Exploration of Online News Source Data Set using Machine Learning

Kai Mcconnell

Abstract

This paper explores and analyzes the Online News Source dataset from UC Irvine's dataset repository. In the process multiple new features are added such as word popularity, word embeddings, and parts of speech. These new features are used to create a machine learning model to predict the popularity of each article. The prediction rate as a result of these new features increases dramatically showing the correlation between these features and an article's success.
`https://github.com/Kaipie5/FinalProject`

1 Introduction

The basis for this paper is a machine learning and data mining dataset called Online News Popularity found in UC Irvine's dataset repository [1]. The dataset comes with a number of features already defined and a success metric. The success of each article is defined as the number of shares the news article receives. The goal of this advanced project is to discuss and evaluate the addition of several new predictive attributes to the Online News Popularity Dataset, related to the title of the articles. These include the popularity rating of words, word embeddings, and parts of speech. The research question is whether any of these additions increase the accuracy of predicting the success of an online news article and if so which features have the most benefit for the model.

2 Background

To understand the background of this project it is important to understand what the original data looked like and how it related to the new features. The original dataset had 39644 articles with 60 features of varying types each. The features were all either boolean values (0 or 1) or numbers. There were features such as the URL which I use in my data preprocessing steps. Another important feature was the channel type of each article. Channels act as different sections of the online news website such as business, technology, and health. Beyond these features which were used to gather more information for my new features, there were the positive or negative sentiment of the article's content, number of

images in the article, and what day of the week the article was published on. The feature I thought conspicuously missing was the titles. This meant in order to analyze the popularity of the words in the title of the articles I would need to use previously existing features to find the title information for each article.

3 Data Preprocessing

There were a number of steps I had to take along the path to adding my new data to the data set. The first step was to find the actual titles of the articles. Since the dataset only contains each article's URL and not the actual title I had to scrape the titles off of the web. In order to do this I sent requests to each URL and got back each webpage's HTML. I then used a regular expression to search for the first reference to <title></title> on the page. I then scraped the title of the article which would be in between these two HTML tags.

It was here that I ran into my first struggle. There are over 39000 rows in the dataset and each page request was taking about a second and a half. This would have translated into over 15 hours of compute time. I began to look into how I could speed this up. My first thought was that the requests were what was slowing my computer down. Because I thought I was request bound rather than CPU bound my first thought was to parallelize the code so that I could have multiple requests going in tandem. This appeared to work a little bit however I ran into a problem where my computer would break after about 30 minutes of computing. I then decided to look closer at why my code was taking so long and it turns out that it was not request bound but CPU bound. The library I was using, Beautiful Soup, was struggling to fully parse the entire HTML page from the request rather than searching for one small piece. It was then that I realized I could simplify this process by using a regular expression instead. By speeding up the rate at which I could read the requests I found using a serialized solution not nearly as burdensome as before. My computer still took about 6 hours to scrape all the titles off of the news site however once I had successfully gotten the titles I stored them in a file using the pickle library. This meant that I only had to do this computer intensive part of my experiment once. After that I just reloaded the file I had created and worked from there.

After getting all of the titles for my dataset the next step was to break these titles into words and give them weight based on the number of shares that specific title had received. By doing this for every title I was able to create a dictionary mapping words to the number of shares. I then sorted this dictionary with the greatest number of shares at the top. As I had expected the top words tended to be unimportant filler words such as "and, but, the, it, etc". In order to fix this I decided to divide the value of each word by the number of articles that word was in. This served to normalize my data and ensure that only words of actual value showed up on top of the list. I repeated this process for each channel such as business and entertainment in order to not only have an overall list of successful words but also words that would be successful in

specific contexts. I stored the dictionaries for each channel as well as the overall data in files once again so that I would not have to recompile them each time I ran my program.

4 New Features

The first new feature is the highest title word popularity. I assumed that the word which had the most shares in the title, would be the greatest predictor for success. Once I had the number of shares I would normalize it by dividing by the largest number of word shares. In my case it was 441000. In order to further increase the predictive value I also subtracted the normalized number of shares for the title that I was calculating the feature for, so as not to have it help predict itself. Once I had completed the normalized weight for each title I added it to my dataset as a new column. I did all of this for each channel as well as overall. This meant that I added 7 new attributes to my dataset in this word popularity phase of my experiment.

The next feature I added was word embeddings. Word embeddings take the form of vectors which represent the associations between words. The best example being king + woman = queen. These word associations are built off the words found within the titles and their association to each other. Each vector is 100 values and so in order to flatten it so my models could handle the data I added 100 new columns each of which were the sum of the words found in the title's corresponding vector value. This gave me 100 new features for my dataset to train and test on.

The last feature I added was parts of speech. I took each sentence and parsed words into their parts of speech. To put this into my dataset I created new columns for each part of speech and input a 1 if the word was that part of speech and a 0 if not. I then took these 35 new columns and added them to my existing dataset as new features.The last step for my experiment was to see how these additions affected the accuracy of my models.

5 Methods

My methods draw upon the prior work of Xu who had done extensive analysis of the base data set [3]. Xu's work was to discover the best threshold for number of shares to predict success as well as which models would be the most successful. Xu also worked to analyze the best training to testing ratio which I then used in my own models. Using Xu Wu's work my models worked on the assumption that fewer than 1400 shares was unpopular and greater than 1400 shares was popular. The models I used were Naive Bayes, Random Forest, and Decision trees from sklearn's library [2]. These models would then be trained on 60 percent of the data and be tested on 40 percent of the data again based on Xu Wu's work. I trained and tested on a random sample of the dataset 10

times and then averaged the results to improve accuracy.

6 Results

I ran all my results with the ratio of 60 percent training and 40 percent testing. Tables 1, 2, and 3 compare the accuracy achieved with additional features to that of the original data set.

Models	Original Data Set	Word Popularity Data	Increase in Accuracy
Decision Tree	58.328%	73.389%	+15.061%
Naive Bayes	60.830%	70.267%	+9.437%
Random Forest	66.348%	78.512%	+12.164%

Table 1: compares the accuracy of the three models on the previous data with adding the word popularity data.

Models	Word Popularity Data	Parts Of Speech Data	Increase in Accuracy
Decision Tree	73.389%	74.842%	+1.453%
Naive Bayes	70.267%	71.248%	+0.981%
Random Forest	78.512%	79.883%	+1.371%

Table 2: compares the accuracy of the three models on the Word Popularity data with adding the parts of speech data.

Models	Word Popularity Data	Word Embeddings Data	Increase in Accuracy
Decision Tree	73.389%	74.117%	+0.728%
Naive Bayes	70.267%	71.106%	+0.839%
Random Forest	78.512%	79.705%	+1.193%

Table 3: compares the accuracy of the three models on the Word Popularity data with adding the word embeddings data.

7 Discussion

There were some important improvements that occurred as I added features to this dataset. The most substantive improvement came as a result of the addition of word popularity weights for each news channel. I found I was able to increase the accuracy of all 3 models I explored by quite substantial amounts. My biggest improvement was on decision trees where I was able to increase the

model's accuracy in prediction from 58.3% to 73.4% an increase of 15%. This means that the model was able to predict correctly slightly over 15 times out of a 100 more than with the original data set. This is a substantive improvement as it means that my new attributes had quite the predictive weight on this problem. Based on my data the model which will have the highest accuracy in predicting the success of a news article is the Random forest Model. I was even able to increase this model's accuracy from 66.3% to 78.5% an increase of 12.2% which is also quite the success as this raises the likelihood that my model can predict a successful article to nearly 80 percent. Unfortunately word embeddings and parts of speech only improved the prediction rate by at most 1.4 percent depending on the model used. This showcases that the first set of features I added were the most powerful predictors of a articles popularity.

8 Future Work

I believe that these results could be boosted even more through a number of additional data cleaning steps I could have taken in my program. These steps include the fixing of my words to always have either upper case or lower case letters so as to not receive erroneous results, breaking up hyphenated words and the removal of certain types of words in a title such as numbers. Any or all of these steps could be taken to further improve the new predictive attributes. If I had more time I would have probably tried exploring using a neural network as my model. It is quite possible these newer model types could improve the prediction rate even further. I think it also would have shown some interesting results when comparing neural networks to the other 3 models I used.

9 Acknowledgements

I thank Janet Davis for her help advising on this project and editing this document. I also thank Chris McConnell for helping talk through some of the problems encountered along the way.

References

[1] Paulo Cortez Pedro Sernadela Kelwin Fernandes Pedro Vinagre. 2015. Online news popularity data set. `https://archive.ics.uci.edu/ml/datasets/online+news+popularity`. (2015).

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

[3] Xu Wu. 2015. Online news popularity. https://github.com/xuwu0922/Online-News-Popularity/blob/master/scripts/performanceCaculation.pyy. (2015).