter, J. Schmidhuber. *Long short-term memory.* Neural Computation, **9**(8), 1735, (1997).

[15] D. Jurafsky, J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Chapter 4, 3rd Draft Edition, (2018). `https://web.stanford.edu/~jurafsky/slp3/`