Kaggle Report-Team ABBABA OH NO ABBBAB

Nick Halliwell, Aina Lopez, Yaroslav Marchuk

March 14, 2016

Introduction

Our team consists of Nick Halliwell, Aina Lopez, and Yaroslav Marchuk. In this competition, we were given both a training and test set consisting of features of various web links from mashable.com. We were asked to predict whether the website link would fall under one of five potential categories: Obscure, Mediocre, Popular, Super Popular and Viral. We ran several algorithms to make these multi-class predictions, and found that a random forest with tuned parameters outperformed all other algorithms. Below we underline the approach we took to this competition.

Method

1. Create new variables
The first step we took was to examine the data, and determine what kind of variables we could generate from the given data. Of the variables we were given, we decided that the url of the articles included useful information that could be extracted. An example of the url is the following:

http://mashable.com/2014/01/12/game-of-thrones-season-4-trailer

From the url, we took the year, day and month it was published as well as some keywords of the content of the article. Using text mining techniques, we collected all the keywords of the article titles. With this we created new variables indicating whether a specific word appears in the title of the article. Given that we have many keywords, we created a threshold for variables, meaning we created variables only for keywords that have appeared in at least 150 observations.

2. Feature Selection
3. Train the model
4. Evaluate the model
In order to evaluate our models, we used 5-fold Cross validation. Here, we randomly subset our training data into 5 folds, training the model on 4 folds and testing the model on the last fold. This process is repeated 5 times, with each fold used once for training. To stay consistent, we used 5-fold Cross validation on all algorithms used in competition.

Variable Selection: RFF

Models

- **Random Forest:**
  We decided to implement a regularized Random Forest, as this algorithm typically perform well on large datasets. Random forests often do not overfitting, however, the computational speed of this algorithm can be slow. The regularized Random Forest performed feature selection for us, and then fed those relevant features into a Random Forest. We optimized the algorithm for the number of trees used, the minimum size of terminal nodes, and the number of variables randomly sampled for potential splits. Ultimately this was our most successful model.

- **K-nearest neighbors:**
  We implemented a k-NN algorithm, trying both a function written by one of us, and an R package installed from CRAN. In both cases, the k-NN performed poorly. In terms of predictions, it's accuracy was lower compared to the Random Forest, and the k-NN algorithm was very expensive to compute. We ran the algorithm optimizing for different values of k, measuring distance using the Euclidean norm, yet the algorithm could not outperform the Random Forest.
- **SVM**
- **Boosting**

Results

(plot ?) Kaggle score vs CV score