READ IT ON REDDIT: LSTM NETWORKS FOR TEXT CLASSIFICATION

Luis Ulloa
ulloa@stanford.edu

Tassica Lim
tlim98@stanford.edu

Tatiana Wu
twu99@stanford.edu

ABSTRACT

Text classification is an important issue in the field of natural language processing. The difficulty of text classification stems from the nuances in language, such as differences in dialect and spelling. To tackle this task, we classified Reddit self-posts into genres. We implemented three baseline models: multinomial naive Bayes, multinomial logistic regression with a bag-of-words (BOW), and multinomial logistic regression with word embeddings. From there, we trained three long short-term memory (LSTM) network variants. Our best model achieved a test accuracy of 78.7%. Common sources of misclassifications were found to be an imbalanced dataset (i.e., significantly more examples of Reddit self-posts for certain genres than for others), similarity between different genres (e.g., *rpg* and *board game*), and generality of certain genres (e.g., *meta*, *advice/question*, etc.).

Keywords NLP · Classification · GloVe · LSTM

1 Introduction

Text classification is a common task in natural language processing (NLP). The difficulty of this task stems from the inconsistency of written communication — i.e., improper conventions, geographic differences, and constantly-evolving jargon. Extracting the semantic meaning of a piece of text has broad applications that range from sentiment analysis of product reviews to fraud and spam detection.

We treat this task as a supervised learning classification problem. Our project seeks to classify Reddit self-posts, which frequently display these very inconsistencies, to their corresponding genres. Given the title and body of a piece of text, we return the probability distribution over all the classes and classify each text into the class with the highest probability. For our project, we implemented several baseline models, as well as more complex long short-term memory (LSTM) networks, evaluating our models based on classification accuracy as well as balanced accuracy.

2 Related Work

There has been a significant amount of work in text classification as an NLP task. Jones approaches this problem using the same dataset, classifying Reddit self-posts to subreddits (each of which falls within a larger *genre*) using bag-of-words (BOW) models with naive Bayes and FastText. [1] Fiallos and Jimenes (2019) consider common approaches to text classification such as k-nearest neighbors, decision trees, kernel methods, and neural networks. [2] They specifically note the advantage of using word embeddings in their models to learn more context from the order of words in a piece of text, as opposed to a BOW approach.

In more recent years, there has been a shift to more complex models using a combination of word embeddings and LSTM networks with and without convolution layers. The most popular word embeddings used are GloVe [3] and Word2vec [4]. Zhou et al uses a combination of pre-trained Word2vec (one of the most popular word embeddings), LSTMs, and convolutional neural networks (CNNs) for text classification. [5]

3 Data

We used the Kaggle *Reddit self-post classification task* dataset [6], which consists of over one million Reddit self-posts among 1,013 subreddits. Each subreddit consists of 1,000 self-posts.

3.1 Input and Output

Specifically, each example contains the title (*title*) and text (*selftext*) of the post, as well as the subreddit classification (*subreddit*). The dataset also provided information mapping each subreddit to a genre (*genre*), which we added as

an additional column to our data. In a preliminary run, we found attempting to classify each self-post to the appropriate subreddit to be a difficult and frequently intractable task given that each class only had 1,000 examples, especially on our baseline models. We therefore chose to classify our data by genre, of which there are 39 classes.

The inputs to each of our classifiers were *title* and *selftext*, and the outputs were probability distributions over the genres. Specifically, we outputted vectors representing the likelihood of each example (a self-post consisting of *title* and *selftext*) being a particular genre. We classified each self-post to the genre with the highest probability.

3.2 Data Preprocessing

We did some minor preprocessing to our data to clean the text:

- We lowercased all text within *title*, *selftext*, and *genre*.
- Within *title* and *selftext*, we removed commas, periods, and text within and including angle brackets (e.g., <lb>).
- We also wrapped question marks and exclamation points in whitespace. [1]

4 Methods and Experiments

4.1 Feature Extraction

For each of our models, we used one of two common feature extractors: BOW and word embeddings.

BOW. In the BOW models, we concatenated the textual inputs (*title* and *selftext*) and tokenized the text to convert the input into a vector of word frequencies.

Word embeddings. Word embeddings are learned representations of words, where words with similar meaning have vector representations that are close to each other in the vector space. [7] Although it is not uncommon for models to train word embeddings on the training data, we used pre-trained word embeddings. Specifically, we used a 300-dimensional GloVe model with a vocabulary size of 400,000. [3]

4.2 Baseline Models

We first implemented three baseline models: multinomial naive Bayes with BOW, multinomial logistic regression with BOW, and multinomial logistic regression with word embeddings.

4.2.1 Multinomial Naive Bayes

Naive Bayes is a common baseline model for text classification tasks. It is a conditional probability model that seeks to calculate, given an input $\mathbf{x_i}$ with features $\phi(\mathbf{x_i})$ of dimension d,

$$p(C_k \mid \phi(\mathbf{x_i}))$$

for each of the classes (i.e., genres) C_k. To do so, naive Bayes assumes that all features in $\phi(\mathbf{x_i})$ are mutually independent, conditioned on C_k. The model therefore calculates probabilities

$$p(C_k \mid \phi(\mathbf{x_i})_j) \propto p(C_k) \prod_{j=1}^{d} p(\phi(\mathbf{x_i})_j \mid C_k)$$

and classifies $\mathbf{x_i}$ to the class with the highest probability.

Algorithm Naive Bayes

Fitting:
for all inputs $\mathbf{x}$ **do**
  for all genres C_k **do**
    calculate

$$p(C_k \mid \phi(\mathbf{x_i})_j) \propto p(C_k) \prod_{j=1}^{d} p(\phi(\mathbf{x_i})_j \mid C_k)$$

  end for
end for

Prediction:
for all inputs $\mathbf{x}$ **do**
  classify $\hat{y} = \arg\max_{C_k} p(C_k)$
end for

We fit a multinomial naive Bayes classifier using a BOW model, where each input comprised a vector of 712,033 features, with each feature representing the frequency of a particular word within the input.

4.2.2 Multinomial Logistic Regression

We also implemented two logistic models as baselines, one with BOW and one with word embeddings. Logistic regression works as follows: Given an input $\mathbf{x_i}$ with features $\phi(\mathbf{x_i})$, the model outputs a probability

$$p(C_k) = \frac{e^{\beta_{C_k} \cdot \phi(\mathbf{x_i})}}{\sum_{\hat{k}=1}^{K} e^{\beta_{\hat{k}} \cdot \phi(\mathbf{x_i})}}$$

for every class C_k. We then classify $\mathbf{x_i}$ to the class with the highest probability.

[1] Note: We included question marks and exclamation points in our BOW models in case they provided some information about the possible subset of genres under which a self-post may fall. For example, a question mark may indicate that a self-post is likely classified within the *advice/question* genre.

Algorithm Logistic Regression

Fitting:
for all inputs $\mathbf{x}$ **do**
  for all genres g **do**
    calculate

$$p(C_k) = \frac{e^{\beta_{C_k} \cdot \phi(\mathbf{x})}}{\sum_{\hat{k}=1}^{K} e^{\beta_{\hat{k}} \cdot \phi(\mathbf{x})}}$$

  end for
end for

Prediction:
for all inputs $\mathbf{x}$ **do**
  classify $\hat{y} = \arg\max_{C_k} p(C_k)$
end for

For both logistic regression models, we used the categorical cross-entropy loss. This loss function is defined as

$$L(y, \hat{y}) = -\sum_{i=0}^{N} \sum_{j=0}^{M} (y_{ij} \cdot \log(\hat{y}_{ij}))$$

where $\hat{y}_{ij}$ is the predicted value of example i for class j, and y_{ij} is the corresponding true value (given M classes and N examples). Note that only one class can be correct, so the true values are represented as a one-hot encoded vector (i.e., a vector of all 0's and a 1 for the single true value). Intuitively, the closer the model's outputs are to this one-hot encoded vector, the lower the loss.

We fit both logistic regression models using SAGA, which is an incremental gradient method that works well for large datasets [8], and cross-entropy loss with a tolerance of 0.0001. Both models were trained for 500 iterations.

For the logistic regression classifier with BOW, we use the same input as for naive Bayes. Specifically, our input was a matrix where each row consisted of the 712,033 features representing word frequencies.

For the logistic regression with word embeddings, we mapped every word in each input $\mathbf{x_i}$ to a word vector using the pre-trained GloVe model. We then averaged over all the word vectors for $\mathbf{x_i}$ to yield a 300-dimensional vector. If a word was not in the GloVe vocabulary, we ignored it; if an entire input was not in the vocabulary, we set the vector to the zero vector rather than removing the example from our data. For faster convergence of SAGA, we centered the vectors to the mean and component-wise scaled by variance. [2]

4.3 Oracle

Our oracle was a human with access to tools such as search engines because humans have a much better understanding of the nuances of language than any algorithm, and they also have the ability to learn new information without large amounts of additional data. To determine the accuracy of the oracle, we classified a random sample of 150 self-posts to their predicted genres based only on the *title* and *selftext*. Our oracle achieved a classification accuracy of 98%, well above the performance of any of our models, including the LSTM networks we introduce next (see the Results and Discussion section for performance results).

The gap in performance between the oracle and all of our other models occurs between the oracle is able to interpret inconsistencies in the data that make text classification so difficult of a task.

4.4 LSTM Models

When humans read and process text, the words read previously influence the understanding of the current and future words. Thus, we considered using an LSTM network, a type of recurrent neural network (RNN), as it can remember important information about different examples over long periods of time. [9]

Unlike other RNNs, LSTM is capable of learning long-term dependencies and mitigating the vanishing gradient problem. Since Reddit self-posts are generally longer than a few words, using an LSTM network allows us to remember key words that show up frequently in certain genres, as well as key sequences of words. This ability to remember information comes from the chain-like nature of recurrent neural networks, where information is passed from one step of the network to the next, and its unique architecture.

In order for LSTMs to retain its memory, it has four layers and retains a cell state C, unlike the standard RNNs that have a single neural network layer. Three of these layers are sigmoid neural net layers that act like gates that allow certain information to pass through. The three gates are:

1. **Forget layer:** This layer determines what information the network is going to remove from the cell state. Using information from the previous hidden layer h_{t-1} and the current input x_t, a vector

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

of values between 0 and 1 is generated, where a value of 0 removes the information.

2. **Input gate layer:** This layer decides which values in the previous cell state C_{t-1} are going to be updated such that

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i).$$

3. **Candidate layer:** This layer creates a vector of candidate values $\tilde{C}_t$ by passing it through the tanh function such that

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C).$$

Combining this layer with the input gate layer, the state is updated.

[2] We used scikit-learn's preprocessing library (`https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scale.html`) to scale the data.

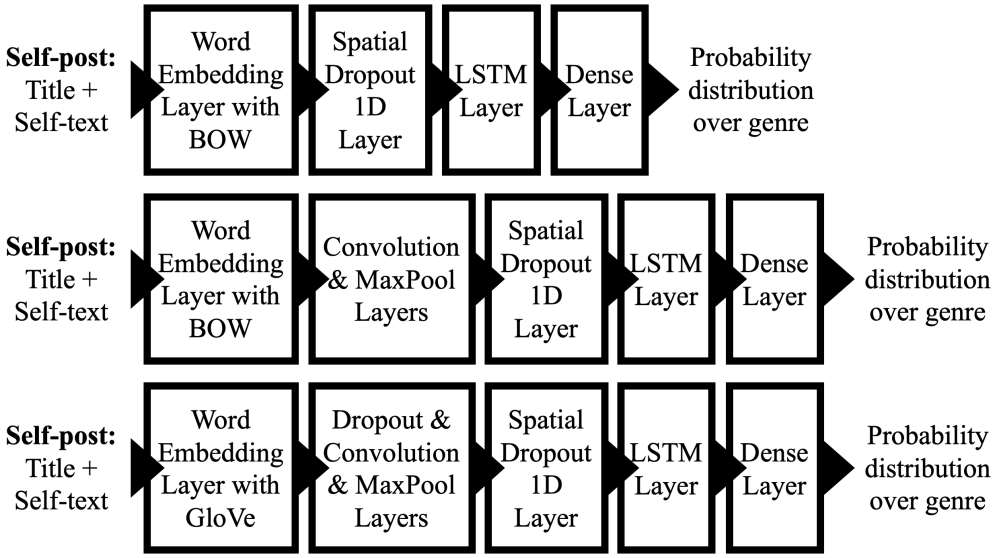

Figure 1: Architectures for Network A (*top*), Network B (*middle*), and Network C (*bottom*).

From these three gates, we can obtain the new cell state C_t such that

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t.$$

Essentially, the new cell state is obtained by removing the values that we want to forget from the previous cell state C_{t-1} and adding the scaled candidate values.

Finally, the fourth layer determines the output by generating a vector that filters the cell state such that

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o).$$

This vector is then multiplied with the cell state C_t after it has passed through tanh (to ensure that the values are between -1 and 1), producing

$$h_t = o_t \odot \tanh(C_t).$$

For our project, we explored several different architectures for our LSTM networks (see Figure 1). We implemented three LSTM network variants. [3]

4.4.1 Network A: LSTM.

Our simplest neural network contains four layers:

1. **Embedding:** Given a matrix of word indices, we used a maximum sequence length of 366 (the median length of all self-posts), truncating longer self-posts, and padding shorter ones with 0s. [4] We also used a vocabulary size of 50,000 words (i.e., we only included the most frequent 50,000 words in our data) against the entire training data of 797,258 unique words. Our embedding layer uses a dimension of 100.
2. **Spatial Dropout:** This involves randomly dropping entire 1D feature maps instead of individual elements. Spatial dropout is a form of regularization that allows our model to generalize and helps prevent overfitting. We use a dropout rate of 0.2.
3. **LSTM:** This layer is as described previously. We used 100 hidden units with both dropout and recurrent dropout rates of 0.2.
4. **Dense:** This is a fully-connected layer in which a softmax activation function is used to output the probability distribution over all the classes.

The architecture for Network A is as follows: Embedding layer, SpatialDropout1D layer, LSTM layer, and Dense layer (top architecture in Figure 1).

4.4.2 Network B: CNN-LSTM

This network is identical to Network A but includes the following additional layer after the embedding layer: a convolutional layer with a ReLU activation function and max pooling. We used 128 filters (i.e., the output dimension is 128) and a kernel size of 5, and we used a pool size of 2 for max pooling.

The architecture for Network B is as follows: Embedding layer, Conv1D layer, MaxPooling1D layer, SpatialDropout1D layer, LSTM layer, and Dense layer (middle architecture in Figure 1).

[3] Implemented using Keras.

[4] We limited the sequence length for computational feasibility, and using longer sequences did not significantly improve our results with our current models.

4.4.3 Network C: CNN-LSTM-GloVe

This network builds upon Network B, with two fundamental changes. First, we use word embeddings instead of word indices. Second, we add an additional dropout layer directly after the embedding layer.

For word embeddings, we use the same 300-dimensional pre-trained GloVe model from our baseline models. Using the frequency matrix, we mapped each word to its word vector, and used this new matrix as our input to the LSTM network. [5]

To use GloVe embeddings, the embedding layer for this network has dimension 300. We initialize the weights for this layer with the pre-trained GloVe weights. Although we increased our vocabulary size to 400,000 words, we continued to restrict our maximum sequence length to 366, as we did with Networks A and B. For the dropout layer, we used a dropout rate of 0.2.

The architecture for Network C is as follows: Embedding layer, Dropout layer, Conv1D layer, MaxPooling1D layer, SpatialDropout1D layer, LSTM layer, and Dense layer (bottom architecture in Figure 1).

4.4.4 Implementation Details

For all three of the LSTM networks above, we trained over twenty epochs with a batch size of 1024. Our models used categorical cross-entropy loss and Adam optimizer. [6]

5 Results and Discussion

5.1 Results

We evaluate our models using classification accuracy and balanced accuracy. Balanced accuracy is essentially the average recall over all the classes (i.e., the accuracy where smaller genres are weighted more). [7] Since our dataset is imbalanced in the number of examples per genre — for example, the *video game* genre comprises 9.9% of our dataset, whereas *travel* comprises only 0.49% of our dataset — the balanced accuracy can reveal if a model is taking advantage of the imbalance in our dataset by guessing that an example falls under an over-represented genre.

The results for our baseline and LSTM models are summarized in Tables 1 and 2, respectively. For multinomial naive Bayes, we obtained an accuracy of 62.6% and a balanced accuracy of 47.2%. For multinomial logistic regression with BOW, we obtained an accuracy of 73.2% and a balanced accuracy of 68.9%. For multinomial logistic regression with scaled GloVe, we obtained an accuracy of 50.6% and a balanced accuracy of 44.1%.

For our LSTM network (Network A), we obtained an accuracy of 78.7% and a balanced accuracy of 75.8%. For CNN-LSTM (Network B), we obtained an accuracy of 75.6% and a balanced accuracy of 72.6%. For CNN-LSTM with GloVe (Network C), we obtained an accuracy of 74.6% and a balanced accuracy of 71.6%.

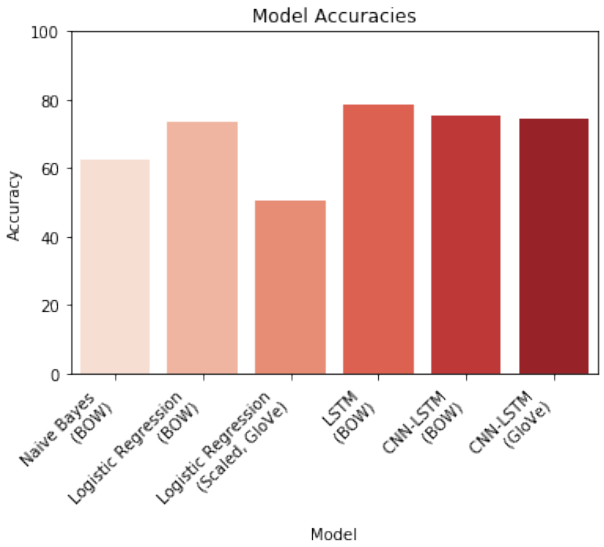

Figure 2: Accuracies for each model.

5.2 Model Comparisons

The best baseline model was logistic regression with BOW, which achieved an accuracy of 73.5%. This model performs rather comparably to all three LSTM networks in terms of classification accuracy. However, we were notably able to increase our balanced accuracy more using LSTM. The difference between the balanced accuracy and accuracy decreased from the logistic regression with BOW model to the LSTM models. Specifically, the difference between the balanced accuracy and accuracy for logistic regression with BOW was 4.6% (which was the lowest difference among the baseline models), while all of the LSTM models had a difference of 3% or less. This means that logistic regression with BOW takes more advantage of the imbalance in examples in the dataset in comparison to the LSTM models.

As seen in Table 2, the first LSTM network (Network A) performed the best in terms of accuracy, while the CNN-LSTM (Network B) and CNN-LSTM-GloVe (Network C) performed marginally better than our baseline logistic regression with BOW. The marginal increase in accuracies

[5] Note: This is different from how word embeddings are used in our logistic regression model, which averages over all the word vectors for each input $\mathbf{x_i}$; in our LSTM model, we do not average the vectors and instead leave them.

[6] Although we trained our models with other hyperparameters as well, we only report results for this particular setting of parameters, which achieved the best results.

[7] For more details, see `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html`.

Metric	Naive Bayes	Logistic Regression (Bag of Words)	Logistic Regression (Scaled GloVe)
Accuracy	62.6%	73.5%	50.6%
Balanced Accuracy	47.2%	68.9%	44.1%

Table 1: Accuracy and balanced accuracy scores for baseline models.

in the latter two models could be because we cut all of the examples at the median length. While this made training and testing our models computationally feasible, the accuracies of these models may have suffered because we are ignoring a significant portion of our data, whereas our baseline logistic regression trained on the entire dataset.

5.3 Error Analysis

For the baseline models, we expected the logistic regression classifiers to perform better than naive Bayes, because naive Bayes makes strong assumptions about independence in the data that we do not necessarily believe to be true. This is somewhat supported by our results, as logistic regression with BOW did perform significantly better than naive Bayes. However, we expected word embeddings to perform better than a BOW model, which is not reflected in our results. This may be due to the fact that averaging the word vectors in our embeddings loses much of the additional context that we were seeking to acquire.

The three LSTM models appear to have similar trends with respect to their misclassifications. Some of these may be due to the uneven distribution of the dataset, while others could be due to the extreme similarities between two different genres.

For the case of uneven distribution, we can see from the confusion matrices in Figure 4 (Appendix B) that we frequently predict *video game* (37), *profession* (26), and *health* (18) for our data. These three genres also occur frequently in the dataset, comprising 9.9%, 5.5%, and 5.7% of the dataset, respectively. It is not unlikely that our models frequently (and incorrectly) guess these genres, simply because these genres are over-represented in the dataset. In other words, if a model is unsure about a particular example, it may choose to classify the example under a genre that occurs frequently in the dataset, since this would maximize the probability that the classification is correct.

For the case of genre similarity, we can observe hotspots in the confusion matrices that are not along the diagonal. We often confuse *board game* (6) and *rpg* (29), *programming* (27) and *software* (32), and *drugs* (11) and *health* (18). In these three cases, the mispredictions make sense, as the genres are closely related in meaning and content. For instance, board games can also be role-playing games (rpg) and vice versa. Both software and programming are related to utilizing code to provide instructions to tell a computer how to perform a task. Drugs are prescribed by doctors to help people stay healthy. It may be difficult even for a human to correctly classify these self-posts; the classification task becomes even more difficult for models.

We found that the five genres with the worst performance for all of the LSTM models were *social group*, *advice/question*, *writing/stories*, *meta*, and *stem*; the accuracy for these genres were generally less than 65%. Together, they comprise only 7.8% of the dataset. Many of these genres are inherently broad and open-ended, making it difficult to determine words that may indicate a self-post within that genre. For example, *advice/question* could be related to nearly every other category, since it is not limited to a particular subject. Likewise, *writing/stories* can be about any topic that a writer chooses. The genres *social group*, *stem*, and *meta* are equally as broad, as social groups can be centered around a variety of topics, stem covers a variety of science, technology, engineering, and mathematics topics, and meta is a genre dedicated to all things specific to Reddit (e.g. `/r/OutOfTheLoop`, where people make self-posts about things happening on Reddit, which can be about any topic that could easily be classified under another genre).

Nevertheless, there are certain genres with a small number of examples that manage to perform fairly well, namely *rpg*, *movies*, and *travel*. These genres managed to yield a 70-77% accuracy despite having only 7,000, 7,000, and 5,000 examples, respectively (in a dataset of over one million examples). These genres in particular likely benefit from having certain key words that easily distinguish them from other genres. For example, the name of a role-playing

Metric	LSTM (Bag of Words)	CNN-LSTM (Bag of Words)	CNN-LSTM (GloVe)
Accuracy	78.7%	75.6%	74.6%
Balanced Accuracy	75.8%	72.6%	71.6%

Table 2: Accuracy and balanced accuracy scores for LSTM networks.

game would be a strong indicator of *rpg*, while the name of a movie would be a strong indicator of *movies*. Moreover, the name of a popular tourist location would be a strong indicator of *travel*. Of course, this might have led to mispredictions with similar genres that may occasionally use these key words, as we saw earlier with *rpg* and *board game*.

6 Future Work

As mentioned previously, our data is imbalanced in terms of the number of examples per genre. We did not account for this imbalance in our current models, so a future step may be to try balancing the dataset by removing some of the examples from over-represented genres. From the differences in balanced accuracies between our baseline and LSTM models, we know that this imbalance does factor into the performance of our models, so we are interested to see if balancing the dataset could further improve the performance of our LSTM models.

Due to limitations on computation power and time, we trained our models on large batch sizes and over relatively few epochs. In addition, we limited the maximum length of our self-posts in our LSTM networks to the medium length across the entire dataset. This means we are effectively ignoring a significant portion of our training data. In the future, we may tune these hyperparameters and train longer over more data. We may also use a different pretrained word embedding with a larger vocabulary. Our GloVe had a vocabulary size of 400,000, which is only half of the almost 800,000 unique words in our dataset. Another possibility we have not yet explored is how different word embeddings (e.g., Word2vec, GloVe, FastText) may affect our classifications.

7 Code

The project code can be found at `https://github.com/ulloaluis/CS221-Read-it-on-Reddit`.

References

[1] Mike Swarbrick Jones. The reddit self-post classification task (rspct) : a highly multiclass dataset for text classification.

[2] Angel Fiallos and Karina Jimenes. Using reddit data for multi-label text classification of twitter users interests. *ICEDEG 2019 Conference*, 2019.

[3] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[5] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.

[6] The reddit self-post classification task. `https://www.kaggle.com/mswarbrickjones/reddit-selfposts`. Accessed: October 2019.

[7] Amit Mandelbaum and Adi Shalev. Word embeddings and their use in sentence classification tasks. *arXiv preprint arXiv:1610.08229v1*, 2016.

[8] Aaron Defazio, Francis Bach, and Lacoste-Julien Simon. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *arXiv preprint arXiv:1407.0202*, 2014.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

A Performance by Genre

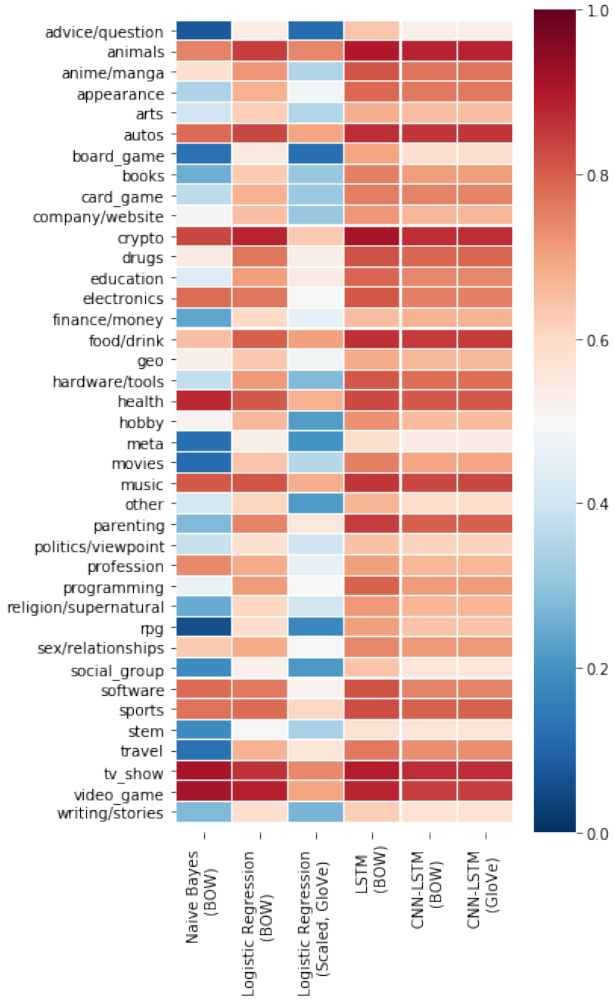

Figure 3: Accuracies by genre for each model.

B Confusion Matrices

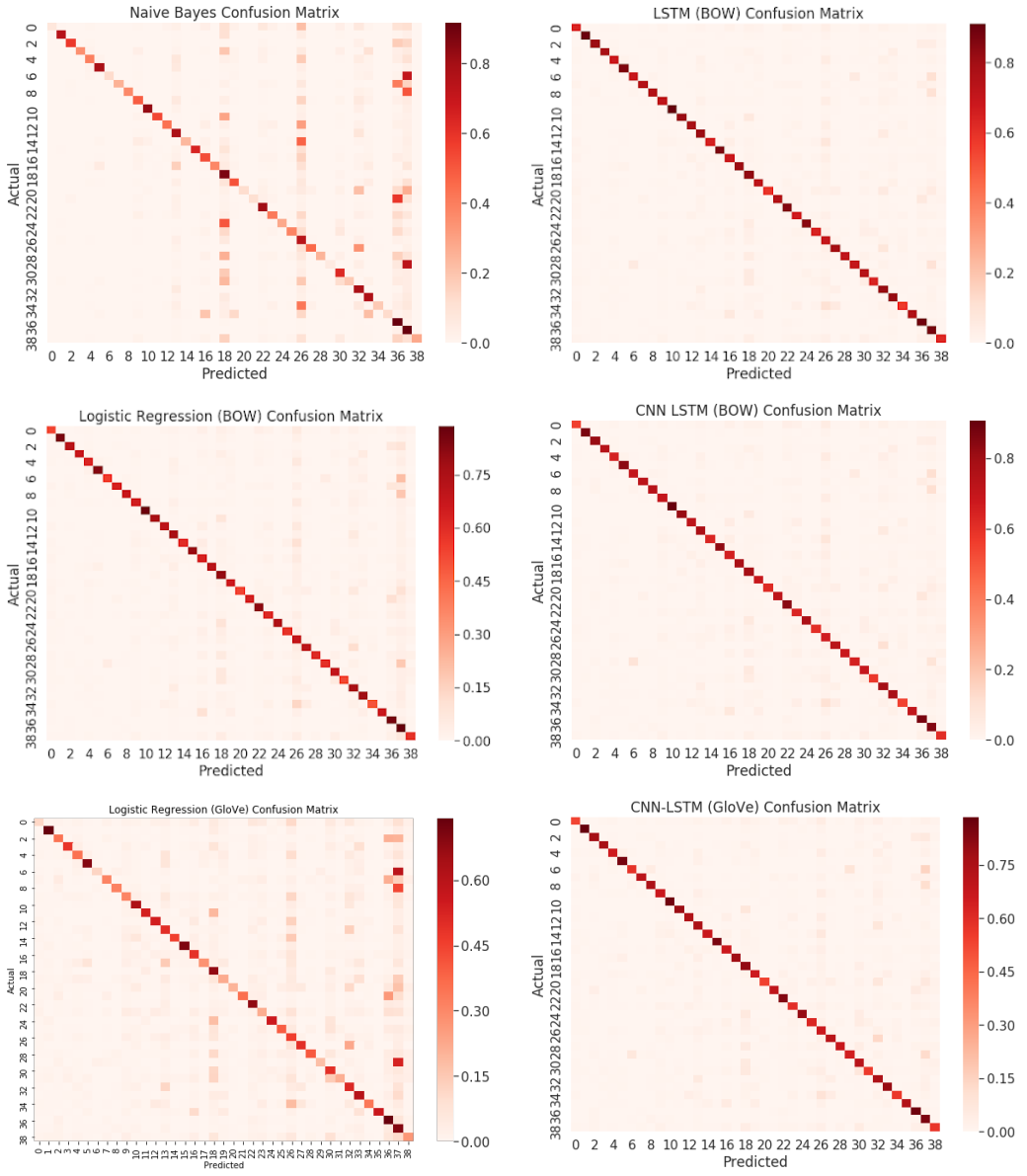

Figure 4: Confusion matrices for each model.