CS 229 Final Project Report
Predicting the Likelihood of Response in a Messaging Application

Tushar Paul (SUID: aritpaul), Kevin Shin (SUID: kevshin)

December 16, 2016

1 Introduction

A common feature of messaging services such as Facebook and iMessage is to send users a notification when they receive a message. However, in a large messaging group, not all messages may be relevant and a user could get overwhelmed with the amount of notifications.

The goal of this project was to identify the best learning model to classify incoming messages into {`likely to respond, unlikely to respond`}. This data could potentially be used to make a "smart notification" that only sends the user a notification when they are likely to respond. The input to the learning models were messages along with the properties that came with them including: the sender, time stamp, type of message, and message number. We explored five models: Bernoulli Naive Bayes, Gaussian Naive Bayes, Logistic Regression, Support Vector Machines (SVM) with a Radial Basis Function (RBF) Kernel, and a Neural Network with a Long Short Term Memory (LSTM) layer.

2 Prior Work

Classifying short messages is not a novel problem. The most similar prior work to our problem comes from [1] and [2]. Both studies analyze tweets to classify them and determine user interest. This is similar to our task since both tweets and messages tend to be short and numerous, and we want to classify our messages based on the user's interest. In addition to considering just the text via a bag of words representation, the researchers exploited the social relationships to help them classify text [1] [2]. We found merit in this approach, and thus considered other features (such as the message sender) besides just the message text.

Another similar prior work is spam filtering [3]. Spam filtering is similar to our problem in that it aims to filter out emails the user is unlikely to be interested in, just as we aim to filter out notifications for messages that the user is unlikely to be interested in. We also need to be able to minimize the false negatives (filtering out messages that shouldn't have been filtered) which is the same problem that spam filtering addresses. The work done in [3] evaluates the performance of five different types of Naive Bayes model, which we leverage.

In terms of neural networks, we found that [4] and [5] both used state of the art recurrent neural networks to classify short-text and we followed suit. Both of these studies shows that combining convolutional and recurrent neural networks can produce superior results. In addition, [4] showed that taking into account sequences (as all short text within the messaging context is sequential) and using a Long Short Term Memory (LSTM) neural network (discussed in the **Learning Methods** section) can produce higher accuracy levels than that of other models [4].

3 Dataset and Features

In this section, we will discuss our dataset and how we processed it to extract relevant features.

3.1 Data Source

The data used for this project is from Facebook Messenger. A user can download all of his or her data directly from Facebook [6]. Our dataset came from one Facebook Messenger Group that both group members were a part of. There are 27 users in this group with over 17,300 messages over a period of one year. After accounting for multiple consecutive messages sent by any given user (and counting it as one message), the total amount of messages came out to 10,000 messages.

3.2 Data Preprocessing

Facebook offered the data in an html page with all of the group messages in the page. We wrote a python script to parse this web page, extract only the relevant messages, and convert it into an easier to handle `.csv` file format. This data was then labeled, randomized and then split into training and test data with 30% of the data reserved for testing and 70% reserved for training. We also computed a vocabulary where we assigned each word in our corpus an integer. For our Neural Network, we converted the message into a sequence of integers by mapping each word to its corresponding integer representation. For all other models, we converted the message to a bag-of-words representation, where each index i of the input vector contained the number of times word i in our vocabulary appeared.

3.3 Data Labeling

We used this data to visualize the distribution of the average response time of each message as shown in **Figure 1**.

Using the distribution we found, we decided to use $response\ threshold = 20$ min. since it was roughly within the mean and 1.5 standard deviations. With a mean of $\mu = 4.63$ min, most messages will have been "responded to" about 3-4 messages within the 20 min response threshold. This is a reasonable assumption given the dynamic nature of conversations within a messaging group.

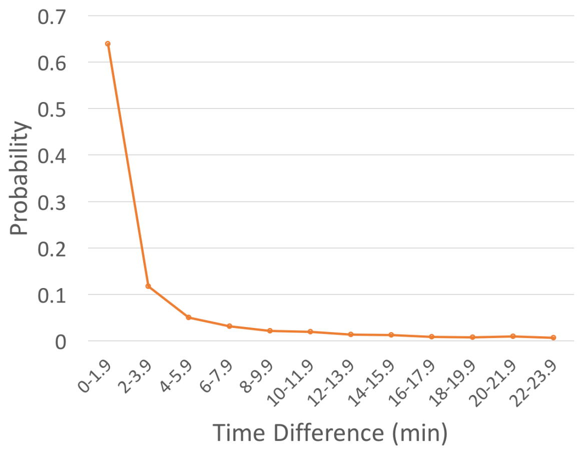

Figure 1: Probability Distribution of Time Differences Between Messages

The response threshold was then used to label whether or not a message has been responded to by a particular user. If we saw the user in question send a message, all messages sent 20 minutes before was marked as "responded to."

3.4 Features

From each message in the dataset, we extracted several features that we used to help us classify the messages. They were:

- Current `Number of messages` in the conversation - grouped into 11 buckets from `[0-9, 10-19, ..., 100+]`[1]
- `Message Length` (word count) - grouped into 11 buckets from `[0-99, 100-119, ..., 1000+]`
- The `sender` of the message - each sender given a unique index
- `Time of day` - indexed between 0-23 with 0 being 12:00 am - 12:59am, 1 being 1:00 am - 1:59 am, and so on
- `Day of week` - indexed between 1-7 with 1 being Monday and 7 being Sunday
- `Message type` (link, question, statement)
- `Raw text` - turned into a vocabulary as discussed in the subsection **Data Preprocessing**

Since the experimental results differed for each member of the group and each member's behavior in the group chat changed over time, we focused the results on one user who had the most consistent behavior over the lifespan of the group. From this point on, all of the data is conditioned on User X's responses.

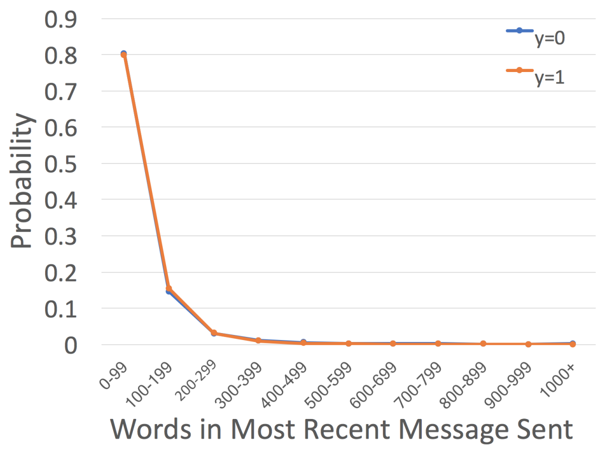

Figure 2: Probability of User X's Message Length Conditioned on Response or No Response

To gain some intuition for User X's data and justify the use of each feature in our model, we plotted $p(x_i|y)$ (i.e. $p(feature = x|y)$ where $y = 1$ if user responded and $y = 0$ if they didn't respond). **Figure 2** shows a plot of $p($`Message Length|User X's Responses`$)$. For this particular user, we found that the length of the message did not affect their response rate. We discuss the implications of this observation further in the **Results Section**).

4 Learning Methods

We are using **supervised learning** to classify messages received into one of two possible classes: {`likely to respond`, `unlikely to respond`} (also referred to as $y = 1$ and $y = 0$ respectively). If

[1]The current conversation is defined as a period in time in which two or more participants are actively sending messages. The current conversation is typically (though not necessarily) preceded by and followed by with a longer period of time in which there are no messages sent.

we classify a message as `likely to respond`, that means we would send the user a notification. We used and compared five different classifiers: Bernoulli Naive Bayes, Gaussian Naive Bayes, Logistic Regression, SVMs with an RBF Kernel, and a Neural Network with an LSTM Layer.

4.1 Naive Bayes

Naive Bayes is a *Generative Algorithm*; that is we predict:

$$\arg\max_y p(y) \prod_{j=1}^{n} p(x_j|y)$$

The difference between our two Naive Bayes models was in our $p(x|y)$. For Bernoulli Naive Bayes, we had [7]

$$p(x_j|y) = p(j|y)x_j + (1 - p(j|y))(1 - x_j)$$

and for Gaussian Naive Bayes we had [7]

$$p(x_j|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_j - \mu_y)^2}{2\sigma_y^2}\right)$$

4.2 Logistic Regression

In contrast to Naive Bayes, Logistic Regression is a *Discriminative Algorithm*, which means we attempt to learn $p(y|x)$ directly. For logistic regression, we use $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$. We then maximize θ in:

$$p(y|x;\theta) = (h_\theta(x))^y (1 - h_\theta(x))^{(1-y)}$$

and choose $\text{argmax}_y \, p(y|x;\theta)$.

4.3 SVM with RBF Kernel

We use an SVM classifier, which attempts to maximize the margin between two classes, along with an ℓ_1 regularization to prevent overfitting. We use an RBF Kernel to map the features. Combining the dual formulation of the problem and the definition of an RBF Kernel from the CS229 notes, we get that our optimization problem is:

$$\max_\alpha W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j K(x_i, x_j)$$

$$\text{s.t. } 0 \le \alpha_i \le C, i = 1, \cdots, m; \; \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

where

$$K(x,z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right)$$

We predict $y = 1$ if

$$\sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x) > 0$$

4.4 Recurrent Convolutional Neural Network with LSTM Layer

The structure of our Neural Network was as follows:

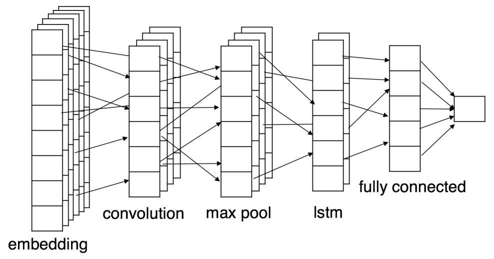

Figure 3: Neural Network with LSTM Layer

The embedding layer learns how words are related, as in word2vec [11]. This is a skip-gram model that maximizes

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

where $w_1, w_2, \cdots, w_T$ is a sequence of words, and $p(w_{t+j}|w_t)$ is defined using the softmax function.

Convolution layers are mostly used in the context of image classification problems [12], but they have seen success in text classification tasks as well [4]. Each neuron in this layer uses ReLu activation [13]; that is, $f(x) = \max(0, x)$. As in many neural network architectures [13], we then pass the outputs of the convolution layer to a max-pooling layer to reduce the dimensionality of the inputs for the next layer.

The outputs of the max-pool layer then are fed into an LSTM layer. Because of their ability to learn long-term relationships over time in sequences like text, LSTMs have become popular in NLP tasks [9]. Each LSTM cell consists of an input gate, and output gate, and a 'forget gate' which learns "to reset [the] memory cell contents once they are not needed any more" [10]. The following equations [4] describe an LSTM cell for the t^{th} word x_t in the sequence. x_t, h_{t-1}, and c_{t-1} are inputs, i_t, o_t, and f_t are the input, output, and forget gates, respectively, and h_t and c_t are the outputs:

$$\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
\tilde{c_t} &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t * c_{t-1} + i_t * \tilde{c_t} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
h_t &= o_t * \tanh(c_t)
\end{aligned}$$

The W and U terms are weight matrices and the b terms are bias vectors. Visually, each LSTM cell looks like:

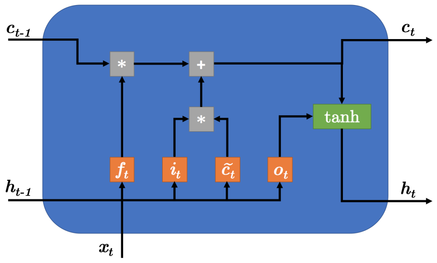

Figure 4: Structure of Cell in LSTM Layer

5 Experiments

We used a combination of `scikit-learn` [7] and `keras` [14] to implement our models. We ran several experiments to choose our parameters, including a combination of automatic and manual tuning.

For our SVM and Logistic Regression classifiers, we used Grid Search Cross Validation with 3 folds. We searched over values of C between 1 to 1000 for both models, and also searched over σ for SVM. We picked the best parameters based on the resulting F1 score. We picked 3 folds due to the computational limitations of our development machines.

On top of Cross-Validation, we ran feature selection on all our models. We tried all combinations of features for each model, and then selected the combination that gave us the best combination of F1 score and Error for each model.

Finally, for our Neural Network we ran several experiments related to the length of the input message. Because we needed to either pad or truncate each message to be the same size, our choice of message length had a large effect on our model's results. We will discuss the implications of our choice in the following section.

6 Results

Our primary metrics for evaluation were error and F1 score. F1 score is defined as the harmonic mean of the precision and recall [8]. We included F1 score as a metric because we found that our classes were imbalanced. Due to privacy concerns, we are unable to present explicit example messages and their classifications. We are, however, able to discuss the performance of the models as a whole. The ideal classifier would be at the top left of **Figure 5**.

Model	Error	F1 Score
Bernoulli NB	0.394	0.022
Gaussian NB	0.594	0.573
Logistic Regression	0.294	0.617
SVM w/ RBF Kernel	0.325	0.654
LSTM Neural Net	**0.244**	**0.729**

Table 1: Training Set Results

Model	Error	F1 Score	Precision	Recall	FP Rate	FN Rate
Bernoulli NB	0.403	0.014	**0.769**	0.007	**0.001**	0.499
Gaussian NB	0.593	0.577	0.406	**0.999**	0.499	**0.001**
Logistic Regression	0.296	0.622	0.643	0.602	0.185	0.285
SVM w/ RBF Kernel	0.326	0.654	0.573	0.763	0.279	0.192
LSTM Neural Net	**0.292**	**0.682**	0.609	0.774	0.252	0.185

Table 2: Test Set Results

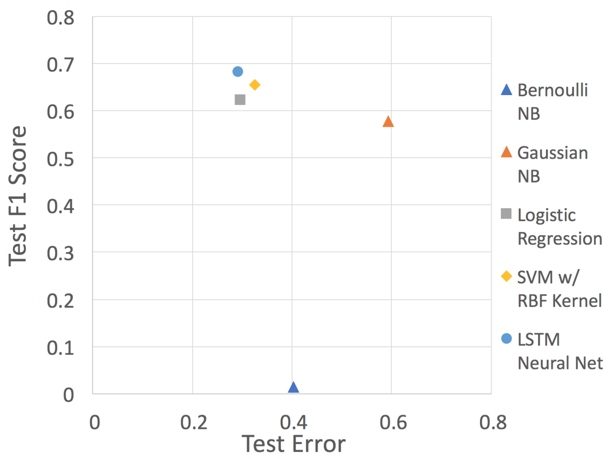

Figure 5: F1 Score vs Error

From **Figure 5**, it is apparent that the Naive Bayes models performed poorly, and the rest of our models were clustered around more reasonable performance. Naive Bayes is not well-suited to datasets with imbalanced classes like ours unless the model makes heavy use of heuristics [15].

To tackle the issue of imbalanced classes, for our other three models we weighted each example inversely proportional to their weights:

$$w_{C_1} = \frac{\text{num train examples}}{\text{num train examples that are of class } C_1}$$

As suggested by the TA, we attempted to use Logistic Regression on our data. Logistic Regression performed better than Naive Bayes, but its F1 score was not as high as SVM and our Neural Network. Furthermore, it had a higher False Negative rate than a False Positive rate, as shown in **Figure 6**. For our application, we would prefer to send a user a notification that they might not be interested in over missing one that might be important. Therefore Logistic Regression's higher False Negative rate is undesirable.

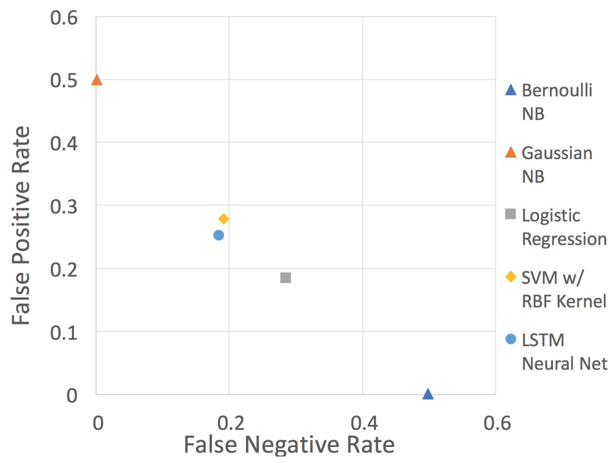

Figure 6: False Positive vs False Negative Rate

Although SVM had a higher error rate than Logistic Regression, it had a higher F1 score and a lower False Negative rate. We were able to get this higher F1 score compared to Logistic regression because we were able to tune σ as well as C for this model.

Finally, we found that our Neural Network model performed the best overall. We believe that this is because the LSTM layer was able to learn more information from the input text. Whereas we used a bag-of-words representation for the other models, we kept the structure of the text intact for LSTM, and the LSTM layer was able to make effective use of this information.

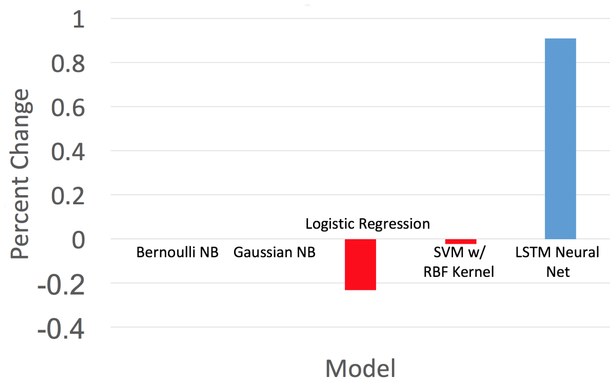

Figure 7: F1 Score Percent Change When Removing Message Length Feature

As **Figure 7** shows, one interesting result from our feature selection was that `message length` either had little effect on or hurt the performance of our models. Our analysis of our data corroborates this result. As **Figure 2** indicated, the length of the message did not have an effect on whether or not a user replies to a message. It seems that the noise of the message length caused the LSTM layer to overfit on that feature, leading to higher test error and lower F1 score.

A related result is that truncating the messages that we fed into our Neural Network increased the performance of the model. This is because if the length of each message was long, most messages would be padded with 0's. The result of this padding is that the Neural Network would learn message length implicitly as a feature. Because we found that learning message length was undesirable for this model, we reduced our input messages to 20 words, and saw an increase of 2% in F1 score.

7 Conclusions

In conclusion, we found that our Neural Network model had the best performance on our data. Our Neural Network achieved 71% accuracy while maintaining a 0.68 F1 score, which is in the range of state-of-the-art short text classifiers [4]. We found that the including length of a message as a feature did not appreciably affect the results of most models, and hurt the performance of our Neural Network. Finally, we found that Naive Bayes failed to offer meaningful predictions for this problem due to our imbalanced classes.

Future work on this problem includes exploring additional neural network architectures, extracting more explicit features from our messages, and trying our model on different group chats. Given more compute resources, we would like to perform more thorough Cross Validation with more folds. We would also like to see how well our models perform in the real world by deploying them on a real chat, and receiving user feedback. This would involve either writing our own chat client or integrating with an existing client.

References

[1] Li, R., & Zhang, Y. (2013). Social-Correlation Based Mutual Reinforcement for Short Text Classification and User Interest Tagging. Advanced Data Mining and Applications Lecture Notes in Computer Science, 444-455. doi:10.1007/978-3-642-53914_38

[2] Lee, Kathy, Diana Palsetia, Ramanathan Narayanan, Md Mostofa Ali Patwary, Ankit Agrawal, and Alok Choudhary. ”Twitter trending topic classification.” In 2011 IEEE 11th International Conference on Data Mining Workshops, pp. 251-258. IEEE, 2011.

[3] Metsis, Vangelis, Ion Androutsopoulos, and Georgios Paliouras. ”Spam filtering with naive bayes-which naive bayes?.” In CEAS, pp. 27-28. 2006.

[4] Lee, Ji Young, and Franck Dernoncourt. ”Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks.” arXiv preprint arXiv:1603.03827 (2016).

[5] Lai, Siwei, Liheng Xu, Kang Liu, and Jun Zhao. ”Recurrent Convolutional Neural Networks for Text Classification.” In AAAI, pp. 2267-2273. 2015. Harvard

[6] Facebook. Accessing Your Facebook Data. Retrieved October 21, 2016, from `https://www.facebook.com/help/405183566203254?helpref=faq_content`

[7] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[8] Powers, David Martin. ”Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.” (2011).

[9] Ghosh, Shalini, et al. ”Contextual LSTM (CLSTM) models for Large scale NLP tasks.” arXiv preprint arXiv:1602.06291 (2016).

[10] Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins. ”Learning to forget: Continual prediction with LSTM.” Neural computation 12.10 (2000): 2451-2471.

[11] Mikolov, Tomas, et al. ”Distributed representations of words and phrases and their compositionality.” Advances in neural information processing systems. 2013.

[12] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. ”Imagenet classification with deep convolutional neural networks.” Advances in neural information processing systems. 2012.

[13] Karpathy, Andrej. ”CS231n Convolutional Neural Networks for Visual Recognition.” *CS231n Convolutional Neural Networks for Visual Recognition.* N.p., n.d. Web. 12 Dec. 2016.

[14] Chollet, François. ”Keras” *Github.* N.p., Web. 12 Dec. 2016

[15] Rennie, Jason D., et al. ”Tackling the poor assumptions of naive bayes text classifiers.” *ICML.* Vol. 3. 2003.