Quarantined Quants: Yelp Fake Review Detection

Amanda Kuznecov (anr431), Sophia Tsilerides (smt570), Ilana Weinstein (igw212)
DS-GA 1003 - New York University
GitHub Repo

1. INTRODUCTION

Fake opinions mislead consumers, preventing them from making good decisions, thus negatively impacting organizations and businesses. We were provided with a Yelp dataset of genuine and fake reviews written about New York City restaurants. Each review recorded the user id, product id, rating, timestamp, the text of the review itself, and a label identifying whether the review was fake (1) or genuine (0). The problem is a binary classification problem involving the handling of imbalanced classes, with fake reviews only accounting for 10% of the total number of reviews. The purpose of this problem was to correctly determine whether a restaurant review from the Yelp dataset is genuine or fake. Thus, our model had to solve a binary classification task where the output was a score, with higher scores indicating a greater likelihood of the review being fake. We explored a few modeling approaches, ultimately landing on TensorFlow's Keras model with 3 layers, which yielded the highest evaluation metrics on the validation set.

2. APPROACH

Keras is a high level neural network library that is built on TensorFlow and other frameworks, which makes it a good place to start for beginners in deep learning like ourselves [1]. Neural networks were appropriate for this problem because of their versatile architectures and ease of incorporating inductive bias for different tasks [2]. Additionally, their structure is not limited by strict data assumptions of normality, linearity, and variable independence like other algorithms. Because of this they can capture many kinds of relationships easier than other models. It uses the typical formula to make predictions in neural networks: $h_i(x) = v^t{}_i(x)$ which takes input times weights, adds a bias, and finally runs it through an activation function in multiple layers. It is then trained with the backpropagation algorithm and optimized with the objective function and loss specified.

Since this problem requires a single output, Keras' Sequential model was appropriate. The model is made up of multiple dense and dropout layers. Dense layers return the results of activation functions called on the inputs and their weights. Weights can be changed and set manually, which we experimented with. Dropout layers control regularization by removing a random selection of a fixed number of the units in a network layer for a single gradient step [3]. An example of the algorithm's process can be seen in Figure 1. The inputs engineered by the original data are passed through two hidden layers; a design layer with ReLU activation and a dropout layer, before being passed through the output layer which uses a sigmoid transformation to return a score.

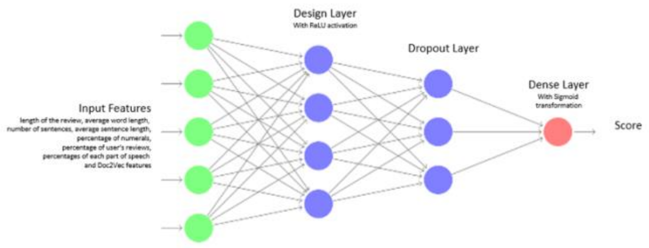

Design Layer
With ReLU activation
Dropout Layer
Dense Layer
With Sigmoid transformation
Input Features
length of the review, average word length, number of sentences, average sentence length, percentage of numerals, percentage of user's reviews, percentages of each part of speech and Doc2Vec features
Score

Figure 1: Neural network algorithm

3. EXPERIMENTS

3.1 Feature Extraction

We began by extracting potential predictive features from the dataset, in part to preserve the original review before preprocessing. These structural features included the length of the review, average word length, number of sentences, average sentence length, percentage of numerals, percentage of total reviews written by a particular user, and percentages of each part of speech. This added 41 features to the dataset. Then, we preprocessed the data using the nltk library, which converted all the words in the review to lower case, removed punctuation and numerals from each word, and filtered out stop words. We also tried to featurize the date column into the day of the week, month, and year, however, this had no predictive power and was removed from consideration. Lastly, the length column covered a big range, so we converted it to log-space.

The distribution of some of these features is shown in Figure 2. Length of fake reviews is smaller than genuine reviews since the reviewer does not have much knowledge about the restaurant and tries to achieve malicious objectives in as few words as possible. Additionally, genuine reviews tend to have more numerals. Lastly, users who provide more reviews tend to be classified as genuine, while rating does not seem to distinguish between fake and genuine reviews.

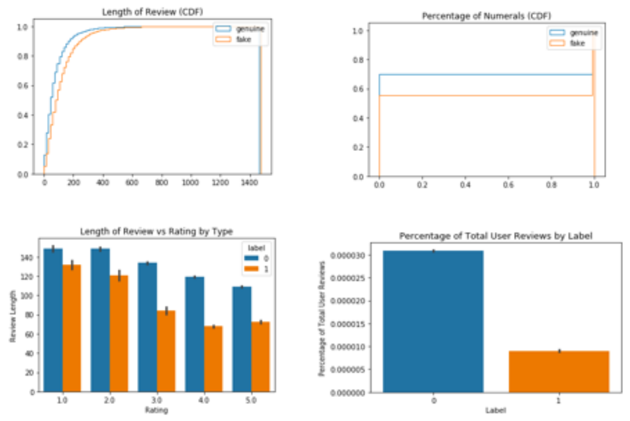


Figure 2: Exploratory analysis on the dataset

We hypothesized that the most impactful task for prediction would be data transformation, in particular, vectorizing reviews that a model could clearly partition. We also hypothesized that our problem would be nonlinear. To test these hypotheses, vectorization was performed on the review text in order to extract useful features from our dataset. We used three different vectorizers for baseline experimentation and two different models.

TF-IDF was used because it captures how important a word is to a document in a corpus by calculating the frequency of a word in a given document offset by its frequency in the overall corpus. Doc2Vec, an extension of Word2Vec, was used to vectorize each review to represent the concept of the review [4]. Doc2Vec was applied for words appearing to our problem, only considering words appearing more than 50 times, and using a vector size of 100 for each review. Latent Dirichlet Allocation (LDA), which has a very similar notion to Doc2Vec, was also implemented. LDA performs topic modeling based on co-occurrences of words within topics [5]. We initially tried to pre-set LDA with 100 topics and selected the top 20% of topics to be assigned to feature vectors. As a comparison, we also tried LDA with 500 topics and selected 100 topics to be assigned to feature vectors, however, the process was three times slower and achieved the same baseline results. Therefore, the smaller topic size was used, because the larger topic size did not improve performance.

3.2 *Class Imbalance*

We addressed the imbalanced classes with oversampling and Synthetic Minority Oversampling Technique (SMOTE). Upsampling was chosen over downsampling since the downsampling technique is prevalent for datasets with instances in the millions. We theorized that the handling of the imbalance through these methods will not improve evaluation and will be addressed through modeling. When comparing the two, SMOTE prevails over Upsampling in comparison to other baseline experiments

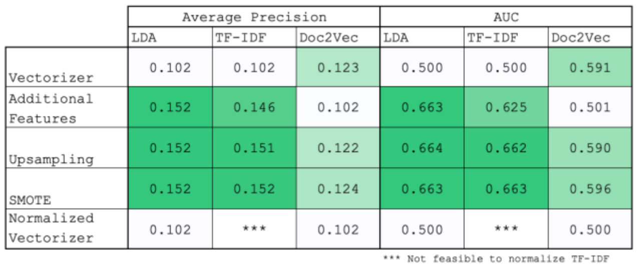
3.3 *Baseline Modeling*

We used Naive Bayes for baseline modeling. Naive Bayes models are extremely fast and simple, making them suitable for high-dimensional datasets like ours. They also have few tunable parameters so they work well for baseline classification problems. The assumption of Naive Bayes models is that the features are conditionally independent given the label, which is not exactly true in text.

Bayesian classification searches for the probability of a label given some observed features, that is P(L|features) [6]. This provides a straightforward probabilistic prediction that is easily interpretable and addresses our hypothesis directly. Depending on our vectorizer, we used either Multinomial Naive Bayes or Gaussian Naive Bayes.

Since this is an outlier detection problem, appropriate evaluation metrics are AUC and Average Precision. One of the advantages of AUC is that it measures the classification accuracy regardless of how many examples of each class there are. This is because it evaluates the True Positive Rate against the False Positive Rate, which is the ratio of misclassified negative cases and the ratio of correctly classified positive cases. Similarly, Precision is calculated as $\frac{TP}{TP+FP}$. It is notable that even if accuracy is deceivingly high, AP will be low because the number of false positives will grow and cause the denominator to become larger.

The results of our Naive Bayes baselines are shown in Table 1. Since it was not feasible to normalize TF-IDF, we took it out of consideration. We continued with LDA and Doc2Vec datasets with additional features and resampling since they produced comparable results.



	Average Precision			AUC		
	LDA	TF-IDF	Doc2Vec	LDA	TF-IDF	Doc2Vec
Vectorizer	0.102	0.102	0.123	0.500	0.500	0.591
Additional Features	0.152	0.146	0.102	0.663	0.625	0.501
Upsampling	0.152	0.151	0.122	0.664	0.662	0.590
SMOTE	0.152	0.152	0.124	0.663	0.663	0.596
Normalized Vectorizer	0.102	***	0.102	0.500	***	0.500

*** Not feasible to normalize TF-IDF

Table 1: Naive Bayes baseline results

We also ran the datasets through a simple Random Forest with 100 trees. This model was able to classify some fake reviews, giving a higher than 0 precision. Since tuning a Random Forest would only increase our results by 10%, we concluded that our problem was non-linear and began exploring more sophisticated non-linear models with the baseline knowledge received from each vectorizer using Naive Bayes and Random Forest.

3.4 BERT

Our next attempt to improve our baseline was with BERT (Bidirectional Encoder Representations from Transformers). BERT's state of the art performance on many NLP tasks is the motivation behind this experiment. We utilized the Huggingface Transformers library to drive BERT training. We chose to fine-tune the pre-trained "bert-base-uncased" model (12-layer, 768-hidden, 12-heads, 110M parameters, trained on lower-cased English text) based on reputation with sequence classification [7]. We leveraged Chris McCormick's notes on BERT fine-tuning with PyTorch and hyper-parameter tuned on epochs and maximum sequence length [8]. Due to computation limitations, we were unable to tune batch and size processes to the maximum sequence length of 512. The highest validation accuracy obtained was AUC of 0.545 and AP of 0.131 on 256 sequence length with a total training time of six hours. To further improve BERT's results, we downsampled the 'genuine' class. This new approach resulted in an AUC of 0.671 and an AP of 0.157. When trained on a maximum sequence length of 512, we achieved an AUC of 0.681 and an AP of 0.162. Most reviews had lengths greater than 512, which were truncated. Due to this computation constraint, we decided to focus on the next experiment.

3.5 Neural Networks

Our most successful model in experimentation was a simple neural network with TensorFlow's Keras model with 3 layers: a dense layer with ReLU activation, a dropout layer, and an output layer with the sigmoid function to return a probability prediction score between zero and one to determine if a review is genuine or fake. When tuning the model, we added dense and regularization layers, however they performed the same or worse on the validation set. For the Dropout layer, the more units dropped out, the stronger the regularization [9]. We set the unit to drop as 0.5, which research showed to be optimal for a wide range of networks and tasks [10]. In the dense layer, 8, 16, 32, and 64 units were tested with 64 units evaluating to be optimal. We selected ReLU as the activation function for the dense layer because it is fast to compute and doesn't have flat regions where gradient descent can get stuck since neural networks are not convex.

The model was compiled with an optimizer, a loss function, and different metrics. The metrics used were accuracy, precision, recall, and AUC, all typical metrics from classification. We tried two different optimizers; Adam and RMSprop. Although both objectives performed similarly, Adam optimizer produced the best results because it is unaffected by the scaling change. It is also very popular to use for imbalanced problems. Lastly, binary cross-entropy was the loss chosen because it is typical for binary classification problems.

To begin experimentation with Keras, we started with calculating the output layer's bias using $log_e(\frac{fake}{genuine})$ this way the model doesn't need to spend the first few epochs learning that fake reviews are unlikely. Doing so significantly improved lowered validation loss as seen in Figure 3. These initial weights were saved and utilized throughout the modeling.

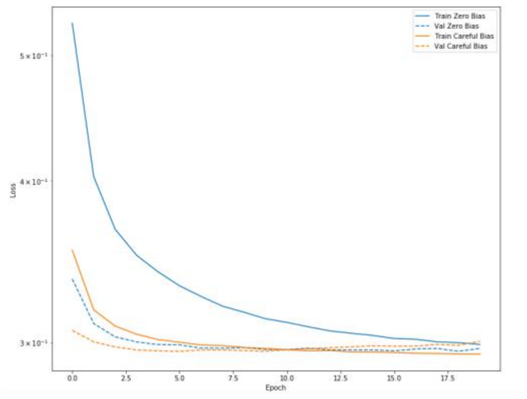


Figure 3: Keras model loss with and without bias

Since we have so few fake reviews in the dataset, in order to better represent these reviews, Keras allowed us to place a heavier weight on them so as to give more attention to the under-represented class. A weight of 0.56 was applied to genuine reviews, and a weight of 4.86 was applied to fake reviews. This was determined by $\frac{(total\ number\ of\ reviews)/2}{number\ of\ reviews\ in\ class\ X}$. Total reviews were scaled by ½ to keep the loss to a similar magnitude. Modeling with weights resulted in the evaluation in Figure 4.

Overfitting is a significant concern when there is imbalanced data, so training history was compared to validation history. It is important to note that comparing performance on our training and validation sets, results are very consistent and do not show signs of overfitting. Lastly, we experimented with upsampling the minority class as we did in baseline testing. With upsampling, it was important to perform Early Stopping on the model to prevent overfitting. These experiments were performed on the LDA vectorized dataset and the Doc2Vec vectorized dataset. The Doc2Vec data produced the best results and they are shown in Figure 5. Evaluation metrics performed across the vectorizers are compared in Table 2.

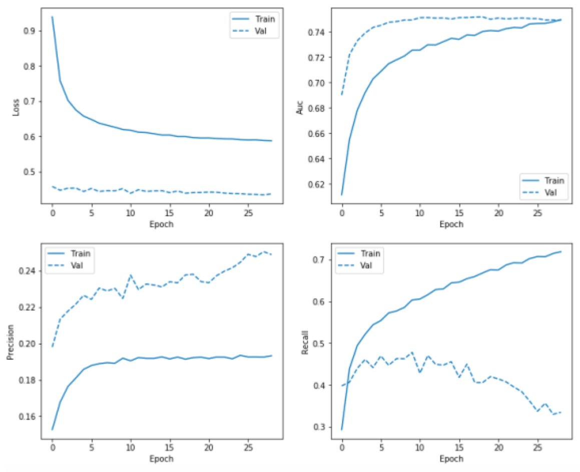


Figure 4: Keras model evaluation with class weights

	Average Precision		AUC	
	LDA	Doc2Vec	LDA	Doc2Vec
Baseline	0.228	0.232	0.746	0.750
Weighted	0.228	0.231	0.750	0.752
Resample	0.224	0.228	0.745	0.748

Table 2: Keras model evaluation results

Thus, we were able to confirm that a weighted Keras model using the Doc2Vec vectorized dataset performed the best. We conducted tuning on hyperparameters, including batch size, number of epochs, and the optimizer's learning rate. Batch size is the number of examples used to train the model in tandem. It was desirable for this to be large enough to ensure every batch has a few fake reviews to learn from. Based on this experimentation, it was notable that small bath sizes did not perform well. A batch size of 100,000 examples was optimum. The number of epochs is the number of times the model goes through the entire dataset. We tested for epochs ranging from 100 to 500. This parameter did not make any considerable impact on model performance, so we left it at 100 for efficiency. Lastly, the Adam optimizer's learning rate was tested for values ranging from 1e-6 to 2.8. For three layers, the optimal learning rate was 0.1. For five layers, the optimal learning rate was 0.001. This makes sense as we deserted the five-layer neural network in favor of the three-layer because of better evaluation results. The smaller learning rate means that it took the model longer to learn with smaller steps.

4. DISCUSSION

This paper provided an overview of the methodology used to optimize the effectiveness of our fake review detection model and its performance on a hidden test set. During our study, we chose to extract additional features from the text which did not significantly alter our results. This likely means that detecting a fake review is not highly dependent on these additional features. For future modeling, it could be

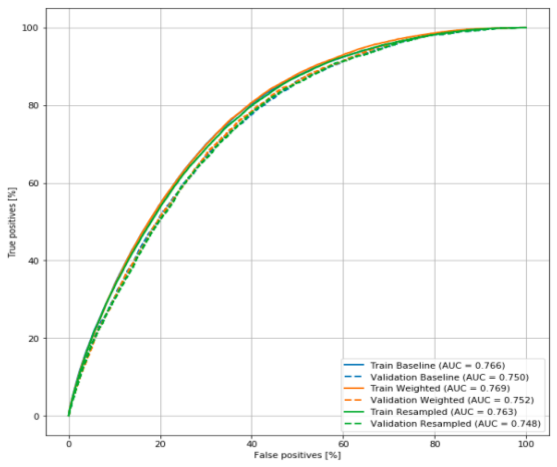


Figure 5: Keras model comparison of different methods

beneficial to obtain a more all-encompassing dataset that includes information on the users such as their geo-locations, account activity, user interactions, if they have ever uploaded a photo as part of their review, and if anyone found their review helpful/funny/cool.

In order to properly represent the text data within the review attribute, it was necessary to perform vectorization to get our data in a form appropriate for modeling. As expected, vectorization improved our model through the use of Doc2Vec and LDA, which are very similar methods that summarize the concept of a review as a single vector. Although our model proved to perform best using these two vectorizers, they did not produce optimal results. Both methods are unsupervised algorithms that are difficult to control. Vectors produced by LDA for each review were based on the algorithm's best guess about the likelihood of a given sample belonging to each of the 20 topics it learned. In a similar fashion, vectors generated by Doc2Vec were applied using the algorithm's best judgment to assign concepts to each review based on other reviews within the corpus. Some shortcomings may be the fact that there is not much similarity across the reviews, thus making it harder for LDA and Doc2Vec to apply appropriate weights. More fine-tuned vectorizers could have possibly given better results if for example the number of topics was increased, the minimum required count of words was decreased, or the vector size was increased. However, due to limitations regarding efficiency, implementing these changes would have been time consuming due to long training times, and without knowing to expect improvements, the additional tuning could have been unnecessary. These additional experiments could be performed as part of future work.

Class imbalance handling was an important task due to the small number of fake reviews relative to the number of genuine reviews. As predicted, these techniques did not improve model performance, which we suspect is due to the nature of the data. Upon performing some research, the handling of the imbalance can simply be the choice in model, which is accurate with our problem [11].

Our hypothesis that this prediction task was not linear was confirmed through baseline modeling, and therefore called for other methods including BERT and TensorFlow's Keras model. BERT did not work as well as we suspected it to considering its high rank in common NLP tasks. We believe that we could have obtained higher results if we incorporated some of our vectorization techniques and had the bandwidth to further hyper-parameter tune and experiment with different pre-trained models. The Keras model performed much better than BERT, however, results were not overly favorable as precision is quite low. Some future work can be to ensemble models by incorporating a BERT layer into the Keras model. Additionally, further fine-tuning of the neural network structure by creating a larger network may improve model performance. Both tasks may be conducted with the hope of improving model generalizability and evaluation on the test set.

REFERENCES

[1] "Classification on Imbalanced Data: TensorFlow Core." TensorFlow, www.tensorflow.org/tutorials/structured_data/imbalanced_data.

[2] He, He. Machine Learning Module 14 - Neural Networks. May 2020, https://github.com/cp71/DS-GA-1003-SPRING-2020-PUBLIC/blob/master/lecture/lec14/lec14-neural-networks.pdf. Presentation.

[3] "Machine Learning Glossary | Google Developers." Google, Google, 2019, developers.google.com/machine-learning/glossary/#dropout_regularization.

[4] Shperber, Gidi. "A Gentle Introduction to Doc2Vec." Medium, Wisio, 5 Nov. 2019, medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e.

[5] Kelechava, Marc. "Using LDA Topic Models as a Classification Model Input." Medium, Towards Data Science, 13 Apr. 2020, towardsdatascience.com/unsupervised-nlp-topic-models-as-a-supervised-learning-input-cf8ee9e5cf28.

[6] "In Depth: Naive Bayes Classification." Python Data Science Handbook: Essential Tools for Working with Data, by Jake VanderPlas, O'Reilly, 2017.

[7] "Pretrained Models¶." Pretrained Models - Pytorch-Transformers 1.0.0 Documentation, huggingface.co/transformers/v1.1.0/pretrained_models.html.

[8] "BERT Fine-Tuning Tutorial with PyTorch." BERT Fine-Tuning Tutorial with PyTorch · Chris McCormick, 22 July 2019, mccormickml.com/2019/07/22/BERT-fine-tuning/.

[9] "Machine Learning Glossary | Google Developers." Google, Google, 2019, developers.google.com/machine-learning/glossary/#dropout_regularization.

[10] Srivastava , NitishHoussaine. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." Journal of Machine Learning Research, vol. 15, 2014, http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf.

[11] Boyle, Tara. "Methods for Dealing with Imbalanced Data." Medium, Towards Data Science, 4 Feb. 2019, towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18.