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

Nate C.-M. Bartlett

I. DEFINITION

A. Project Overview

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 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 advances 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 important to note 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 RNNs perform on the sentiment analysis task, and compare their performance to that of NB and SVMs. The Large Movie Review Dataset (LMRD) [12], also known as the Internet Movie Database (IMDB) review dataset, which was constructed to serve as a benchmark for binary sentiment classification, will be used to evaluate model performance.

B. 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 labels among which each $x \in X$ is associated with. 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. Three such models will be described in Section IIB.

C. Metrics

The primary evaluation metrics that will be used to quantify the performance of both the NB and SVM models as well as the RNN models will be accuracy, precision, and recall. 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 can be misleading in cases of severe class imbalance. For LMRD, the classes are balanced, so accuracy is a fair metric; however, all three metrics will be evaluated for completeness. Below are the definitions of accuracy, precision, and recall in the case of binary classification:

$$accuracy = \frac{tp + tn}{tp + tp + tn + fn} \tag{1}$$

$$precision = \frac{tp}{tp + fp} \tag{2}$$

$$recall = \frac{tp}{tp + fn} \tag{3}$$

where tp, tn, fp, and fn denote the number of true positives, true negatives, false positives, and false negatives, respectively.

From the above equations, we can see that accuracy expresses the fraction of correctly classified reviews, precision expresses the fraction of reviews classified as positive that were correctly classified, and recall expresses the fraction of positive reviews that were correctly classified. This information is often summarized in a confusion matrix

	Predicted +	Predicted -
True +	tp	fn
True -	fp	tn

II. ANALYSIS

A. Datasets

The LMRD dataset contains 50,000 IMDB movie reviews labeled as positive or negative. It 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. 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 highly polar. Thus, it is well-suited to the study of binary sentiment classification.

One important consideration to note is that the lengths of movie reviews vary widely in general. Figure 1 shows the distributions of review length for LMRD. The minimum review length is 7 words and the maximum is 2492 words, while the mean and standard deviation are 234 and 173 words, respectively. Given that all review lengths greater than roughly 500 words can be considered outliers, it seems reasonable to truncate reviews to 500 words.

B. Algorithms and Techniques

1. Naive Bayes

The first model that will be used to establish baseline performance will be the NB classifier. Since it is easy to implement and interpret, as well as 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.,

$$\hat{y} = \text{argmax}_{y \in Y} P(y|x). \tag{4}$$

The probability of document x having label y is calculated using Bayes' rule: [13]

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{5}$$

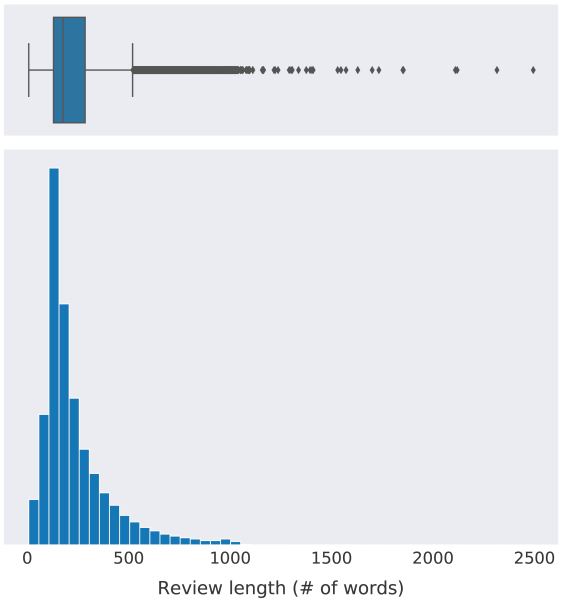

FIG. 1. Distribution of review lengths

where $P(y|x)$ is the conditional probability of observing class y given document x, $P(x|y)$ is the conditional probability of observing 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, since 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

$$\hat{y} = \text{argmax}_{y \in Y} P(x|y)P(y). \tag{6}$$

The $P(y)$ can be calculated easily—they are the fraction of $x \in X$ with label y. Calculation of the $P(x|y)$ 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:

$$\begin{aligned} P(x|y) &= P(w_1, w_2, ..., w_n|y) \\ &= \prod_i P(w_i|w_1, w_2, ..., w_{i-1}, y) \\ &= \prod_i P(w_i|y), \end{aligned} \tag{7}$$

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

$$\hat{y} = \text{argmax}_{y \in Y} P(y) \prod_i P(w_i|y). \tag{8}$$

Each $P(w_i|y)$ is just the frequency of word w_i in all documents with label y. In the proposed study, Laplace smoothing 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. (8) will contain some $P(w_i|y) = 0$.

To train an NB classifier on text data, the set of documents must be encoded in the appropriate representation. In the case of LMRD, each movie review is represented as an n-dimensional vector, where n is the number of words in the corpus vocabulary. The ith component of the vector contains the number of occurrences of the ith word in the corpus vocabulary in that movie review. This is called the bag-of-words (BoW) representation, and is discussed in more detail in Section IIIA. A slight variant on the NB classifier, the Bernoulli NB classifier, instead takes as input binary vectors of word absence/presence (as opposed to integer vectors of word counts), and such a classifier will be evaluated in the present study as well.

2. *Support Vector Machine*

The second traditional machine learning model that will be implemented is a linear SVM classifier. SVMs can be applied to a wide variety of machine learning tasks, including linear and nonlinear classification and regression, and have been used extensively for sentiment analysis [14–16], where they can provide better results than NB classifiers [7].

Support vector classifiers are margin classifiers, which means that they aim to find a decision boundary that results in the largest distance (margin) to the nearest training instance of a given class. In general, since such a margin only exists if the data is linearly separable, what is more often implemented is known as soft-margin classification, which aims to find as large a margin as possible while minimizing the number of points that fall within the margin, sometimes referred to as margin violations.

The mathematics behind support vector classifiers is relatively straightforward. The decision boundary is expressed as [17]

$$\mathbf{w}^T \cdot \mathbf{x} + b = 0, \tag{9}$$

where $\mathbf{w}$ is the vector of trainable weights, $\mathbf{x}$ is the feature vector, and b is the bias term. In the case of binary classification, classes are predicted for instances $\mathbf{x}_i$ as follows:

$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{w}^T \cdot \mathbf{x}_i + b < 0, \\ 1 & \text{if } \mathbf{w}^T \cdot \mathbf{x}_i + b > 0 \end{cases}$$

Finally, for soft-margin classification, $\mathbf{w}$ is determined by minimizing the following cost function,

$$\frac{1}{2}||\mathbf{w}||^2 + C\left[\frac{1}{n}\sum_i^n \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i - b))\right], \tag{10}$$

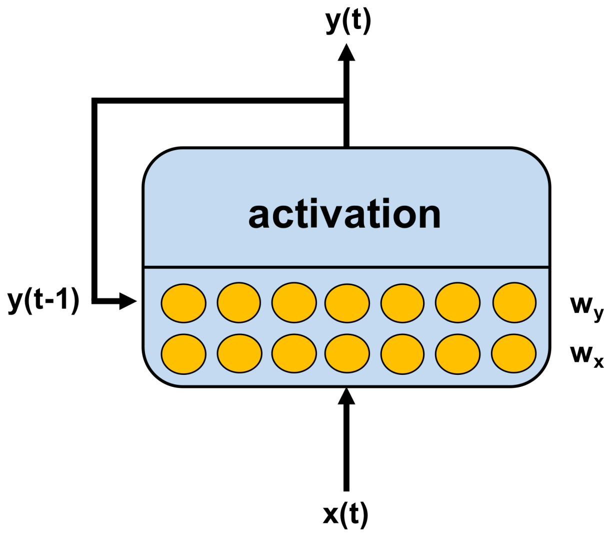

FIG. 2. Schematic illustration of the basic RNN structure. At time step t, the network receives input $\mathbf{x}(t)$ and the output from the previous time step $\mathbf{y}(t-1)$, which are multiplied by weights $\mathbf{w}_x$ and $\mathbf{w}_y$, respectively. The sum of these two products is added to the bias and passed through an activation function to give the output $\mathbf{y}(t)$.

where C is a hyperparameter that determines how margin violations and classification error are traded off. When C is small, more emphasis is placed on finding a large margin (at the expense of some misclassifications), and when C is large, more emphasis is placed on correctly classifying instances (at the expense of a smaller margin).

Conveniently, the BoW representation of the input data used to train an NB classifier can also be used to train an SVM.

3. *Recurrent Neural Networks*

The performance of the NB and SVM classifiers described in the previous two sections will be compared against that of a RNN classifier. Recurrent neural networks differ from traditional neural networks in that they have a loop structure, as shown in Fig. 2, which makes them ideal for working with sequential data such as time-series data and text. At any given time step, the RNN receives as input the data as well as the output from the previous time step. This means that the output at time t depends on the inputs/outputs from all previous time steps leading up to it, allowing RNNs to retain something of a “memory” of previous information.

The output $\mathbf{y}(t)$ of a general RNN is given by the following: [17]

$$\mathbf{y}(t) = g(\mathbf{w}_x^T \cdot \mathbf{x}(t) + \mathbf{w}_y^T \cdot \mathbf{y}(t-1) + b), \tag{11}$$

where g is an activation function, $\mathbf{x}$ is an instance, $\mathbf{w}_x$ and $\mathbf{w}_y$ are weight matrices, and b is the bias term; the tanh activation function is often used with RNNs. Note

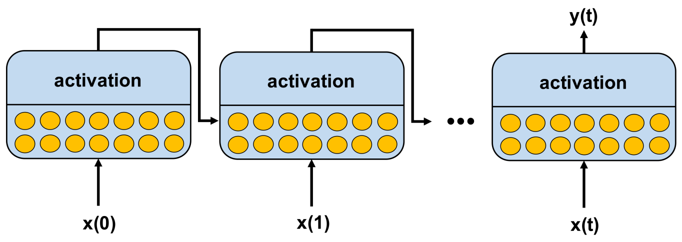

FIG. 3. Schematic illustration of the many-to-one RNN architecture.

that the output at time step t, $\mathbf{y}(t)$, depends on both the input $\mathbf{x}(t)$ and the output from the previous time step $\mathbf{y}(t-1)$. The weights are typically learned using backpropagation (called backpropagation through time in the context of RNNs) and gradient descent.

When working with text data, the simplest option is to encode the input $\mathbf{x}$ as a sequence of one-hot vectors, whose length is the vocabulary size and whose "hot" entry is the index of the word in the vocabulary being encoded. Note that because one-hot vectors are orthogonal, the cosine similarity of any two different words is zero. Therefore, one-hot vectors are unable to encode any relationships between words. An alternative to this encoding is word embeddings, which in contrast to one-hot vectors are dense vectors, typically of dimension up to a few hundred. They encode relationships between words, such as analogy relationships, and thus can potentially provide superior results in NLP tasks. Word embeddings can either be learned during RNN training (the first layer can function as an "embedding" layer) or loaded into the network pre-trained; both approaches will be evaluated in this study.

There are several basic RNN constructions depending on the application, but in classification tasks the so-called many-to-one or sequence-to-vector RNN is used, where the input is a sequence of vectors and the output is a single vector or scalar. This is illustrated in Fig. 3. Here, with a softmax activation on the final output layer, the model outputs the probability that the input instance belongs to the positive or negative class.

Two of the major shortcomings of RNNs are that with long sequences, they are highly susceptible to vanishing and exploding gradients, and they eventually lose their memory of earlier inputs. Both of these issues are often dealt with by using LSTM cells. This approach is taken here because the 500-word sequences used in this study are considered long, and although LSTM-based RNNs are considerably more conceptually challenging than basic RNNs, they are just as easy to implement in machine learning frameworks such as Tensorflow and Keras.

TABLE I. Benchmark results for LMRD

Model	**Accuracy**	**Recall**	**Precision**
Naive Bayes	0.819	0.772	0.852
Bernoulli Naive Bayes	0.838	0.813	0.856
SVM	0.882	0.885	0.881

C. Benchmark

The performances of three benchmark models, NB, Bernoulli NB, and a linear SVM classifier were evaluated on the LMRD dataset, and the results are summarized in Table I and in Fig. 4. Implementation details are provided in Section IIIB. Interestingly, the NB and Bernoulli NB classifiers return significantly more false negatives than false positives, as reflected in their recall scores, whereas the SVM classfier returns around an equal number of both types of misclassifications.

III. METHODOLOGY

A. Data Preprocessing

LMRD is included in the `Keras` framework as a built-in dataset, and can be loaded into memory using the `load_data` method of the `imdb` class contained in the `keras.datasets` module. The provided dataset comes preprocessed with punctuation and capitalization removed. No stemming/lemmatization was performed in this study and stop words were not removed. In the `Keras` representation of LMRD, each review is encoded as an array of word indices, where in this case the corpus vocabulary is indexed in descending order of word frequency. By default, index 1 of the vocabulary corresponds to the ⟨START⟩ character, which marks the start of a review, index 2 corresponds to a word not in the vocabulary (often denoted ⟨UNK⟩), and index 3 is skipped. Thus, the most frequent word in the corpus, "the", has an index of 4. Index 0 is reserved as a padding character. The complete LMRD vocabulary consists of 88,584 unique words, but the top most frequent 12,000 words were retained in the present study; this is a selectable parameter when importing LMRD using `Keras`. One reason for this choice is that the top 12,000 words already account for over 95% of all the words in the documents combined; the 12,001st most frequent word only appears in 49 ($< 0.1\%$) documents. Another reason is that working with a smaller vocabulary reduces the number of parameters that need to be learned in the embedding layer of the RNN.

Below is an example of the first ten words of the first training set instance along with its encoding:

> ⟨START⟩ this film was just brilliant casting location scenery story ...
>
> [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, ...]

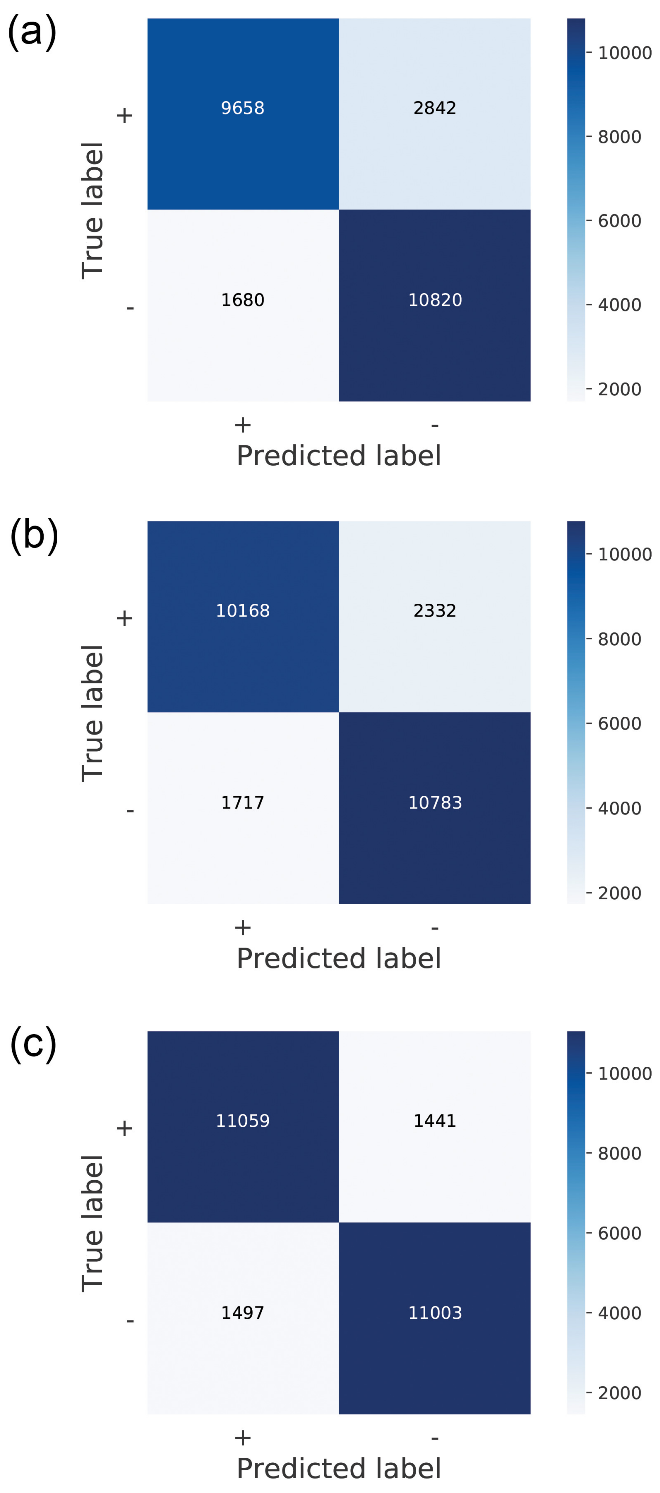

FIG. 4. Confusion matrices for (a) NB, (b) Bernoulli NB, and (c) SVM classifiers. The results correspond to predictions made on the test set.

For all reviews, the first entry in the array is a "1", which is the ⟨START⟩ character, and in this example, the second entry is "14", which reflects the fact that the word "this" is the 14th most common word in the corpus.

As mentioned in Section IIB, the movie reviews should be encoded in the BoW representation for use with the NB and SVM models. To this end, each review was first converted back to its text representation (removing the ⟨START⟩ character) and then to the BoW encoding using the `CountVectorizer` class in the `sklearn.feature_extraction` module of the `scikit-learn` library. The `CountVectorizer` class returns either integer or binary arrays, so that the output can be used with either the NB or Bernoulli NB models, respectively.

The encoding provided by `Keras` can be used almost as is for RNNs, except for one small modification. To allow for vectorization, the input data must all be of the same dimension. Since in general reviews differ in length, the encoded arrays need to somehow all be made the same dimension. One common solution is to make the length of all the arrays in the dataset equal to the length of the longest array, and to pad the shorter reviews with zeros. That is the method used here, and this is easily implemented in `Keras` using the built-in function `pad_sequences` contained in the `keras.preprocessing.sequence` module. Recall that reviews were capped at 500 words in this study. After fixing the review length, the test set was split into a validation set and a new test set using a ratio of 50:50. The splitting was carried out using the `train_test_split` function of `sklearn.model_selection` in a stratified fashion.

Finally, the weights of the embedding layer of an RNN can either be trained from scratch, pre-trained and frozen, or pre-trained and refined. The first and third cases were explored in this study. A pre-trained embedding layer was generated using the GloVe Wikipedia 2014 + Gigaword 5 pre-trained word vectors. First, a map from word to GloVe vector was generated, and this was used to construct the embedding layer, which is an $n \times m$ dimensional array, where $n = 12,004$, the number of words (plus special characters) retained in the LMRD vocabulary, and m is the dimension of the word vectors ($m = 50, 100, 200, 300$)). The row index of the matrix is the same as the vocabulary index, i.e., row 4 corresponds to the GloVe vector for "the". For words in the LMRD vocabulary but not in the GloVe vocabulary, the word vector was randomly initialized according to a normal distribution, with a mean and standard deviation calculated using all the elements of all the GloVe vectors.

B. Implementation

1. *Naive Bayes*

Training of the NB classifer was implemented in the `scikit-learn` framework using the `MultinomialNB` class contained in the `sklearn.naive_bayes` module. The default parameters were used, which includes Laplace smoothing, as previously explained. The accuracy, precision and recall of the model on the test instances was evaluated using the `predict` method of the trained `MultinomialNB` class instance in conjunction with the `accuracy_score`, `precision_score`, and `recall_score` functions contained in the `sklearn.metrics` module. The `predict` method takes as input either a single instance or an array of instances and returns the predicted class(es). The `accuracy_score`, `precision_score`, and `recall_score` functions take the predicted classes and the actual classes as inputs and return the corresponding metric score. Finally, the confusion matrices presented in Fig. 4 were calculated using the `confusion_matrix` class contained in the `sklearn.metrics` module.

The procedure described above was repeated using the `BernoulliNB` class contained in the `sklearn.naive_bayes` module to evaluate the performance of BoW vectors with binary features, i.e., when the features are either 0 or 1 to indicate the absence or presence of a word in a document, as opposed to the number of times the word appears.

2. *Support Vector Machine*

The linear SVM classifier was also trained in `scikit-learn`, this time using the `LinearSVC` class contained in the `sklearn.svm` module. A hyperparameter grid search was conducted on the C hyperparameter (see Section IIB), over the values 0.01, 0.03, 0.1, 0.3, 1, 3, and 10. Hinge loss was used and default values were selected for all other hyperparameters. The grid search was automated using the `GridSearchCV` class contained in the `sklearn.model_selection` module using accuracy as the scoring metric and 5 folds for the cross validation. The optimal value of C was found to be 0.01. The accuracy, precision, and recall metrics, and the confusion matrix were calculated using the same procedure described in the previous section.

3. *Recurrent Neural Network*

The RNNs were implemented in the `Keras` framework using the basic architecture summarized in Table II. The embedding layer weights were either initialized randomly using the default initializer, or a pre-trained embedding layer constructed from GloVe vectors was used. In the latter case, the weights were were allowed to vary during training (i.e., they were not frozen). For the LSTM layer, dropout (DO) and recurrent dropout (RDO) were both set to either 0.2 or 0.4 to avoid overfitting, and the default parameters were used otherwise (this includes a tanh activation function). Note that DO temporarily severs the input connections to the LSTM cell, while RDO temporarily severs the connections between recurrent units [18]. Sigmoid activation was used in the output Dense layer, and the default values were used for all other parameters. The output dimension of the LSTM layer is typically set to be the same as the dimension of the word embedding vectors. The gradient descent optimizer used was either adam or nadam with a default learning rate of 0.001, and the loss function used was binary cross-entropy. The RNNs were trained for 3 epochs using a batch size (BS) of either 32 or 64, but the weights were only saved if the loss on the validation set decreased during a given epoch.

TABLE II. Basic RNN architecture

Layer	**Input dim**	**Output dim**	**Parameters**
Embedding	12,004[a]	100	1,200,400
LSTM	100	100	80,400[b]
Dense	100	1	101

[a] Note that the dimension here is 12,004 not 12,000 because the first three indices in the vocabulary are reserved for the padding, start, unknown characters, and the fourth index is unused.

[b] When bias units are used, the number of parameters in an LSTM layer is calculated as $4(mn + n^2 + n)$, where m is the dimension of the input vectors and n is the dimension of the output vectors.

TABLE III. Initial diagnostic of RNN performance

DO	**RDO**	**Optimizer**	**BS**	**Test set accuracy**
0.2	0.2	adam	32	0.865
0.4	0.4	adam	32	0.841
0.2	0.2	adam	64	0.847
0.4	0.4	adam	64	0.834
0.2	**0.2**	**nadam**	**32**	**0.880**
0.4	0.4	nadam	32	0.843
0.2	0.2	nadam	64	0.880
0.4	0.4	nadam	64	0.860

Eight initial models were evaluated using a randomized embedding layer to better understand the effect of DO and RDO, the optimizer, and BS on the test set accuracy. Note that the validation set—not the test set—was used for model tuning. The results are shown in Table III. Based on the results, the following hyperparameters were used in all subsequent evaluations: DO = 0.2, RDO = 0.2, optimizer = nadam, and BS = 32, as highlighted in the table.

Two additional RNN architectures were also evaluated, adding complexity to the RNN described above, in an attempt to improve the model. These architec-

TABLE IV. Additional RNN architectures considered

Layer	Input dim	Output dim	Parameters
Embedding	12,004	100	1,200,400
Bidirect. LSTM	100	100	160,800
Dense	100	1	101
Embedding	12,004	100	1,200,004
Conv1D	20,000	100	9632
MaxPooling1D	20,000	100	0
LSTM	100	100	53200
Dense	100	1	101

TABLE V. RNN performance using a GloVe embedding layer

Embedding dim.	Parameters	Test set accuracy
50	620,451	0.897
100	1,280,901	0.899
200	2,721,801	0.900
300	4,322,701	0.899

tures are summarized in Table IV. The same hyperparameter choices described above were applied to these models as well. The number of filters and the kernel size in the Conv1D layer were set to 32 and 3, respectively, and the pool size in the MaxPooling1D layer was set to 2. The test set accuracy of the Bidirectional LSTM model was 0.880 and that of the Conv1D+LSTM model was 0.882. Since neither of these models provided significantly better performance compared to the basic LSTM model, they were not explored further. In addition, all models were also evaluated using a gated recurrent unit (GRU) [19] instead of an LSTM, but the results were not different enough to further discuss this architecture.

IV. RESULTS

Using the RNN architecture and optimized hyperparameters described in the previous section, the randomly initialized embedding layer was replaced with an embedding layer constructed from pre-trained GloVe vectors of dimension 50, 100, 200, or 300. In each case, the output dimension of the LSTM layer was matched to the dimension of the GLoVe vectors. Since the GloVe vectors are pre-trained on a different corpus, the weights were allowed to vary during the training. The results are summarized in Table V.

In each case, the test set accuracy exceeded the performance of all the RNNs evaluated up to this point. However, one important consideration is the amount of time required for training and prediction (Intel 2.7 GHz i7-8559U quad core), which is summarized in Table VI. As can be seen, training an RNN is an extremely time-consuming process, with even the simplest model considered here taking around 6 minutes/epoch to train on 25,000 instances and around 30 seconds to make predictions on 12,500 testing instances. Although the 200-dimensional GloVe vectors provide the highest test set accuracy of 0.900, the 50-dimensional GloVe vectors provide virtually equivalent performance; moreover, they can be trained around 1.4 times faster and predictions can be made about twice as quickly. Thus, the 50-dimensional GloVe RNN (hereafter GloVe50-RNN) model likely provides the best trade-off between accuracy and efficiency.

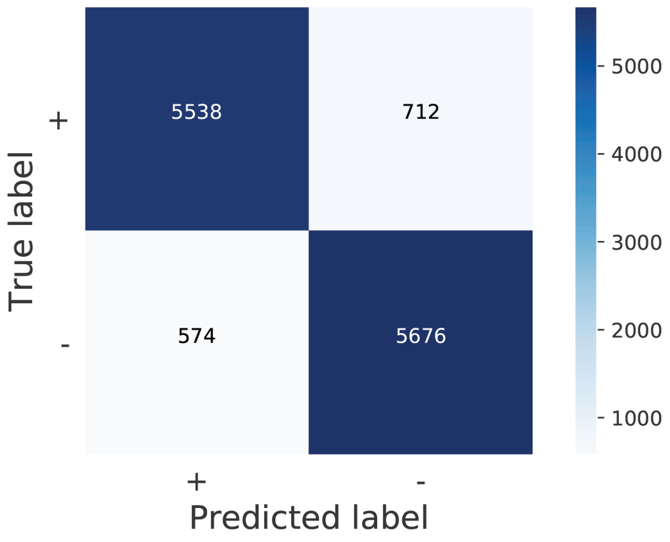

FIG. 5. Confusion matrix for GloVe50-RNN. The results correspond to predictions made on the test set. Note that the test set used for the RNN models is half the size of that used for the benchmark models because 50% of the instances were allocated into a validation set used for model tuning.

TABLE VI. RNN training and prediction times

Embedding dim.	Training (s/epoch)	Prediction (s)
50	354	27
100	384	30
200	492	52
300	842	115

The recall and precision for GloVe50-RNN were calculated to be 0.889 and 0.908, respectively, which are both higher than the values observed for all the benchmark models. The confusion matrix for this model is shown in Fig. 5. The GloVe-50 RNN model outperforms all of the benchmark models in terms of all metrics considered in this study. Although the improvement in accuracy compared to NB and Bernoulli NB is significant, there is only a slight improvement of around 2% over the much simpler and faster linear SVM classifier. In cases where accuracy is critical, the GloVe50-RNN model is the best solution; however, if speed is the top priority, the SVM is the better choice. Interestingly, GloVe50-RNN achieves a significantly higher precision score than the SVM classifier, as reflected in the lower proportion of false positives. This could be beneficial, as most people would likely rather

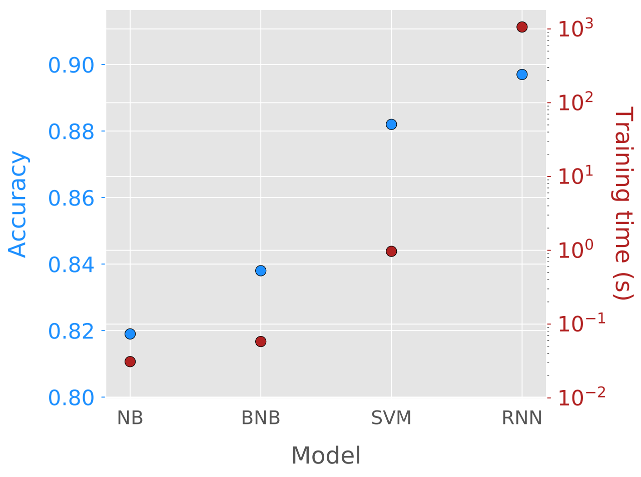

FIG. 6. Accuracy and training time for models considered. Note the log scale on the right vertical axis.

prefer to miss out on watching a well-reviewed movie as a result of a low-recall model than waste time watching a poorly-reviewed movie as a result of a low-precision model.

V. CONCLUSION

The performances of several RNN architectures for binary sentiment classification were evaluated and compared to those of two benchmark models, NB and SVM, using the LMRD dataset and the accuracy metric. The final RNN selected, denoted GloVe50-RNN, has a embedding layer consisting of fine-tuned pre-trained 50-dimensional GloVe word vectors, a single LSTM layer of output dimension 50, and an output sigmoid layer consisting of a single neuron. To prevent overfitting, dropout and recurrent dropout in the LSTM layer were both set to 0.2, and training was limited to 3 epochs. The layer weights were only saved if the validation loss decreased between epochs. To avoid biasing the model, a separate validation set was used for hyperparameter tuning, and the model accuracy was evaluated using a completely unseen test set.

The NB and Bernoulli NB classifiers achieved an accuracy of 0.819 and 0.838, respectively, and the linear SVM classifier achieved an accuracy of 0.882. The highest accuracy obtained in this study for an RNN model was that for GloVE200-RNN, at 0.900, although this model was rejected in favor of GloVe50-RNN, which achieved an accuracy of 0.897, due to considerations of computational cost. Fig. 6 shows the accuracy and training duration for the benchmark models and GloVe50-RNN. It is clear that the additional performance benefit that GloVe50 provides may not be enough to justify its use in all situations, particularly when speed is a concern. However, if accuracy or precision are top priorities, then GloVe50-RNN is likely the best choice.

One of the main issues that prevents the RNNs studied here from performing better is likely the size of the LMRD dataset. Even though it is more than 10 times larger than the dataset used in the seminal Pang et al. study, 25,000 instances is still quite small. When the RNNs were trained for 4 or more epochs, the training set accuracy diverged from the validation set accuracy, which was evidence of overfitting, and is a symptom of insufficiently large datasets. In the future, it would be interesting to evaluate the performance of the RNNs studied here on the SAR14 dataset [20], which also consists of IMDB movie reviews and is much larger (233,600 instances), but contains raw rating scores rather than positive and negative and is class imbalanced. It would not be surprising if even better performance was observed on this larger dataset.

[1] M. K. Mika V. Mantyla, Daniel Graziotin, Computer Science Review **27**, 16 (2018).
[2] D. Droba, American Journal of Sociology **37**, 410 (1937).
[3] M. Rambocas, Journal of Research in Interactive Marketing **12**, 146 (2018).
[4] O. E. A.-M. Popescu, *Natural Language Processing and Text Mining* (Springer, 2007).
[5] A. Pak and P. Paroubek, in *LREC* (2010).
[6] A. Tumasjan, T. O. Sprenger, P. G. Sandner, and I. M. Welpe, in *ICWSM* (2010).
[7] B. Pang, L. Lee, and S. Vaithyanathan, CoRR **cs.CL/0205070** (2002).
[8] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, CoRR **abs/1301.3781** (2013).
[9] J. Pennington, R. Socher, and C. D. Manning, in *EMNLP* (2014).
[10] Z. C. Lipton, CoRR **abs/1506.00019** (2015).
[11] T. Young, D. Hazarika, S. Poria, and E. Cambria (2017).
[12] `http://ai.stanford.edu/~amaas/data/sentiment/`.
[13] D. Jurafsky and J. H. Martin (2008).
[14] A. S. Nurulhuda Zainuddin, in *I4CT* (2014) p. 333.
[15] T. Mullen and N. Collier, in *EMNLP*, Vol. 4 (2004) pp. 412–418.
[16] G. G. Esparza, A. de Luna, A. O. Zezzatti, A. Hernandez, J. Ponce, M. Álvarez, E. Cossio, and J. de Jesus Nava, in *Distributed Computing and Artificial Intelligence, 14th International Conference* (Springer International Publishing, Cham, 2018) pp. 157–164.
[17] A. Gron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. (O'Reilly Media, Inc., 2017).
[18] Y. Gal and Z. Ghahramani, in *NIPS* (2016).
[19] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, CoRR **abs/1412.3555** (2014).

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