Andrew Vernon
4//12/2021
CPT_S 315
Data Mining Project

Amazon Review Sentiment Analysis

Introduction

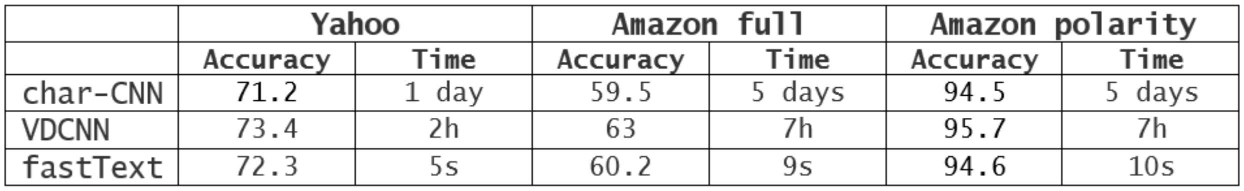
Customer reviews are integral for a company in order to respond to customer feedback. Small companies with not many reviews might have the luxury of being able to read through the reviews and make adjustments accordingly. Some big companies, however, have upwards of millions of reviews. This is a trend called data overload that is becoming a growing problem for many successful corporations. It's impractical for these large companies to hire a team to read and process these reviews by themselves. Due to the number of reviews that need to be processed, it would take far too long to make accurate assessments of all the reviews. Moreover, each review is subject to the reader's bias based on the opinion or even the mood of the reader. The companies know that they need more insights to make better decisions, but many of them struggle with how to acquire those insights efficiently and accurately. Sentiment analysis provides companies with information that is much simpler to process than written reviews, and also gives a much clearer path of action from those reviews. Because of this, even smaller companies with less reviews to process can benefit from sentiment analysis.

Data Mining Task

The data mining task I initially planned on solving was to implement a binary classifier from scratch and modify it to be as accurate and efficient as possible to classify Amazon reviews as having either a positive or negative sentiment. However, there are many production-level classifiers that are free to use already, all of which would certainly be more accurate and efficient than the classifier that I would implement. Because of this, the data mining task takes the form of exploring the different ways of representing text reviews and learning methods. I wish to gain a better understanding of the strengths and weaknesses of the methods observed and provide insight as to when which classifier would be most useful.

At first, I viewed this task as extremely daunting because of the enormous number of reviews in the data set (four million across both the training data and the testing data!). Despite my preliminary concerns, I underestimated the capabilities of production-level machine learning classifiers.

The primary library of interest is fastText, used for efficient learning of word representations and text classification. Developed by Facebook Research in 2016, it has been shown to be on par with deep learning classifiers in terms of accuracy, while being multiple orders of magnitude faster. The below image shows a summary of the tests that Facebook Research conducted to test the speed of fast text to deep learning classifiers. As the results show, the accuracy of fastText is astounding given how little time it took to run.



	Yahoo		Amazon full		Amazon polarity	
	Accuracy	Time	Accuracy	Time	Accuracy	Time
char-CNN	71.2	1 day	59.5	5 days	94.5	5 days
VDCNN	73.4	2h	63	7h	95.7	7h
fastText	72.3	5s	60.2	9s	94.6	10s

https://research.fb.com/wp-content/uploads/2016/11/post00048_image0002.png

Other classifiers that will be observed are

- Linear SVM
- Stochastic Gradient Descent
- Decision Tree Classifier

For more insight, all of the classifiers above will be tested using a TFIDF-vectorizer as well as a count vectorizer. I am curious about the time difference using these two different vectorizers in terms of accuracy as well as speed, and hopefully even extrapolate on their differences.

Given how impressive fastText appears and how relatively new it is, I expect it to blow the other classifiers out of the water, specifically in speed. Another reason for this assumption is that it was built specifically for text classification, whereas the other classifiers convert the data to a vectorizer and then run the learning and predictions. Do any of the other classifiers even hold a candle to fastText? We will see later in the results.

Technical Approach

The data was already suitable for fastText when downloaded, so aside from the preprocessing there was no extra work done. For all the other classifiers, however, they must be converted to a vectorizer. For each I chose to use two different vectorizers just in case there's a notable difference in accuracy or speed when compared to each other. These vectorizers need separate inputs of training data and training labels, so I separated them into their own respective text files for this.

After creating the vectorizers, the rest was pretty easy. I input the vecterizors into each classifier and calculated the time it took to run as well as the accuracy. Below is a representation of the vecterizors and each classifier that uses them.

Evaluation Methodology

The data files were downloaded from Kaggle from a user named Adam Bittlingmayer. He lifted the data from another user's Google drive, by the name of Xiang Zhang. The data was originally in a csv format, but was converted to the following format in order to be compatible for the fastText library.

```
__label__<X> ... <Text>
```

Because the data being used only has two labels, `__label__1` and `__label__2`, all samples of data either have `__label__1` or `__label__2` followed by the review, no quotes. The training data contained 3.6 million lines of data in the format above, and the testing data contained 400k lines. Because of the size of these files, the downloading of LTFViewer (Large Text File Viewer) was required to even open the files.

Once viewed, it was apparent that the text should be normalized, since some words contained uppercase letters or punctuation. Using the command line, the following commands were used to achieve this on both data sets.

```
cat train_reviews.txt | sed -e 's/\([].[\!?@#$:,"/()]\)/ /g' | tr
"[:upper:]" "[:lower:]" > train_preprocessed.txt
```

```
cat test_reviews.txt | sed -e 's/\([].[\!?@#$:,"/()]\)/ /g' | tr
"[:upper:]" "[:lower:]" > test_preprocessed.txt
```

****

As fast as fastText already is, this preprocessing would increase the speed even further due to the vocabulary of the model being reduced. This preprocessing would also increase accuracy due to words with a capital letter or punctuation being treated the same.

The metrics that were chosen are speed and accuracy. Given that we are observing binary classifiers, accuracy is the most relevant metric for performance, and speed is the most relevant metric for efficiency. Practically, these are the most important, but confidence gets an honorable mention due to being able to adjust the learning rate. Given how large and diverse the data set is, a supervised learning rate is unlikely to change the results in a significant way.

Results

The results were almost as predicted. FastText didn’t outperform as much as expected, accounting for all of the apparent advantages. All the other classifiers with the exception of both of the decision tree classifiers kept up in terms of accuracy, while staying reasonably back in terms of speed.

	Accuracy	Speed (minutes)
fastText	0.911	2.87
Linear SVM - TFIDF	0.911	22.43
Linear SVM - Count	0.902	52.96
SGDC - TFIDF	0.891	21.3
SGDC - Count	0.908	21.7
Decision Trees - TFIDF	0.674	1.25 days
Decision Trees - Count	0.578	≅7 days

Lessons Learned

Since the data being handled is a large data set, having processing that is time efficient is key. In practice, it’s hard to imagine using any of the other classifiers for text representation when fastText is available. In hindsight, it’s a bit of an unfair test between fastText and the other classifiers because fastText was developed specifically for text representation. All the other classifiers can have other elements that can be learned and predicted on, giving them many more applications practically. Specifically for binary classification on text reviews, however, look no further than fastText.

That all said, the other classifiers held up better than I thought they would. In terms of efficiency, they were about just as good as fastText. The time it took to set up the vectorizers took the majority of the time for most of the classifiers. Therefore, if the vectorizers are saved, you could hypothetically run multiple classifications consecutively without having to set the vector up again.

Something striking as well was the time difference between the count vectorizer and the tfidf-vectorizer on the Linear SVM Classifier. In every other case as well, the count vectorizer performed slower than the tfidf-vectorizer. The main factor to consider here is that the data set was huge, thus potentially favoring the tfidf algorithm in that respect. In all classifiers on this data set, tfidf-vectorizer was faster. There was therefore no evidence to favor the count vectorizer over the tfidf-vectorizer, though in most cases the tfidf-vectorizer is only favored slightly.

Bibliography

Abu-Rmileh, Amjad. “How Does FastText Classifier Work under the Hood?” *Medium*, Towards Data Science, 24 Feb. 2019, towardsdatascience.com/fasttext-bag-of-tricks-for-efficient-text-classification-513ba9e302e7.

Bojanowski, By: Piotr, et al. “FastText.” *Facebook Research*, 8 Nov. 2016, research.fb.com/blog/2016/08/fasttext/.

Kyle Murray Professor of Marketing, and Dominic Thomas Senior lecturer. “Managing the Highs and Lows of Data Overload.” *The Conversation*, 9 Feb. 2020, theconversation.com/managing-the-highs-and-lows-of-data-overload-97363.

Levitin, Daniel J. *The Organized Mind: Thinking Straight in the Age of Information Overload*. Dutton an Imprint of Penguin Random House, 2017.