Predictive Modeling Project for COMS 4772

Team Sandwich: Peter Darche, Tonia Sun, Ian Johnson

May, 5 2016

Abstract

In this predictive modeling project, our team used several techniques for data preprocessing, and tried a variety of combinations of features, feature transformations, and classifiers. Our initial approach was to spend time researching various techniques that are common among Kaggle participants, and to devise a strategy based on articles and testimonials from previously-successful users. Our final predictor was a Random Forest classifier with 200 estimators and default parameters for max_features and max_depth. These defaults are hard-coded into sklearn as the square root of the total number of features, and None, or no max depth of the estimators, respectively. We also made use of one-hot-encoded dummy variables for the categorical columns and used cross validation on the training data to get an average prediction accuracy before submission. Additionally we dropped one of the columns from the dataset that was causing a large increase in the number of features all by itself, many of which did not have enough data points per value of the predictor to justify its inclusion in the final classifier. The result of this combination of features resulted in a classifier that achieved a public score of 0.94954%, and a private prediction accuracy of 0.94929%.

1. Data Preprocessing and Feature Design

Based on our reading, we found that the vast majority of time that goes into making predictions is directed at successful feature engineering. To handle this task, we decided to split up the research of feature design into four categories:

a. Understanding the features / basic EDA / cleaning - Peter

b. Feature transformation (normalizing, etc.) - Peter

c. Feature reduction/selection, data size reduction, data partitioning - Ian

d. Feature design and creation - Tonia

a. We familiarized ourselves with the dataset in two ways: researching the features and exploratory data analysis. Though scrutinizing the documentation surrounding the dataset didn?t provide us a full description of the semantics of the features, it gave us valuable hints about what they represented and how we could modify them. We found, for example, that many of the columns had exactly similar value sets (but differing values), indicating they represented actions and / or attributes of the instruction giver and follower. We also found there were computed features (which were the concatenation of pairs of other features in the dataset).

Exploratory analysis gave us further hints about how to preprocess the data. We started with basic checks for data cleanliness. Besides duplicate records, the data was clean and dense (which is not surprising given how and why it was collected). We then computed a number of descriptive statistics, including the distributions of features, conditional distribution of labels given features, and correlations between features.

b. After familiarizing ourselves with the dataset we started the feature engineering process. Initially, we took steps to reduce the variance across features in the dataset, using several sklearn methods. We used the StandardScaler class from sklearn.preprocessing to standardize each column. After computing the covariance matrix of this standardized dataset using a built-in numpy method, we found that several of the columns had a nontrivial association with our prediction variable, or with one another (see Appendix A).

We made note of these correlations for possible later use in boosting our predictor?s performance. Additionally, we used the PolynomialFeatures class from sklearn.preprocessing to get additional features from our two continuous columns. With degree-two interactions we did not see any improvement in any of our classifiers, and these features did not make it into our final feature set.

c. Another method that we used for feature selection was the RFE class from sklearn.feature_selection. This class recursively reduces the amount of features used for the classifier, until a designated number of features is reached. Trying a subset of features from all of the numeric (either boolean or zero-one-two or continuous) columns, up to 1000, did not provide us with a more accurate predictor, and so we did not end up subsetting the features based on this method in our final classifier. We nonetheless noted which columns came back from this function for later use. We also tried using an ExtraTreesClassifier from sklearn, which had a best_features method, to rank and find the top 50-100 features by importance, which allowed us to test our algorithms on a smaller subset of features as we iterated so we could test more models quickly. We found by graphing the best features that the top 50 contained 80% of the total importance, and by the top 200, most of the feature importance dropped down to a very low level. Ultimately, we decided to run a custom feature importance ranker that ran RF, the classifier we chose, 30 times and dropped the features with zero importance. Additionally, there were several columns that had a notable correlation with the label, including both continuous columns, which each had a 41% correlation with our label. This led us to decide to keep these correlated columns, and mark them for additional feature transformation.

d. The most powerful feature transformation for us ended up also being the most necessary. Sklearn classifiers require all features to be continuous, and thus all of our categorical columns needed to be converted to a more useable form. We used the get_dummies method from Pandas to perform one-hot-encoding on these columns and include them in our classifiers. This raised our number of features to 5587, and was the set of features for which we did the subsetting in part C.

2. Model/algorithm description

Before selecting our model, we tried spot checking as many algorithms as we could in order to determine which performed relatively well on the pre-processed data, and later, as we refined our feature engineering process, we iteratively tried the algorithms that had performed well on the processed data before finalizing which to use for our final submission.

We began by testing some of the top algorithms recommended by Wu et al., 2008 on the top 10 algorithms in data mining. Some of the algorithms we tested included:

Random Forest (RF): 0.71422 (numerical features, raw), 0.89 (intermediate test with subset sample of 100 columns) Random Forest is an ensemble method that combines weak learners to form a strong learner, with each ?tree? acting as a weak learner and the entire combined forest as the strong learner. Compared with other classifiers, it is considered one of the most efficient and fastest in run-time, especially compared with other eager learners. Because they work better with uncorrelated trees, data processing is especially important in improving the performance of the RF to cull the correlated features. We used sklearn.ensemble?s RandomForestClassifier library. RF performed quite well on the raw data and thus made our short-list of models we tested later with the refined data set. When we used all of the numerical features, our score increased to 0.86953.

K-Nearest Neighbor (KNN): 0.81184 (numerical features, raw), 0.83 (intermediate test with subset sample of 100 columns) KNN is a classifier that is considered a lazy learner, in that it does not build a classification model as a part of the training process. Instead, it takes the k closest training data points (?nearest neighbors?), as determined by a distance metric, and classifies new data using the classes of the neighbors. We used sklearn.neighbors? KNeighborsClassifier library. Though KNN was successful in our spot checking stage, we found that it performed worse relative to RF once we implemented data cleaning and train / test data splitting, possibly because it performs less well with noisier data, and it was more difficult to test in our post-processed stage due to the higher storage requirements of KNN.

Logistic Regression: 0.72234 (numerical features, raw) The logistic regression algorithm is considered an extension of linear regression, except with the ability to estimate class probabilities using feature values. We used sklearn.linear_model?s LogisticRegression library and found that as we iterated over our data set and feature selection, RF outperformed LR.

Discriminant Analysis. Linear: 0.71770, Quadratic: 0.70242 (numerical features, raw); LDA: 0.87 (intermediate test with subset sample of 100 columns) Discriminant Analysis finds the linear boundary between classes to separate them (either quadratically or linearly), using distributional assumptions. We used sklearn.discriminant_analysis? LinearDiscriminantAnalysis and QuadraticDiscriminantAnalysis libraries.

Support Vector Machines (SVM): 0.70846 (numerical features, raw) SVM classifies the data into two classes by using a hyperplane; by using a kernel, the SVM can map data into higher dimensions and separate with the hyperplane by maximizing the margin between the plane and each of the two classes. We used sklearn.svm?s Support Vector Classification library. SVC performed relatively well on the raw data, and we tested it in our later stages as well. However, we found that ultimately, even when testing with our one-hot-encoded categorical columns, it reached a maximum of 85%, which was still short of the performance of RF.

AdaBoost: 0.73176 (numerical features, raw) AdaBoost uses boosting, an ensemble algorithm, with each new iteration of the algorithm training the weaker learners and refining the weights for the learners. We used sklearn.ensemble?s AdaBoostClassifier. While AdaBoost performed well in the initial raw data run-through, when we tested it with our refined data set, it also did not manage to overtake RF in performance.

3. Model/algorithm selection

A. Parameter Tuning We tried using the grid_search library from sklearn to test several combinations and ranges of parameters to see whether there was an ideal number or combination of depth, features, and estimators that would improve our score. We found certain high enough levels of each would take very long times to run (and thus slow our iteration testing process), without having a noticeable impact on performance. For example, we found setting max depth to 250 instead of 5 had an impact of +8% in performance. For number of features, we found improvement of 6% from 10 to 100, but no clear noticeable difference from 100 to 1000. After plotting a graph of prediction accuracy versus values for each of these settings, the accuracy seemed to asymptote at the following values:

N_estimators: 200 Max_features: 100 Max_depth: 200

We came to the conclusion that increasing past the following values for the parameters could only lead to marginal improvements in classification, and that we could use these values to speed up our training process:

B. Algorithm Selection From our initial model spot checking, we determined that RF was one of the top algorithms we wanted to focus on with our cleaned data set. While we tested with others with top results (e.g., KNN, SVM), these continually did not perform as well as RF with the updated data set. In addition, RF was one of our fastest algorithms and allowed for more iterations of testing cleaned features, which was also part of our consideration.

While we had a small range of cross validated scores for RF on the cleaned data due to the different ways we tried our own feature engineering and column renaming, we continued to see RF as the highest performer. For example, in one run on the same set of partially cleaned data, RF was at 0.914, whereas SVM was 0.874 and Logistic Regression 0.852.

Our selection of RF is backed up by secondary research and the experience of other Kaggle competitors, which boosted our selection of this choice. According to Caruana et al., who tested the performance of various popular supervised learning algorithms at ranges of dimensionality from 1,000 to 700,000 features, RF is one of the consistently highest performing algorithms, across the entire range (2008). Certain algorithms, especially SVMs, perform better at higher-dimensional spaces (at the higher end in the range, especially above 10,000), while towards the lower end of this spectrum (where we are, at 5,000 features), tree ensemble methods tend to perform better and at greater efficiency. KNN and Decision Trees were found to decrease in performance at higher than 1,000 features.

Outside of the literature, from our reading on data science and Kaggle competition blog posts and forums (e.g., Kaggle forum, R Bloggers, Fast ML, KD Nuggets, ML Wave, Quora), we determined that RF has consistently performed well as an algorithm across competitions, with much of the difference in winning performance determined by feature engineering to feed into the RF, which is consistent with our own empirical findings, as well as the published literature.

We considered doing PCA but decided against it for RF because we did not have a high need for dimensionality reduction, since RF can run easily on large data sets with high number of predictors because only a random subset is used to build each tree.

4. Predictor Evaluation

As a part of the evaluation process, we cross validated our data and split it between training and testing sets. Originally, we used the test_train_split method from sklearn.cross_validation, and computed the mean prediction rates from several iterations of training on a subset of the training data, testing on the corresponding labels for this subset of the data. We saw 2% improvement between using 10% of the data each for train / test, and 50% of each. We later moved to using the cross_val_score method directly from sklearn, which was a more concise way to do our test/train split and K-folds in the same method.

5. Results of Evaluation and Analysis

After multiple iterations of trial-and-error feature selection and parameter tuning we were able to get our classifiers to achieve 93% accurately. We had difficulty, however, improving accuracy beyond that and so attempted a more detailed error analysis to inform later iterations of feature engineering. This error analysis consisted of plotting ROC curves and confusion matrices (Appendix C) and scrutinizing the distributions of false positives and negatives. We also tested whether underperformance was due to high bias or variance, and after testing error rates over varying training data sizes concluded that it was more attributable to the latter (Appendix C).

6. Team Work Allocation

Originally we divided the research work between our three team members as was described in section 1. There was some overlap between research from each member in these areas, and each team member ended up building and testing various classifiers based on our findings from this initial research. Each member contributed to the production of the final classifier and to this final report.

7. Appendices

Appendix A. Correlation between columns

a. Columns 9 and 5 - 100%

b. Columns 2 and 3 - 62% and 48% correlation with column 29

c. Columns 1 and 2 - 76% and 60% correlation with column 9

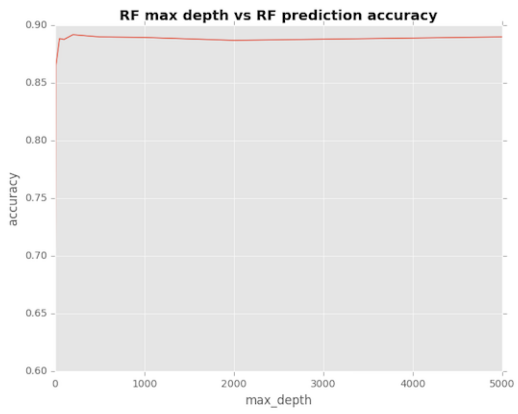
d. Columns 27 and 28 - 41%

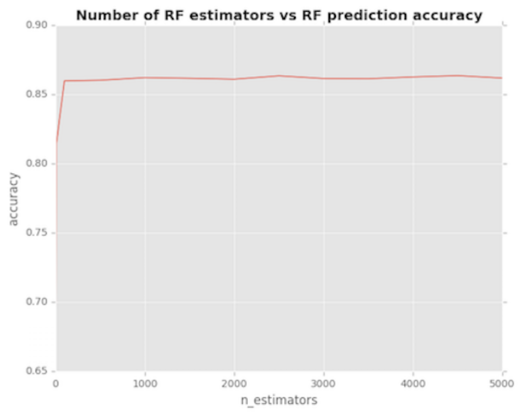
e. Columns 31 and 32 - 99%

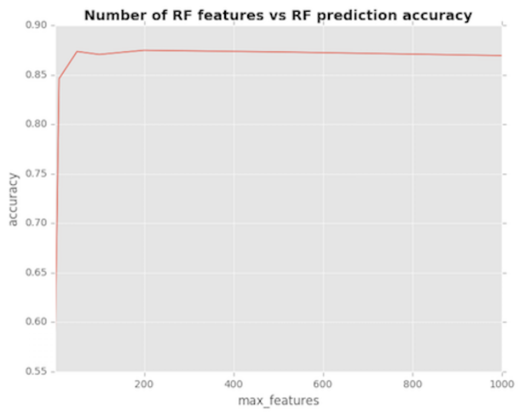
f. Columns 33, 34 - 89%

g. Columns 31 and 32 - 41% correlation with label

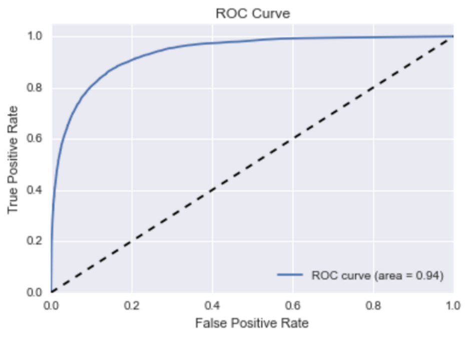
Appendix B. Random Forest optimal parameter calculation

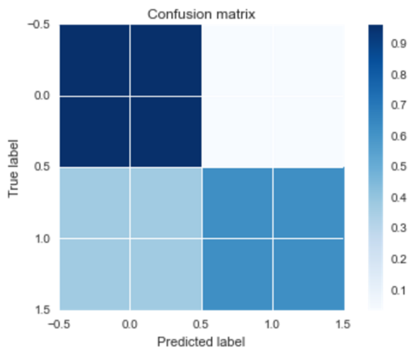


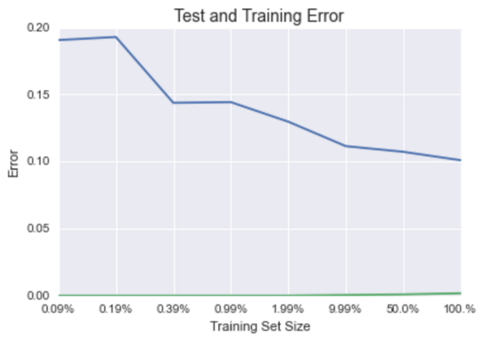


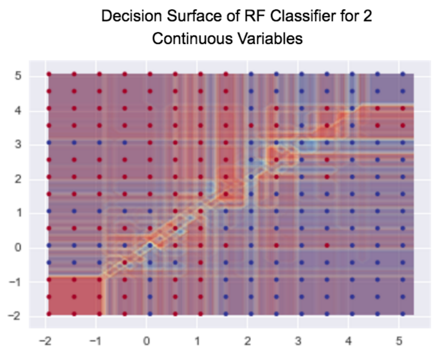


Appendix C. Error analysis









References

Caruana et al. “An Empirical Evaluation of Supervised Learning in Higher Dimensions.” ICML 2008.

Wu et al. “Top 10 algorithms in data mining.” Knowledge and Information Systems. Jan, Vol. 14, Issue 1. 2008.

Henry S. Thompson et al. “The HCRC Map Task Corpus: Natural Dialogue for Speech Recognition.” Humanizing Language Teaching 1993 Proceedings of the workshop on Human Language Technology

Isabelle Guyon, Andre Elisseeff. “An Introduction to Variable and Feature Selection” The Journal of Machine Learning Research. Volume 3, March 1, 2003.

Wind, D. K., & Winther, O. “Model Selection in Data Analysis Competitions.” Meta-learning and Algorithm Selection, 2014

Dong Ying “Beating Kaggle the Easy Way.” July 2015 Masters thesis, TU Darmstadt, Knowledge Engineering Group, Darmstadt, Germany, 2015