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 sopular, and viral. We ran several algorithms to make these multi-class predictions. 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. We also standardized our features. 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
We used Regularized Random Forest (RRF package) to choose the best feature set.

3. Train the model
We trained different models optimizing their parameters explained in more detail below.

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.

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 overfit, however, the computational speed of this algorithm can be slow. The regularized Random Forest performed feature selection for us, and we 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 one of our most successful model.

- **K-nearest neighbors:**
  We also 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 than most of the other algorithms, and it was very expensive to compute. We ran the algorithm using the distance metric of the Euclidean norm, and optimized for different values of k, yet the algorithm could not outperform our Random Forest.

- **SVM**
  In addition to k-NN and Random Forest algorithms, we implemented a Support Vector Machine using a polynomial kernel. We optimized weights for the 5 different classes, giving higher weights to classes 4 and 5. The svm was outperformed by the Random Forest, and given the time complexity of the algorithm, we turned to other methods to use in competition.

- **Boosting**
  Finally, we tried boosting methods, gradient Boosting (XGBoost) more specifically. However, there were too many parameters to optimize and we did not observed significant changes during optimization. The accuracy did not outperformed Random Forest, hence discarding it. In addition, we implemented Generalized Boosted Regression Modeling (gbm) model, using h2o to speed up computations. In this case, we selected variables in a different way, using the variance importance attribute of gbm. The results were impressive, and very similar to random forest.

Choosing Parameters

In order to optimize the parameters of each algortihm, our approach was to create a wide range of possible values, with large intervals between each value. We then cross-validated all the models and chose the best. We then created a new range of values around the optimal value found, and used a smaller interval. We cross-validated again, and chose the optimal value. We repeated this procedure as many times as needed, shrinking the interval on each repetition.

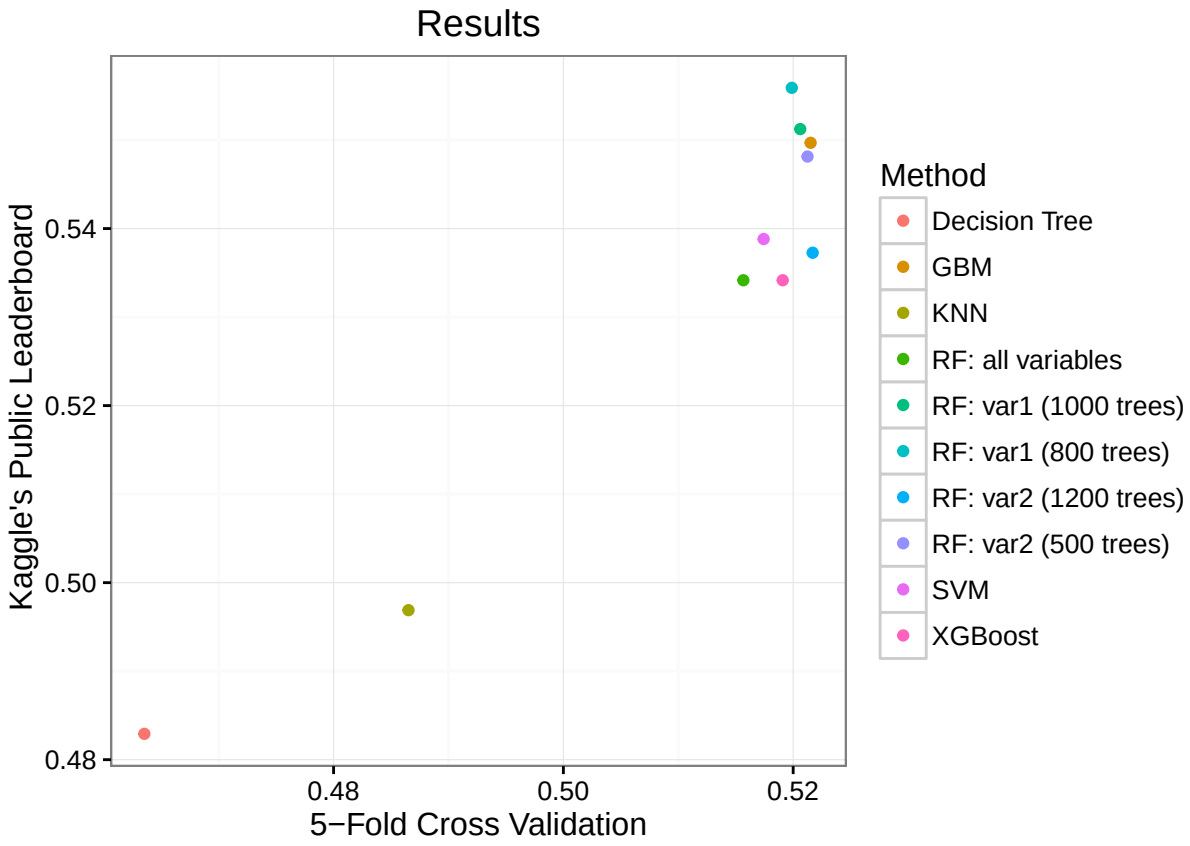
We found that only a few iterations were necessary. With more complex models however, this method became inefficient. We saw that choosing each parameter separetely is not practical, and that optimizing parameters is best done selecting them together. This is why we decided against using Support Vector Machines.

Results

We have plot some of the model accuracies (Kaggle public Leaderboard vs our 5-fold Cross Validation). The models that we are displaying are:

- RF: is the random forest. var1 and var 2 states for a couple of feature sets selected via RFF using different parameters.
- GBM
- XGBoost
- KNN
- SVM
- Decision Tree

Those are a small sample of all the models that we have tried, but represent the best accuracies achieved by each algortihm family.



Conclusions and Classifier

As we can see in the above figure, the accuracy displayed in Kaggle's public Leaderboard was not equal to the best accuracy in our 5-fold CV. Since Kaggle's public leaderboard is just showing a 7% of the test data, we decided to rely on our cross-validation results for model comparison.

For our final submission, we decided to choose the two models with best balance between their scores in Kaggle and in our Cross-Validation. Those models were: *GBM* and *RF: var 1 (1000 trees)*.

Since we have unbalanced data, confusion matrices of our models have been a very powerful tool for evaluation. We observed that most of the models struggled to predict classes three, four, and five. However, *GBM* had better prediction accuracy with the minority classes.

- GBM confusion matrix:

.	1	2	3	4	5	Error
1	5418	3942	118	0	0	0.4284
2	2502	11061	201	0	0	0.1964
3	822	3944	946	0	0	0.8344
4	129	681	86	103	0	0.8969
5	3	26	5	0	13	0.7234

- Random Forest (var 1 and 1000 trees) confusion matrix:

.	1	2	3	4	5	Error
1	4393	4947	137	1	0	0.5365056
2	2573	10771	420	0	0	0.2174513
3	714	4570	426	2	0	0.9254202
4	111	749	137	2	0	0.9979980
5	2	35	10	0	0	1.0000000

This increased accuracy in underrepresented samples is one of the biggest strengths of our gbm classifier.This classifier was implemented in h20 in order to increase computational speed. Regardless of what classifier we used, the error when predicting classes three, four, and five were higher than the other two.

The second model that we decided to use is Random Forest. We chose this one in particular as it achieved some of the best results both in Kaggle and in our Cross-Validation. Comparing its confusion matrix with the gbm model, we can see that its predictions for all classes are worse. Additionally, training this model was quite slow.

Unfortunately, we didn't had time or computational power to implement all of our ideas (supersampling, optimizing the seed, etc). However, we are confident in how our models will perform.