# Amazon Reviews Sentiment Analysis

### Project 3: Report Michael Turnell & Taylor Walraven

# Problem Statement

We attempted to use various NLP models and embeddings to predict the sentiment (positive or negative) of a given review.

# Data-sets

The data was 4 million reviews pulled from Amazon and posted on [Kaggle.](https://www.kaggle.com/bittlingmayer/amazonreviews/version/7#) The reviews were sorted into ‘positive’ (4 and 5 stars) and ‘negative’ (1 and 2 stars). Three star reviews were discarded as ‘neutral’.

Additionally, the CSVs are, at the time of writing, hosted on AWS:

* <https://mt-proj-001.s3.us-east-2.amazonaws.com/train.csv>
* <https://mt-proj-001.s3.us-east-2.amazonaws.com/test.csv>

# Data Wrangling

The CSVs hosted on Kaggle were already split into train & test groups, so all that was required was passing them through an embedding. We used both CountVectorizer from scikit-learn and a spaCy vectorizer. Due to the size of the data set and hardware restrictions encountered, the models run through the spaCy vectorizer used only 25000 rows.

# Exploratory Data Analysis

The only exploratory analysis we did was to investigate the positive/negative bias of the dataset -- the dataset itself is fairly close to evenly split between the two categories.

# Modeling

### Train-Test Split

The “given” train-test split was 90-10: 3.6 million train rows, 400k test rows. In using the smaller dataset for spaCy, this ratio was maintained.

### Keras Deep Learning Network w/ LSTM

#### Model Selection

This model used two stacked unidirectional LSTM layers to attempt to understand some amount of context within each review -- to differentiate between “Do not miss on buying this item” and “Do not buy this item”, for instance.

#### Model Scores

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F-1 |
| Negative | 0.84 | 0.84 | 0.84 |
| Positive | 0.86 | 0.86 | 0.86 |

### Keras Deep Neural Network w/ Hyperas

#### Model Selection

This was a standard Keras model with two dense layers. The hyperparameters were tuned with Hyperas, but our tuning parameters were implemented improperly, leading to a worse model. It is included for completeness’ sake.

#### Model Scores

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F-1 |
| Negative | 0.79 | 0.43 | 0.55 |
| Positive | 0.61 | 0.89 | 0.73 |

### Keras Deep Neural Network (Untuned)

#### Model Selection

This was the initial Keras model with two dense layers, but without any tuning done. This ran on 1 million rows as a subset, due to less processing time and RAM being required.

#### Model Scores

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F-1 |
| Negative | 0.86 | 0.85 | 0.86 |
| Positive | 0.86 | 0.86 | 0.86 |

### 

### Logistic Regression on TF-IDF

#### Model Selection

This was a standard logistic regression classifier model run on the TF-IDF vectors of each review. For being a bag-of-words model, it performs quite well.

#### Model Scores

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F-1 |
| Negative | 0.89 | 0.89 | 0.89 |
| Positive | 0.89 | 0.89 | 0.89 |

### 

### Naive Bayesian Model on TF-IDF

#### Model Selection

This model was run using scikit-learn’s built-in naive\_bayes library.

#### Model Scores

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F-1 |
| Negative | 0.81 | 0.84 | 0.83 |
| Positive | 0.84 | 0.81 | 0.82 |

### Support Vector Machine Classifier on TF-IDF

#### Model Selection

This model was run using scikit-learn’s built-in SGDClassifier(). It was improved by trying Random Search and Grid Search. Though grid search was the most expensive in terms of computation time, it ended up having the best results, listed below.

#### Model Scores

|  |  |  |  |
| --- | --- | --- | --- |
|  | Precision | Recall | F-1 |
| Negative | 0.88 | 0.89 | 0.88 |
| Positive | 0.89 | 0.88 | 0.88 |

# Conclusion

With bag-of-words vectors, it is very difficult to push into truly accurate territory to classify sentiment. It really does require a large dataset with a model that takes into account contextual meaning of words. While we were able to do quite well with some models, the difficulty these models have in taking into account that the two words “not” and “good” together mean the opposite of only seeing the word “good” can effectively ruin a prediction. So while perhaps these models might be useful in a different NLP context, different vectorizers or models with memory seem to be necessary for sentiment analysis.

### Further Exploration

We would want to investigate and explore more hardware solutions for processing the entire data set in a context manner. Given a suitable hardware solution could be found, we would use bidirectional LSTM layers with Keras, use BERT embeddings for a more contextual base of vectors, and even bigram/n-gram approaches to try and pull context out of the reviews to gain stronger models for sentiment prediction.

Furthermore, this provides a future opportunity to attempt to learn how to parallelize and batch-process training a model, in order to circumvent the massive computational time costs we ran into during the two weeks of this project.

# References

Scikit-Learn: <https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html>

Hyperas: <https://github.com/maxpumperla/hyperas>

Keras: <https://keras.io/layers/core/>

Countless stackoverflow pages: <http://stackoverflow.com>

Several blogs: <http://towardsdatascience.com>