Popular Truth

News Article Popularity Classification with Article Metadata and Content

Motivation & Data

I investigate different approaches to predicting the popularity of a news article, given article metadata and content. In a 2015 CS229 project, He Ren and Quan Yang used a random forest to achieve 69% accuracy in classifying articles as either popular (>1400 shares) or not. I aim to break the baseline by engineering new features on the scraped article content itself, combining what I've learned in CS229 and various NLP techniques. I obtained the dataset of article urls and metadata from UCI's Online News Popularity Dataset. To create a robust classifier, I used beautifulsoup to scrape all the articles of its author and content.

Models Rationale

Ridge Regression - Regression, particularly after the engineered features always serve as a good baseline.

Support Vector Classification - This tries to find margin that maximally separates popular and non-popular articles. There are metadata features, like sentiment and polarity, that may help construct an effective separation boundary.

Kernels - The cosine kernel will work well on the word2vec features, but it's a question what kernel to use for the LSA features. A custom "composite" kernel may get the best of both worlds.

Random Forests - This is based off of the baseline set by the previous project with 500 trees; with the new features I engineer, it's a question whether the same sampling/splitting strategy will work.

Neural Networks - It's not an ML project without this. If logistic regression performs well, it's natural to add layers.

Metadata Features

Attributes and Statistics

Time - Article popularity appears to be negligible long-term trend. I hypothesize the popularity gained over time "cancels out" with the site's long-term rise in visitors.

Date - Seven features are formed via the one-hot encoding for day of the week of publication. This may be useful for splitting a decision tree or a separation boundary.

Channel - Eight features are formed via the one-hot encoding for channel of the article - similar to date.

Summary Statistics

Tokens - These include number of tokens in title and content, number of unique tokens, etc. They may be weakly correlated and mildly useful.

Number of _ - (Images, videos, references, etc.) These may be weakly correlated as well.

References - (Min/max shares of referenced article); This is useful, but adds a lot of variance. This once again explains why random forests worked so well.

Processed Metrics

Sentiment, Polarity, Subjectivity - These are metrics in their own right. They are continuous and very useful for regression.

Best/Worst/Average Keyword - This may explain why random forests worked so well.

LDA - There're features that model closeness to modeled topics; these numbers are useful after splitting into random forests.

Content Features

Title
(TF-IDF Word Vectors on N-grams)

Word2Vec is a form of unsupervised learning that maps a set of words to continuous vectors in a way words that appeared in similar contexts (sentences, documents, etc.) have similar features have higher dot products. By taking a linear combination of the word vectors in a title, we can approximate a "title" vector. We want to make sure common words like "new" are weighted less than words like "breaking", so we scale each vector by the no. appearances in the title and the inverse log of no. documents it appeared in. We also add in bigrams, trigrams, ..., to octo-grams (until accuracy stops improving) to the vocabulary. The resulting title vector is appended as fifty new numerical features.

Keywords
(Word Vectors and Random Forests)

Similar to the title, we average the word-vectors of keywords associated with each article and append it as fifty new numerical features. Unlike the title, it doesn't make sense to analyze TF-IDF or N-grams; keywords are used to tag articles based on commonalities, hence decision trees.

Content
(LSA with Lasso Regression)

Again, we form a vocabulary of all n-grams up to hex-grams (we can push further, but six consecutive words seem like a reasonable cutoff for phrases) so we can apply tfidf. But wait. That's a lot of features! To reduce the dimension, we have two approaches - a) use Lasso Regression with a strong regularization constant to drive trivial feature coefficients (n-grams) to zero, or b) use LSA to capture the variance of the tf-idf's of the existing vocabulary set. There seems to be a trade-off (bias-variance) between dimension reduction via the two approaches, so we experiment to find the what combination of each. The result is (Table 2).

Table 1. The age of the article has little correlation with # shares, and so we need not to worry about it being a dealbreaker.

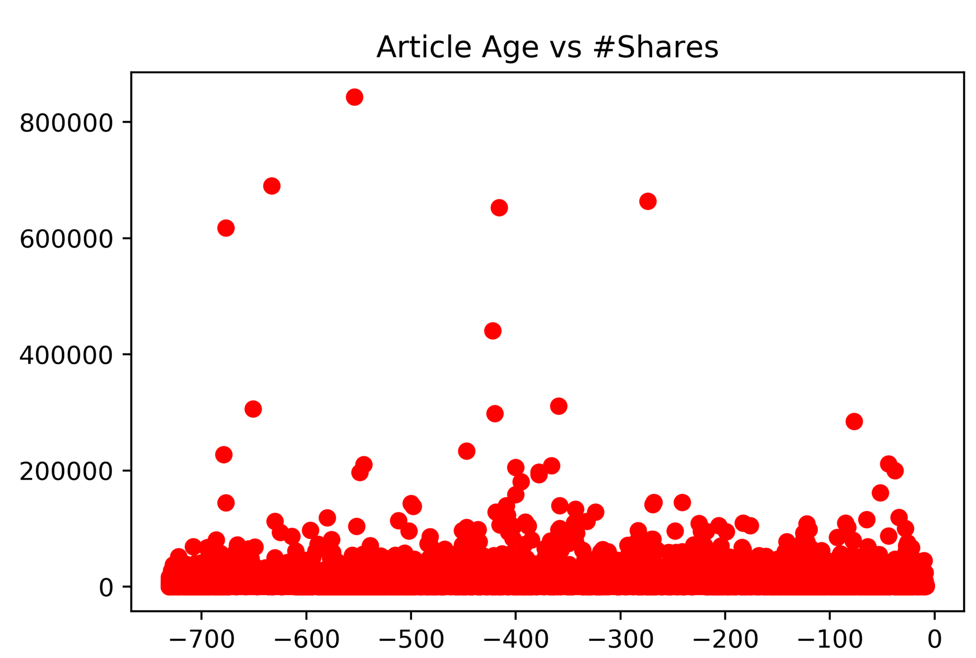
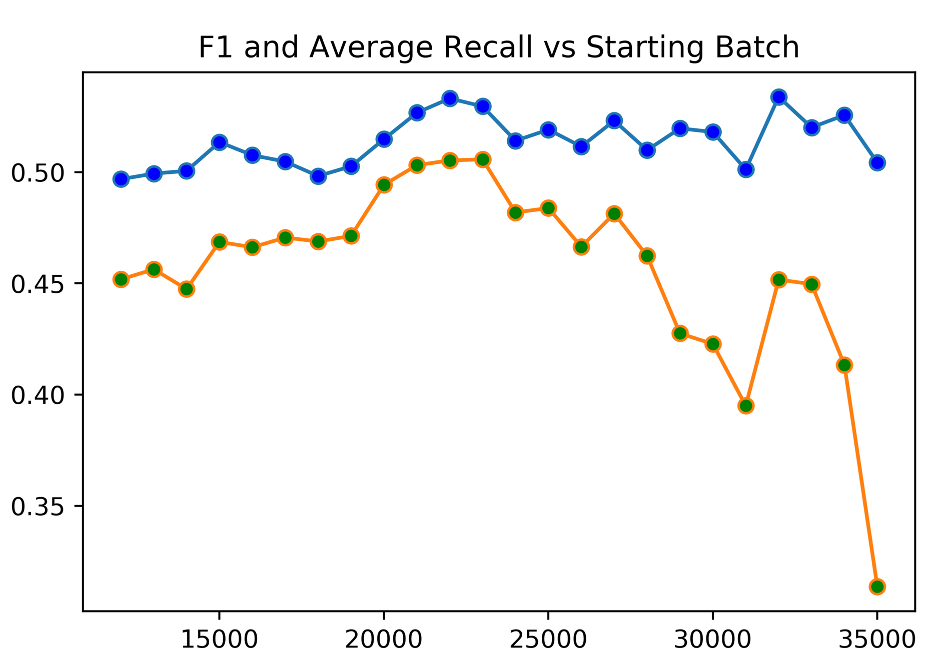


Table 2. There's a point of diminishing returns to how much historical content is useful for popularity.



Discussion & Conclusion

My main takeaway from this project is the uncertainty and excitement involved in applying machine learning to a relevant problem in industry. Text is incredibly rich in information, so extracting the right features from it required me to think critically and resourcefully, taking only data I need (Table 2) and eliminating features/dimensions when necessary (Lasso and LSA). I also got deeper domain insights, like the relevance of keywords, importance of titles, and reducibility of word vectors from the content. Finally, I learned how to combine a heterogenous set of features together, accounting for variable interactions/dependencies, and the bias-variance tradeoffs between different models (i.e. regression with no feature engineering vs random forests on "split"able features).

Conclusions (Being Finalized)

	F1 Score	Average Recall	Comments
Ridge Regression	0.71	0.65	This is useful as a baseline, and it clearly performs well after optimizing the few thousand features extracted from the content.
Support Vector Classification	0.65	0.65	This doesn't perform as well, and understandably so, as features have complex interactions and are tightly connected.
Random Forests	0.69	0.71	This is the baseline set by the CS229 project that helped inspired mine.
Neural Networks	0.70	0.69	This is more of a black box. Three layers with logistic activations perform well. There's a lot of hyperparameter tuning I have yet to experiment with, so take this with a grain of salt.

Future

Ultimately, the goal is to dig deeper into the article extract the best predictors of popularity, but first I am curious how much predictive power is in the url and first sentence of the article. Afterwards, I want to investigate a more recurrent approach by ordering the features (i.e. url => keywords => title => first sentence => first paragraph => etc.) and having the algorithm halt once it has "enough information" (this is similar to how humans read articles). Via rewards, the agent learns to prioritize which features are more important and which to look at first.

Contact Information

Michael Sun
Stanford University
Email: msun415@stanford.edu
Website: shininingsunnyday.com/portfolio

References

K. Fernandes, P. Vinagre and P. Cortez. A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News. Proceedings of the 17th EPIA 2015 - Portuguese Conference on Artificial Intelligence, September, Coimbra, Portugal.

Acknowledgements

All news articles belong to mashable.com.