Stock Price Prediction Using Twitter

Amal Duriseti
904331855
aduriseti@gmail.com

Jonathan Hurwitz
804258351
jdhurwitz@ucla.edu

Fred Xu
004255573
xuzeyuanfred@ucla.edu

Jerry Chen
804449266
hostmissile@hotmail.com

ABSTRACT

This project aims to predict buy or sell indicators for several large S&P500 companies based on Twitter data. This is a form of classification rather than regression. The first iteration of our project aimed to predict mood based on tweets, but we pivoted and leveraged learnings from the previously done sentiment analysis work in order to build our stock predictor. Several machine learning models have been tested and compared. We settled on using a feed-forward neural network, or multi-layer perceptron (MLP), to perform the predictions.

1. INTRODUCTION

The former goal was to create an aggregate mood detector and predictor by training machine learning models to perform sentiment analysis on tweets. This task implicitly includes building some method of obtaining more data, such as a data crawler. While our models showed reasonably good performance when trained on the twitter sentiment analysis training corpus[1], we realized that some sort of unsupervised learning would have to be done in order to label future data obtained via the crawler. This motivated us to explore other ways in which we could leverage our previous work but for a task with an easier means of labeling data.

1.1 Problem Definition and Formalization

Using sentiment to predict buy or sell indicators seemed like a natural progression with an easier way of labeling data. The labeling scheme we chose was to associate each tweet with the corresponding stock price on the day it was tweeted. The crawler pulls tweets and labels each tweet by doing a yahoo finance price lookup. This way, we are able to build an arbitrarily large dataset of labeled tweets in order to better train our models. There is no need for unsupervised learning such as clustering in order to assign class labels.

To formalize, the goal of the project was, given a set of tweets about a company (by stock ticker), to predict whether the stock would go up or down on the open of the next trading day by using sentiment analysis.

All code for this project can be found here: https://github.com/aduriseti/cs145_project. All of the work was done in Jupyter notebooks.

2. DATA CRAWLER

The data crawler performs tweet lookup, preprocessing, and labeling for an arbitrary number of stock tickers or query strings. It is assumed that data for each query will be pulled for each day within the specified date range.

Within the scope of this project, the queries were all stock tickers for several large companies: AAPL, AMZN, SNE, GOOG, NVDA, INTC, HPQ, MSFT, IBM, and TXN. Each of these tickers is given their own thread to perform the lookup, filtering, and labeling. The initial intent was to perform a live lookup on Yahoo finance in order to obtain stock prices for labeling, but there were issues with Yahoo finance historical data. After writing a function to pull data using BeautifulSoup, we decided it would be best to just download local csv files from Yahoo containing the data for each ticker over the desired time range (from min to max date).

For each day in the date range, the crawler queries twitter and limits the number of tweets pulled via the "max_tweets" parameter. Tweets are processed in batches corresponding to each day. Tweets for a specific ticker that were posted on a day in which there is no associated stock price are removed from the set. Duplicates are also removed. Stock price labels are 5-D and consist of the open, high, low, close, and adj close. The intent was to provide the maximum amount of information for us to use in learning algorithm evaluation. Following a successful tweet:label pairing, the crawler writes this to a file along with the date of the tweet and continues.

3. METHODS DESCRIPTION

After obtaining raw data by means of our data crawler, we took the following approach:

1. Preprocess the data via tokenization.

2. Test various models for sentiment analysis performance on a set of commonly used metrics.

3. Choose the most effective model and perform hyperparameter tuning on it.

[1] The sentiment analysis corpus contains 1.5M tweets and can be downloaded at http://thinknook.com/twitter-sentiment-analysis-training-corpus-dataset-2012-09-22/

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

4. Concurrently test exploratory designs on more complicated RNN-based models, such as the LSTM.
5. Integrate the most effective model from above steps into the stock prediction task, and use the same classification metrics as before to determine performance.

As mentioned, in our case the stock price prediction is a classification task rather than a regression task since we are simply outputting binary results: ”expected to go up tomorrow” or ”expected to go down tomorrow”.

4. SENTIMENT ANALYSIS

We trained a variety of classifiers on the Twitter sentiment analysis training corpus. These included random forest, multi-layer perceptron (MLP), and quadratic discriminant analysis. In addition, we tested the RNN-based long short-term memory (LSTM) model. SVM was avoided due to long training time.

The embedding scheme involved using word2vec and a mean aggregation method. The word2vec vectors are trained on the set of aggregated tweets created by the crawler. For each word within a tweet, we check to see if it is in the trained word2vec model. If so, the word2vec vector corresponding to that word gets added to a running sum for the tweet. If a word does not appear in the word2vec model, then it is zero padded. After all the words have been checked, this summed vector is divided element-wise by the number of words in the tweet in order to obtain a mean vector. This is a form of summarization. Our word2vec model is using dimension 100 vectors, and the resulting summarized tweet vectors are also of dimension 100.

4.1 Sentiment Analysis Performance Metrics

We used the following 6 measures for model evaluation: Accuracy, Precision, Recall, F1-Measure, Receiver operating characteristic (ROC), area under curve (AUC), and Precision vs Recall AUC. Here is a description of these metrics within the context of our project:

Accuracy: the ratio of correct predictions to total data points.

Precision: the fraction of tweets with positive predicted sentiment that actually had positive sentiment.

Recall: the fraction of tweets with positive sentiment that we predicted to have positive sentiment.

F1: the harmonic mean of precision and recall. This metric is useful for imbalanced datasets where a high accuracy can be achieved by a random classifier, but because this dataset has roughly equal proportions, it is somewhat redundant here.

ROC AUC: The ROC curve is a plot of a model’s true positive rate vs. its false positive rate for different probability threshold values. Intuitively, thresholds of 1 and 0 give equal true and false positives so the ROC curve will intersect the line through the origin w/ slope 1 at 0 and 1. A random classifier will have equal proportions of true and false positives for all thresholds. Therefore, an intuitive measure of classifier performance is the area under the ROC curve compared to the area under the random classifier ROC curve (0.5 always).

Precision/Recall AUC: The Precision/Recall curve is a plot of (Precision,Recall) pairs for different probability thresholds. A random classifier will have constant precision for all thresholds, but its recall will vary from 0 to 1 with the threshold value, so a random classifier’s Precision Recall curve will be a horizontal line at the class ratio. Similar to ROC AUC, we score the model by its AUC gain versus a random classifier. This metric is useful for problems where the classes are not balanced or where we care more about the positive class, given that this dataset is symmetrical, it gives roughly the same information as the ROC curve.[2]

The following subsections detail performance of various classifiers trained on the sentiment analysis corpus mentioned earlier. This testing allowed us to choose the model for our final rendition of the project involving stock price prediction.

4.2 Random Forest

Random forest provided the initial baseline:

Table 1: word2vec + Random Forest Classifier

Type	Training Set	Test Set
Accuracy	0.684591533095	0.673612152101
Precision	0.681912795793	0.67805020432
Recall	0.82712112152	0.821646476258
F1 score	0.750909267723	0.73868361183

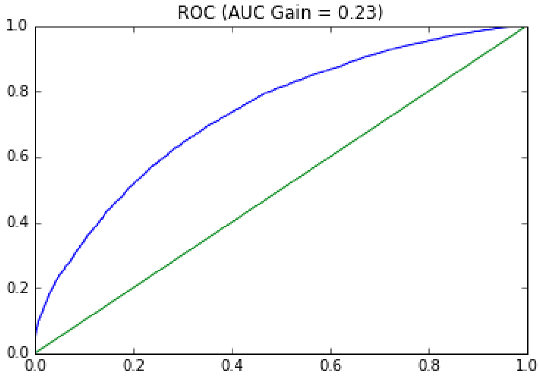

Figure 1: Random forest classifier ROC with AUC Gain=0.23

[2] These metrics would work equally well (although be somewhat more confusing) if we treated tweets with negative sentiment as positives.

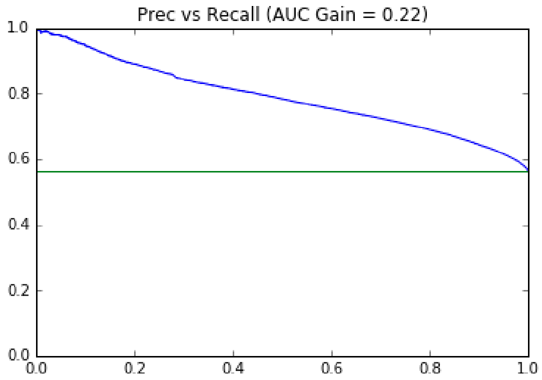

Figure 2: Random forest classifer precision vs. recall with AUC Gain=0.22

4.3 Quadratic Discriminant Analysis

QDA is a Bayesian classifier. Training and test accuracies were better than the baseline random forest but not as high as the multilayer perceptron classifier.

Table 2: word2vec + Quadratic Discriminant Analysis

Type	Training Set	Test Set
Accuracy	0.719376196853	0.71173456698
Precision	0.769485042382	0.760003838403
Recall	0.720252828854	0.706260032103
F1 score	0.744055433157	0.732146984054

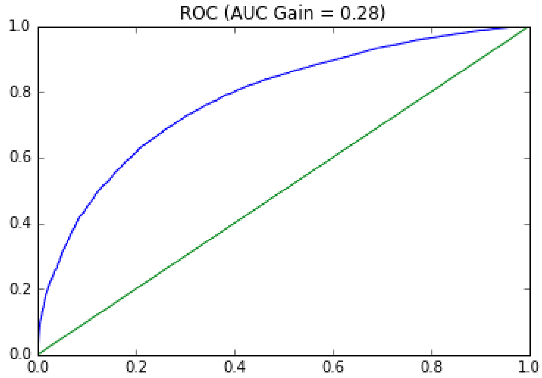

Figure 3: QDA classifier ROC with AUC gain=0.28

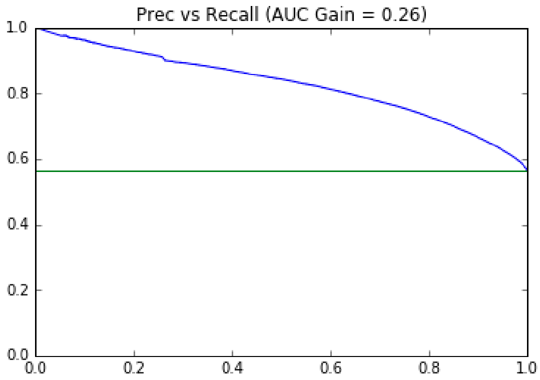

Figure 4: QDA classifier precision vs. recall with AUC Gain=0.26

4.4 Multi-Layer Perceptron

MLP yielded better accuracy on both training and test sets compared to random forest:

Table 3: word2vec + MultiLayer Perceptron (MLP)

Type	Training Set	Test Set
Accuracy	0.811256993379	0.742028552952
Precision	0.828856719281	0.765130190007
Recall	0.840213932107	0.775637595862
F1 score	0.834496685544	0.77034806483

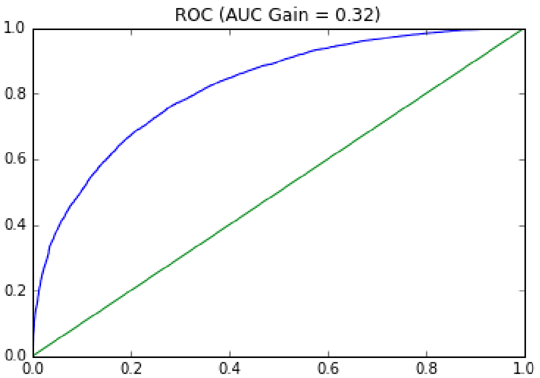

Figure 5: MLP classifier ROC with AUC gain=0.32

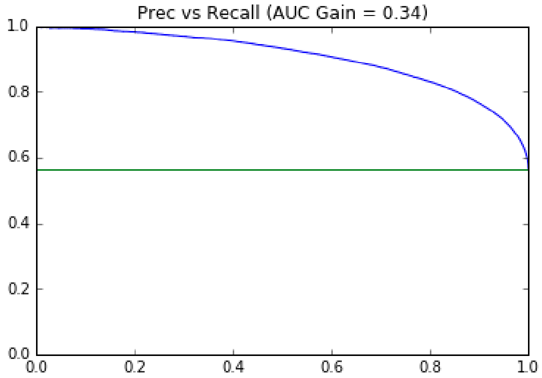

Figure 6: MLP classifier precision vs. recall with AUC Gain=0.34

MLP is simpler architecturally than the LSTM (discussed later), but it is a good choice of classifier because its training time is significantly lower than LSTM.

4.5 LSTM

The LSTM has a notion of memory. Its memory cell consists of four main elements: an input gate, a neuron with a path back to itself, a forget gate, and an output gate. The loop back allows the state of a memory cell to remain constant through time. The forget gate modulates the loop back and can allow the cell to "delete" or "forget" its previous state. LSTMs do not suffer from the vanishing or exploding gradient problem when long sequences are processed. It is an effective model for data that exhibits long-chain dependencies.

4.5.1 LSTM Architecture

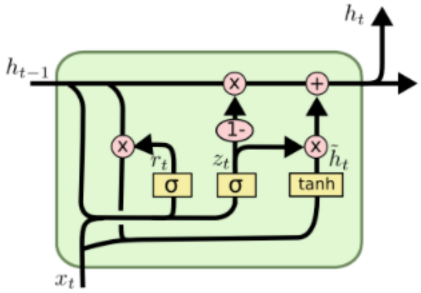

Figure 7: LSTM architecture.

The figure above shows the basic LSTM architecture, including the notion of time (t-1 is introduced) and the gates previously discussed. Corresponding functions are shown below.

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 8: LSTM functions.

Our specific LSTM network's architecture consists of three layers. Words will be fed into the word embedding layer, which will map words to word embeddings. The resulting word vectors are passed into LSTM cells, whose output will be passed into a dense layer with softmax activation function, and makes a prediction. The high level view of data flow is shown in the implementation architecture in the figure below:

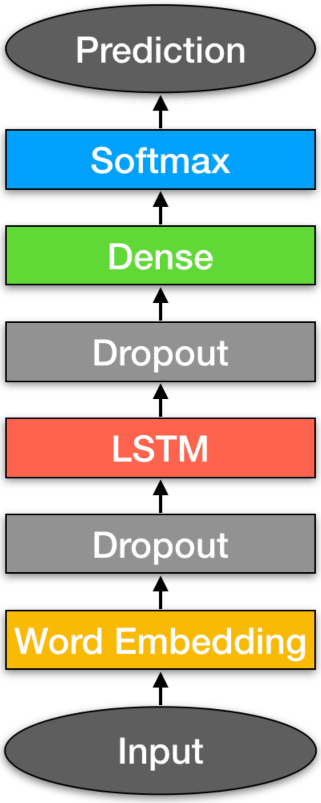

Figure 9: LSTM implementation architecture.

4.5.2 LSTM Implementation

Our LSTM model is implemented with Keras, running on top of TensorFlow. Keras is a high-level neural network API, which allowed us to quickly implement the model by simply stacking layers together:

```
model = Sequential()
model.add(Embedding(...))
model.add(Dropout(...))
model.add(LSTM(...))
model.add(Dense(..., activation='softmax'))
```

Our work largely follows Peter Nagy's implementation[5], but we modified the network and tuned hyperparameters to suit our needs. Training the model took two hours to finish.

4.5.3 LSTM Results

The model achieves an F1-score of 0.81 and an Accuracy of 0.81 on test datasets, outperforming the MLP's F1-score of 0.77 and accuracy of 0.74.

Metrics	Value
Accuracy	0.810448419184
Precision	0.803874201171
Recall	0.822235970902
F1-score	0.812951416989

While the results are good, the fact that the LSTM took so long to train made it a difficult model to use given our computational setup. With a real ML-machine or access to a set of GPUs in the cloud, we would have been able to more easily integrate it into the flow.

4.6 Hyperparameter Tuning

The hyperparameter tuning targets the hyperparameters of the word2vec model. Cross-validation was used on the original dataset. Since the MLP classifier had the best performance during the sentiment analysis evaluation phase, we chose to use this is the go-to classifier and invariant pairing. The F1-score of the word2vec + MLP classifier is used as a performance metric. Higher F1 score indicates a better performance.

4.6.1 Testing Methodology

Word2vec turns text messages into word vectors. However, there is no systematic metric for the evaluation of the model alone. Therefore, evaluation of the bundle: Word2vec + MLP, was used to test the performance.

The default method for hyperparameter tuning is to use an exhaustive grid search. However, since a single run of the 5-fold cross validation takes around 50 minutes, and some hyperparameters, such as size and iter, change the amount of work to be done, a brute force method would be too time consuming. Therefore hyperparameter selection and candidate selection was used. In the selection part, several hyperparameters are examined and the relatively important ones are picked. The resulting candidates for the tuning are:

- Size, which determines the dimensionality of the feature vector.
- `wind_size`, the maximum distance between current and predicted words within a sentence, governs the context of words.
- `min_count`, which defines the cut-off threshold for words.
- iter, number of iterations over the corpus.

The candidates for size is picked in the range from the default 100 to 150, `wind_size` is picked from 1 to 10, which includes its default value 5. `min_count` and iter are similar to the `wind_size`.

This means a 2-D grid search could take around 3000 testing instances, which is impossible to handle. Therefore, 1-D search was performed on each hyperparameter, and a final search in the potential best candidate's neighborhood was conducted.

The final candidates were further reduced based on their the neighbor's frequency in the 1-D search. If one of the two neighbors in the 1-D case has significantly lower score, it is omitted. Furthermore, size values are limited to only multiple of 4 to improve performance.

4.6.2 Testing Results

The following table summaries the 1-D search and the resulting search.

hyperparameter	optimum F1	best value	candidate
`wind_size`	0.6286	1	1,4
size	0.657	130, 140	132, 140
`min_count`	0.6026	7	7,8
iter	0.6101	9	9, 10

The neighborhood search in the end gives the best candidates for hyperparameters: (140,1,7,9), (140,1,8,9), (140,1,8,10), and (140,4,8,10).

5. STOCK PREDICTION

As mentioned earlier, because of a paucity of labeled sentiment data, we did not extend our sentiment analysis work to handle twitter data scraped using the API. Instead, we decided to use historical stock prices as proxy labels for archived tweets. Because we did not want to introduce another evaluation metric for models, preventing us from comparing stock prediction to sentiment prediction, we decided to perform classification on stock prices instead of regression.

To generate our (feature vector, label) tuples, we grouped tweets by company and date. We selected opening stock prices for each company for each day and compared then to the opening price of the stock the next day to generate binary label. For tweets, we reduced the resultant 2D array of word vectors with a multi dimensional average to generate a vector for each stock in our word embedding space. We did not test more elaborate methods of composing word vectors into stock vectors, in future work, we could use the method of "document" embedding described in https://arxiv.org/abs/1405.4053. The authors of this paper augment the standard context vector used as input for the simulated neural net of word2vec by adding an additional vector that describes which "document" (in our case this vector would map to a stock on a specific day) the context vector came from.

We then applied the same battery of evaluation metrics to this dataset as for our sentiment analysis dataset. We split the data using an 80/20 training-testing division and then feed it to the MLP model. The default sklearn model was used, which includes only one hidden layer with 100 nodes. Other parameters are: activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08

Table 4: Performance Metrics for MLP stock prediction

Metric	Train Set	Test Set
F1-Score	0.7862	0.7477
Precision	0.7481	0.7102
Recall	0.8285	0.7894
Accuracy	0.7458	0.7001

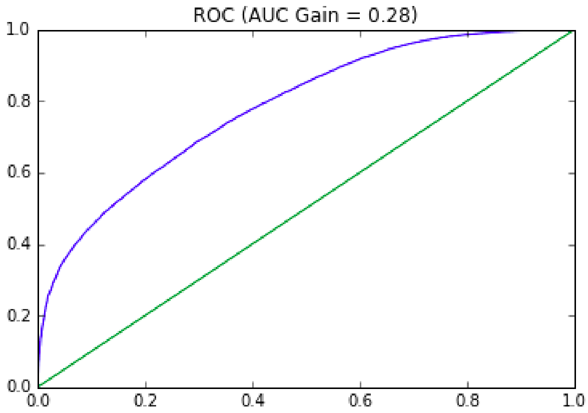

Figure 10: ROC curve for the MLP stock predictor model.

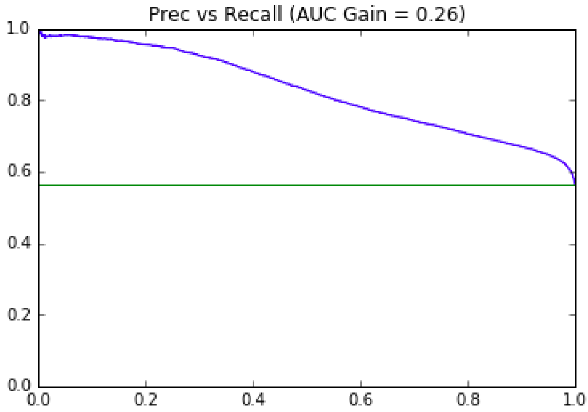

Figure 11: Precision vs. Recall curve for the MLP stock predictor model.

By comparing the (F1,Accuracy,Precision,Recall) 4-ples for training and test sets we see there is some overfitting, but we can see the model extrapolates meaningfully to the test dataset by comparing the ROC and Precision/Recall curves (in blue) of our MLP classifier with the random classifier (where decisions are made solely on a biased coin flip, in green). Note that the ROC and Precision/Recall curves were generated with test set data only.

6. CONCLUSION

Our performance, as indicated by F1-score, precision, recall, and accuracy, was reasonably high for the stock prediction class. Our F1-score of 0.7477 on the test set significantly outperformed the random classifier. We chose MLP over the LSTM due to the faster training time, but given more time we would have liked to integrate the LSTM into the stock prediction pipeline rather than having it exist as just an exploratory design.

6.1 Future Work

Due to time limitations, we were not able to integrate the trained LSTM model with the rest of our functional blocks. However, our design was pipelined such that blocks (preprocessor, embedding scheme, sentiment analysis, and prediction) can be easily interchanged. Apart from adding in the LSTM, we would like to investigate other word embedding methods (such as GloVe), as well as other text-to-vector summarization methods.

Since the LSTM model showed a lot of promise, some future work that could be done on it would be to add attention, since a tweet's core meaning is generally encapsulated in a subset of the words rather than in the whole sequence itself.

6.2 Task Distribution

Our work breakdown is summarized in the table below. Jonathan was responsible for performing the data crawling, preliminary data filtering, and writing the report. Amal preprocessed the crawled data and tested various models for sentiment analysis using the aforementioned performance metrics. Then, he performed the stock prediction/classification task. Jerry performed exploratory work by testing the RNN-based LSTM on the sentiment analysis corpus. Fred performed hyperparameter tuning.

Table 5: Distribution of Work

Task	Owner
Data crawling, parsing, data filtering	Jonathan Hurwitz
Preprocessing, sentiment analysis, stock prediction	Amal Duriseti
Sentiment analysis with RNN-based models	Jerry Chen
Hyperparameter tuning	Fred Xu
Writing Report	Jonathan Hurwitz

7. REFERENCES

[1] Bollen, J., Mao, H., and Zeng, X. Twitter mood predicts the stock market. *CoRR abs/1010.3003* (2010).

[2] Le, Q. V., and Mikolov, T. Distributed representations of sentences and documents. *CoRR abs/1405.4053* (2014).

[3] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *CoRR abs/1301.3781* (2013).

[4] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. Distributed representations of words and phrases and their compositionality. *CoRR abs/1310.4546* (2013).

[5] Nagy, P. Lstm sentiment analysis | keras.

[6] Samih, Y., Maharjan, S., Attia, M., Kallmeyer, L., and Solorio, T. Multilingual code-switching identification via lstm recurrent neural networks. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching,* (Austin, TX, 2016), pp. 50–59.