**1. Introduction**

Group Members

Julia Reisler, Zafir Abou-Zamzam, and Michelle Zhao

Team Name

flickr.com/photos/yisongyue

Github Repository

https://github.com/reislerjul/KaggleCompetition

Division of Labor

* Julia: Wrote helper functions to calculate classification error, write predictions to a file, and use term frequency-inverse document frequency to transform the feature data. Implemented a variety of models, including logistic regression, naïve Bayes, linear classifier with stochastic gradient descent training, Bernoulli naïve Bayes, linear support vector machine, ridge classification, and K neighbors classification. Built off of Michelle and Zafir’s neural networks to create a deeper network using a hyperbolic tangent as the activation function. Developed different methods of blending the models together to use predictions from subsets of different models. Split the data into training and validation sets and used these to optimize model parameters.
* Zafir: Worked primarily on implementing neural network models. Tested models with different activation functions including ReLU, sigmoid, leaky ReLU, and hyperbolic tangent. Used Julia’s data partitioning to train and validate on different subsets of input data. Experimented with different network width and depth, and added regularization and normalization to determine an optimal neural network. Provided analysis of different models to determine models which provided the best training and validation errors, and analyzed submissions to determine most overfitted models.
* Michelle: Wrote code to extract data from the data files and organize them into corresponding testing and training sets. Wrote code for training neural networks on the data, and testing different hyperparameters, including varying the dropout, activation functions, the depth and width of the neural network to try to find the best combination of hyperparameters. Experimented on training the data using naive bayes, gaussian and multinomial naive bayes, recurrent neural networks with LSTM in tflearn. Worked on blending the neural network into the existing blend to improve the model.

**2. Overview**

Models and Techniques tried

* Models: logistic regression, multinomial naïve Bayes, linear classifier with stochastic gradient descent training, Bernoulli naïve Bayes, linear support vector machine, ridge classification, K neighbors classification, neural network, recurrent neural network.
* Techniques: TF-IDF, blending multiple models.

Work Timeline

* Weekend before competition: met to discuss general methods and approaches to the problem.
* Week of competition: tested and submitted models individually in addition to meeting multiple times to work together on models.

**3. Approach**

Data processing and manipulation

* We used the scikit-learn TfidfVectorizer to convert the features into TF-IDF features. To do this, we turned each data point into a string using the words present in the data point, and then used the TfidfVectorizer method. TF-IDF is a way of weighting words based on how important they are in a document and set of documents. The weight for a word increases as the prevalence of the word in a document increases, but decreases as the prevalence of the word in the set of documents increases. We noticed that using TF-IDF generally decreased the validation error from our models. For example, with basic linear regression, the validation error decreases from around 14.725% without TF-IDF to about 14.425%.

Details of models and techniques

* Details of each model: In the end, our best score resulted from a blending of a logistic model, linear classifier with stochastic gradient descent training, multinomial naïve Bayes, and a neural network. We used scikit-learn’s implementation from the first three and used keras for the neural network. After doing research about the best models to use with bag of words data, we decided on these four. Logistic regression relates the probability of an input belonging to a particular group, is relatively simple, and can work well on sparse matrices. Assuming that the data is nearly linearly separable, we thought that a linear classifier would work well due to its simplicity. Naïve Bayes assumes each feature is completely independent of other features and uses maximum likelihood estimates to classify data. As for neural networks, neural networks learn and model non-linear and complex relationships in the training data using a forward learning and back propagation algorithm. We specify in the neural network that the data must be classified into a 2-class output (0 or 1), and by varying how many layers deep the network is and how many nodes there are per layer, we can find the optimal combination that doesn’t underfit or overfit the data. We found that a neural network with 2 hidden layers of 200 nodes each with a dropout of 0.1 and activation using sigmoid was optimal for our approach.
* Blending: Noting that each model uses a different method to classify data differently, we thought that blending the models would reduce overfitting from each model by forming a consensus through different parameters used to classify the data. We also used scikit-learn’s notions of confidence intervals for the linear and logistic models. Noticing that when these two models had high confidence in their classification, the classification error was only around 5%, we decided to only use all four models to classify in cases when the confidence scores for both logistic and linear regression had a magnitude less than 1. In those cases, we took a majority vote for the classification, giving the logistic model two votes since it was the best model.
* Testing: To test our data, we split the given set into 80% training and 20% validation. While this was done to prevent overfitting, we chose parameters that optimized the classification on the validation set, which actually caused a little overfitting to the validation set.

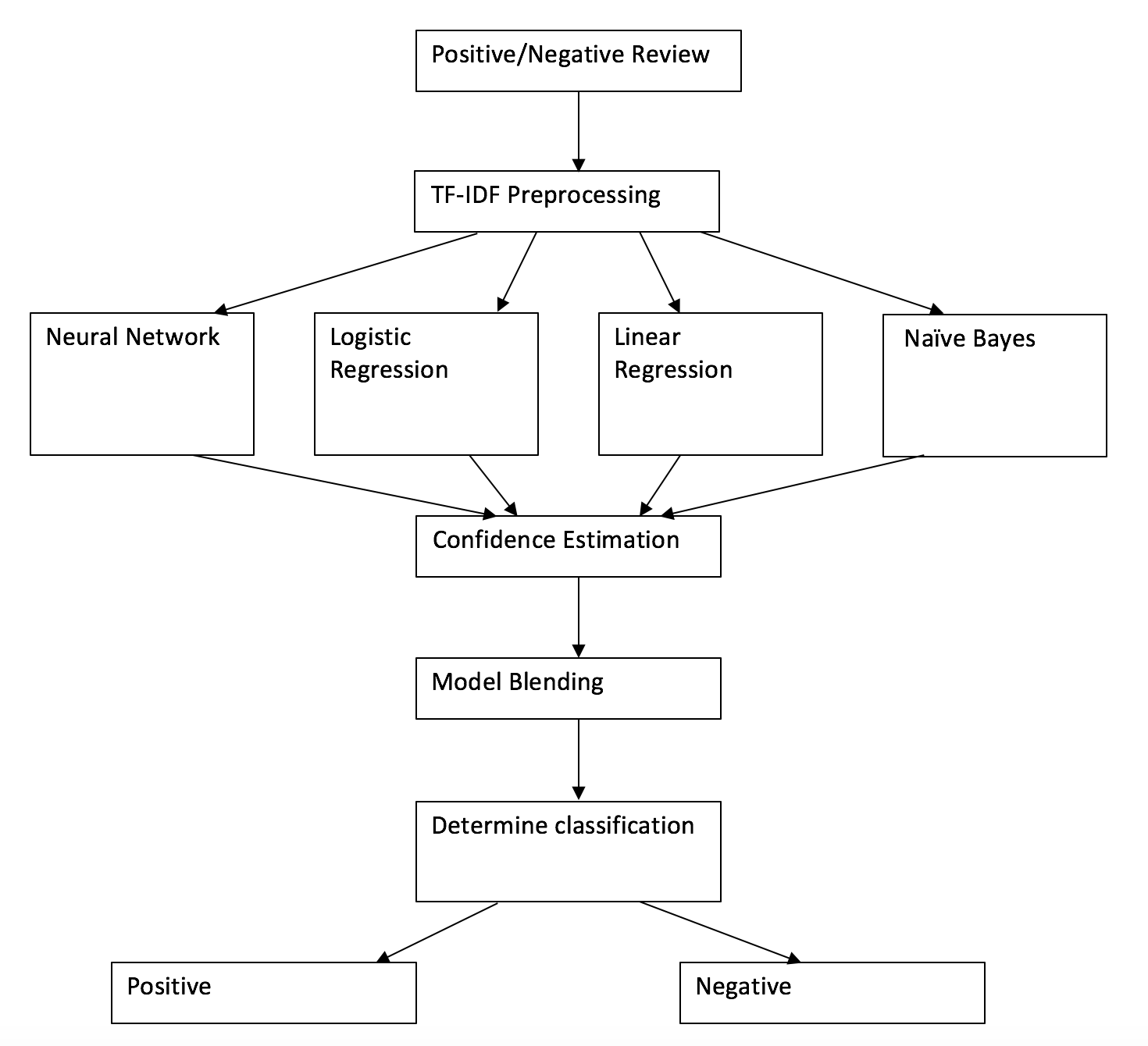


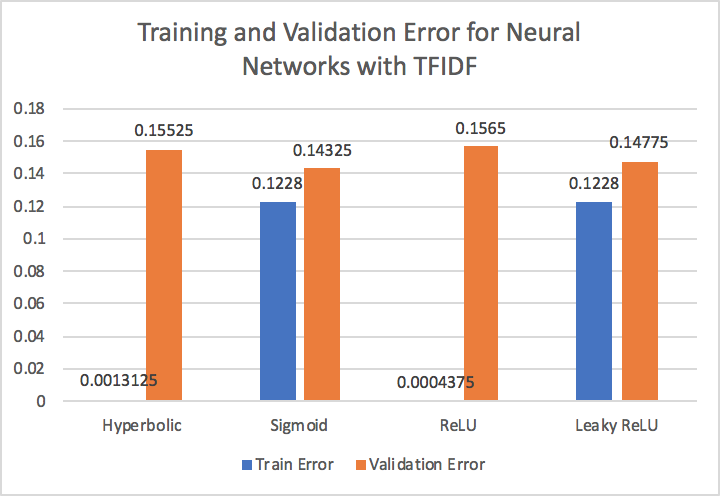
Diagram 1. Outline of Approach for Training Models

**4. Model Selection**

* Validation and Test: To test our data, we split the given set into 80% training and 20% validation. While this was done to prevent overfitting, we chose parameters that optimized the classification on the validation set, which actually caused overfitting to the validation set. Ideally, the best score on the validation set would have helped us figure out which model would perform best on the test set, but in reality, overfitting tended to skew these observations slightly.
* Scoring: We scored our models based on classification error from the validation set. On a particular partition of the dataset, the logistic regression scored 14.775% classification error, the linear regression scored 15.425%, naïve Bayes scored 16.925%, the neural network scored 15.85%, and the blended model scored 14.25%. These scores varied slightly on different partitions of the dataset, but remained in the same range. As a side note, lower scores did not necessarily give the best accuracy on the test set; they were more reflective of how well they could classify the validation set. However, these scores were the basis of which models we chose to submit.

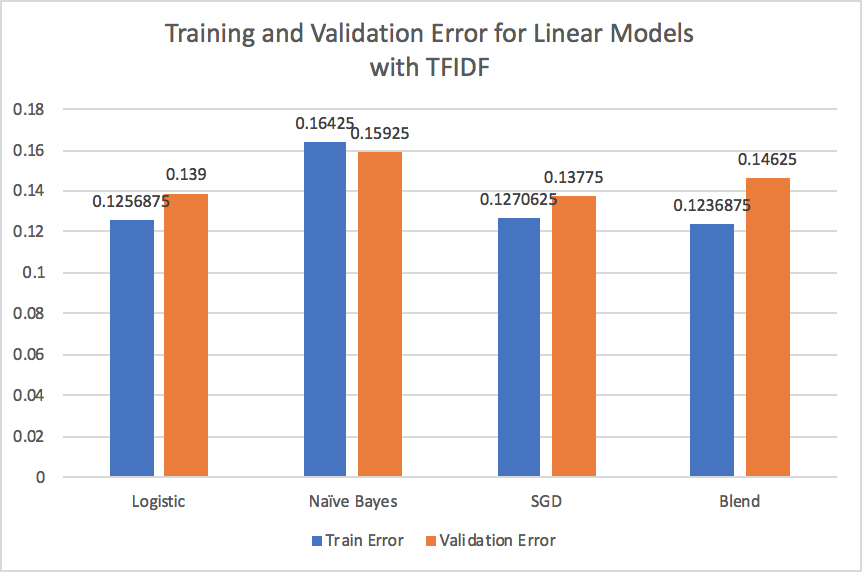
**5. Charts**

Figure 1.

.

The Hyperbolic and ReLU activation functions greatly overfit the training data, resulting in extremely low training error. Sigmoid and Leaky ReLU had better overall performance. With these models, neural networks tended to overfit the training and validation data, so actual submissions had lower scores.

Figure 2.



Logistic Regression had the lowest training error and SGD had the lowest testing error in terms of individual models. However, the blended model performed better for more generalized data, and performed best with our submission.

**6. Conclusion**

* Discoveries: In general, more advanced models tended to be more likely to overfit on the data. We found that the four best methods for training our model were logistic regression, linear regression, Naive Bayes, and a neural network. When combined in a blend of these 4 models, the blend had the best performance on the training and validation data over any individual method. We learned that many of our models that performed better on the public leaderboard performed worse or poorly on the private leaderboard, indicating that overfitting is a very present issue. Many of our top submissions on the public leaderboard did not perform as well on the private one, and our model that scored best on the private leaderboard was well below the public one. Additionally, we learned that sometimes a simpler model is better than a highly complex model. For example, excluding the blend, our logistic regression model had the best individual performance, beating out our best neural network by about 0.02% on the private leaderboard. Finally, we learned that data preprocessing is crucial to learning a good model on the training data, especially for text-based sentiment analysis.
* Challenges: Overfitting is a real issue when training and testing on different data. For many of our models, when training and validating, we had scores ranging from 10 - 12% for the validation set, sometimes even as low as 8%. However, many of these models, when we did submit them, tended to do poorly on the public and private leaderboards, averaging around 15.5 - 16% error. Part of the reason for this issue was that because we were choosing the models with the best validation error to submit, we were not thoroughly analyzing the performance of these models. Although the better models tended to have low validation errors, a challenge that arose was trying to determine if this low error was due to the accuracy of our model, or due to the overfitting of the data. In some cases it was easier to tell. For example, we attempted to create a neural network with batch size of 2000. This outperformed all our other neural network models greatly, having a validation error of 8.125%, while other models had errors of around 13.1 – 14.5%. However, larger batch sizes lead to poorer generalization of the model, which we encountered when submitting this model, where the error was around 15.5%
* Remarks: Naive Bayes is commonly regarded as one of the most effective methods of sentiment analysis. Interestingly, when we tested Naive Bayes, it did not perform better than the other models. Different data preprocessing may have improved its performance, as well as slightly different implementation of the model. Throughout this project, there were other combinations of blending models that we did not have time to try, and it would be interesting to see that if with different blending of our existing models, we would be able to improve our test error. We would also try to improve our methods of normalization, boosting, and preprocessing to help better our model score, and understanding of how these are optimized.