Predicting Coreference labels from Spoken Dialogue Tasks with Machine Learning: One-Hot Encoding and Feature Importances using Extremely Randomized Trees

Daniel M. Sheehan DMS2203@COLUMBIA.EDU
Data Science Institute
Columbia University
New York, NY 10027, USA
Kaggle team (one member): Daniel M. Sheehan

Abstract

This paper describes the methods used in developing a machine learning algorithm for the Kaggle competition for the Predictive Modeling Project in the COMS 4721 Machine Learning for Data Science (Spring 2016) Course at Columbia University's Data Science Institute. While the competition demanded an iterative and experimental approach to developing a model/algorithm to achieve a high matching rate against the public and final test data subsets on the Kaggle website, a final methodology, set of code and parameters was required for the final submission and project report write-up. The machine learning methods and techniques that were ultimately employed in this project were (1) one-hot encoding, (2) extra trees classifier with feature importances, (3) limiting the feature columns by only including important features (defined by a feature importance threshold) and (4) then rerunning the extra trees classifier to fit the final model on the limited feature training data which was then used for prediction on the test data.

Keywords: Coreferences, Spoken Dialogue, One-Hot Encoding, Feature Importances, ExtraTreesClassifier

Introduction

Machine learning can be a valuable tool for predicting based on past or known information (Daumé, 2015). The ability to manually explore data, classify and predict is limited by the number of statistical tests a user can program, interpret and finally decide, hopefully through more testing and more data, whether their test outcomes are confirmed or were due to random chance. Computers and more specifically machine learning algorithms, can iterate and test over data routinely to form reliable models that can also be updated as new data becomes available. So long as these machine learning algorithms are designed and implemented to run in a reasonable amount of time (while leveraging the CPU and/or GPU in a distributed and efficient fashion), they are likely to exhaustively test and model more effectively than most analysts could be capable of with conventional inference. While adequately developed and tested machine learning algorithms are available in existing open-source libraries and are easily implemented by the data scientist, the data scientist chooses which methods to use and specifies the parameters. This report explores those methods and decisions for a Predictive Modeling Project in the COMS 4721 Machine Learning for Data Science (Spring 2016) course at Columbia University's Data Science Institute.

Predictive modeling project platform

This Predictive Modeling Project was modeled after and used the Kaggle format of competition, whereby student teams could submit up to five data files per day a set of predicted labels with the test dataset id's (quiz.csv) based on their model/algorithm which was fit from the training data (data.csv). The team then gets a result score for each submission posted on the Public Leaderboard, which calculates the error based on a subset of the test dataset. The teams retain their best score on the Public Leaderboard until the competition ends, at which time the predicted labels for whichever submission the team selects (they may feel that some submissions 'overfit' the subset of the test data and select the submission from the model they believe best predicts against the whole test dataset) are compared against the rest of the test dataset and a new final score is calculated which is used to rank the teams for the final Private Leaderboard.

Task description

The machine learning classifier task is to train a model from…

> …labeled data from a spoken dialogue task in which pairs of speakers engage in dialogues to accomplish a shared objective. Each record in our data set corresponds to a pair of entities mentioned in the dialogue transcripts, described by context information in the form of numerical and categorical features. Each record also has a binary label (either…1 or…-1) indicating whether these entities are coreferences (Hsu, Predictive modeling project website, 2016).

The features in the dataset can be described as…

> …a collection of numerical and categorical features… Each line in the file starts with the feature name, and is followed by either "numeric"…or a list of the possible categorical values for that feature. For example, the feature called "7" (which is the fourth feature in the list) is categorical and can take on two different values ("vf" and "vg"). Also note that some of the numerical features, in fact, take values only 0 or 1 (Hsu, Predictive modeling project website, 2016).

Limitations, organization and assumptions

Since the course materials and this author's understanding of this domain of study, seemingly known as linguistics, is limited, the project was designed without any emphasis on data-specific domain expertise. While domain expertise is sometimes helpful (Mirchevska, 2013), in this instance, in the interest of time and exercising the methods of machine learning, domain expertise regarding the input data was ignored. It is also possible that the author may misdescribe or make false assumptions as the author has absolutely no formal training in computer science or advanced math since high school.

> …[S]oftware that may be used for the project[:] MATLAB and Python libraries: MATLAB default toolbox, MATLAB Curve Fitting toolbox, MATLAB Optimization toolbox, MATLAB Statistics and Machine Learning toolbox, Python standard library, Python cvxopt library, Python matplotlib library, Python numpy library, Python pandas library, Python scipy library, Python sklearn library, Python statsmodels library (Hsu, Predictive modeling project website, 2016).

For organization and clarity, the scitkit-learn machine learning Python programming language module will be referred to as **sklearn**, and the methods will be referred to by their function names within sklearn. It is assumed the reader has some basic understanding of programming.

1. Data preprocessing and feature design

Since much of the data in this this classification task is discrete (noncontinuous), it is important to recode the data into a data model that is able to be processed by machine learning learning algorithms and more correctly models the data in a matrix. Thus, it was determined to use One-hot encoding as a first step in data processing to achieve this.

One-hot encoding is computationally more efficient (Xilinx, 1995) and more accurately models discrete data numerically. One-hot variables, or somewhat similarly, *dummy variables*, are…

> …a simple and useful method of introducing into a regression analysis information contained in variables that are not conventionally measured on a numerical scale, e.g., race, sex, region, … (Suits, 1957).

So on the training data set, one-hot encoding was applied as a function to create a matrix of the data and to widen the table by creating new columns for unique features. The only column that was not included in the one-hot process was the label, which by nature is a feature that only exists in the projects training dataset. This label column was used later for model fitting and is what makes this exercise a supervised machine learning task (Hastie, 2001). The number of columns derived from one-hot encoding was **7,612**. The initial number of columns in the dataset was 52.

2. Model/algorithm description

The author tried many different machine learning algorithms in the sklearn package. Most of the attempted methods were Ensemble methods which…

…combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator (scikit-learn, 2016).

From the author's experience implementing sklearn in the course's homework two assignment, and at the behest of the author's tutor to compare RandomForestClassifier on the same data as was classified with LogisticRegression, Perceptron, BigLogistic, BigPerceptron and two different Gaussian models it seemed like RandomForestClassifier, "a collection of tree-structured classifiers" (L. Breiman, 2001), is a good classifier as it outperformed all of the other methods and beat the next best classifier by 1.6%, which as the author observed over the course of this project's Kaggle competition may be the difference between having one of the top model/algorithms or having a poorly ranked model/algorithm.

Decision Trees are computationally expensive (Daumé, 2015). As will be discussed in section 3. Model/algorithm selection there were several tests with both large and smaller number of estimators or "number of trees in the forest" (scikit-learn, 2016) to determine the number of estimators necessary for prediction.

2.1 Ensemble Methods

Random Forest

Since the author had some experience using RandomForestClassifier it was one of the methods tested most vigorously. RandomForestClassifier is…

…an efficient and surprisingly effective alternative [which] use[s] trees with fixed structures and random features. Collections of trees are called forests, …classifiers built like this are called random forests. (Daumé, 2015)

Extremely Randomized Trees (Extra Trees)

While RandomForestClassifier did consistently well, it did not perform as well as ExtraTreesClassifier.

The Extra-Trees algorithm builds an ensemble of unpruned decision or regression trees according to the classical top-down procedure. Its two main differences with other treebased ensemble methods are that it splits nodes by choosing cut-points fully at random and that it uses the whole learning sample (rather than a bootstrap replica) to grow the trees (Geurts, 2006).

ExtraTreesClassifier also seemed to run faster than RandomForestClassifier and is identified as having computational efficiency (Geurts, 2006).

Other Ensemble Methods

The following other ensemble methods were also tested but did not seem to perform as well as ExtraTreesClassifier or RandomForestClassifier; AdaBoostClassifier and GradientBoostingClassifier.

2.2 Other Methods

The author also tried K-Nearest Neighbors (KNeighborsClassifier) and Stochastic Gradient Descent (SGD - SGDClassifier) with less immediate success on the Public Leaderboard than ExtraTreesClassifier or RandomForestClassifier.

Output averaging and randomization methods on Extra Trees non-similar classifier labels

The author tried a few other methods such as combining the best scores and voting to determine the label and also introducing random votes to the records that the top prediction output files did not have label agreement on, and recursively doing this with the top 3 Public Leaderboard scores. With this method the top Public Leaderboard score achieved was 0.94992 - which, despite being a different method, is the same as the top Public Leaderboard score for the final submission model/algorithm. The author also combined ~80 output files with voting and submitted that too. These all did fairly well but the author felt that both of these methods may have led to 'overfitting' and were not easy to explain and/or to run and thus determined to stick with a more succinct approach.

3. Model/algorithm selection

While the author is not certain about the soundness of this methodology for selecting the correct classifier, during the first initial passes, including one-hot encoding and running the standard sklearn functions while altering a few of the parameters, the ExtraTreesClassifier appeared to perform better than many of the other available methods in the sklearn toolset (see Figure 1). While more attempts were made than the plots shows, it was determined that there was not much of a benefit to using more than 1,000 (n_estimators) for the ExtraTreesClassifier and in fact ultimately reduced this to **400** in the final model/algorithm.

The author also tested the parameters for the number of jobs, which parallelized the computation of the machine learning algorithm across the computer's cores. The computer used for this project had eight cores and the author used six of those cores (**n_jobs=6**) (scikit-learn, 2016).

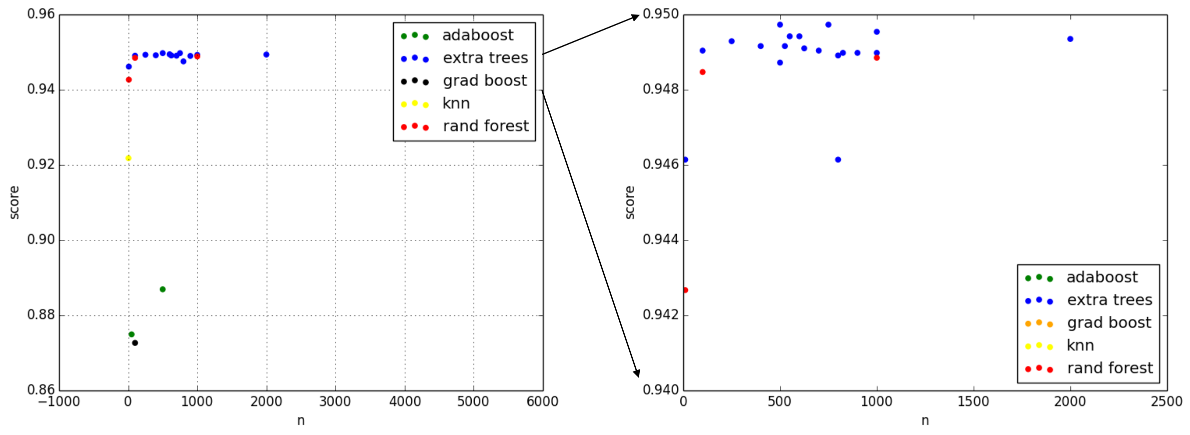


Figure 1. Classifier scores (y-axis), methods (in legend) and number of estimators (n) for the first group of submissions. Note additional submissions are included in the plot on the right and the left plot includes average score for similar n's, rather than the raw data. These plots were for early-on comparison purposes of methods, scores and number of estimators.

Feature Importances

Some of the existing literature and online machine learning and Kaggle message board conversations discuss some methods for machine learning competitions. Some concepts that seemed prevalent were transforming to a lower feature dimensionality, adding features for interaction effects, removing features and adding features iteratively and re-fitting the model(s). Since the sklearn function for ExtraTreesClassifier included attributes for feature importances, it was deemed possible to reduce non-important features within a certain threshold of importance and then re-fit a new ExtraTreesClassifier with only those features that were deemed important (by the threshold of importance). The author tested a variety of thresholds (lowest = 0.000001, highest = 0.001) with many sub thresholds well in between and determined that a feature importance score of **0.00001** seemed to score consistently well. The threshold of 0.00001 for feature importances limited the number of features to ~**2,682** (this will vary slightly each time the model is run), which is ~35% of the original amount of features (7,612) derived from the initial one-hot encoding. The final steps and workflow that was implemented may viewed in **Appendix A** as Appendix Figure A. Diagram of final workflow for algorithm.

4. Predictor evaluation

Throughout selection process for the model/algorithm, the author consistently compared the Public Leaderboard scores and occasionally employed cross-validation (cross_val_score) from the sklearn.cross_validation library. While cross-validation took very long to run (cv=5), it validated if a model would perform consistently well.

Referring to the Public Leaderboard scores likely misdirected the algorithm/model design into ‘overfitting’ territory. In hindsight, more cross-validation should have been performed and programmed into selecting the model parameters.

5. Results of evaluation and analysis

Over the last three weeks of the competition the author’s Public Leaderboard rank dropped severely. At one point early on in the competition the author was in the top ten teams ranked by score in the Public Leaderboard. This rank slowly dropped to fifteen and stayed there for a bit, while occasionally ‘leap-frogging’ with other teams based on their and the author’s submissions. In the last week however, the author’s best submission score was surpassed by many other teams. The author scrambled and tried a bunch of methods in the last week - as mentioned in the section; *Output averaging and randomization methods on Extra Trees non-similar classifier labels*. While the author was able to regain some rank standings and achieve the highest submitted score of **0.94992** with these mixed-methods and voting strategy, it began to generate increasing complexity in the algorithm/model design and explanation and seemed to possibly ‘overfit.’ Thus, the author refocused on feature importances and improving the run time with the number of jobs parameters. Throughout this process the author was able to match a previous high score of **0.94992**, while keeping the model/algorithm less complicated and more straight-forward.

When the Kaggle competition was completed the Private Leaderboard scores were released. The Public “leaderboard is calculated on approximately 50% of the test data” and “the final results [are] based on the other 50%” (Kaggle competition website, 2016). The author was relieved that the lowering in ranking witnessed in the last week of the competition was not due to ineptitude but rather, possibly, due to the other teams in the competition ‘overfitting’ to the public leaderboard.

Kaggle Submissions	Public Leaderboard	Private Leaderboard
Score	**0.94992**	0.94929
Rank	26/37	**19**/37

*Table 1. Score for Kaggle Public Leaderboard and Private Leaderboard. Best Score and best Rank are in **bold**.*

The author’s submission scores did not vary quite so much between the Public Leaderboard and the Private Leaderboard. The high Public Leaderboard score achieved was 0.94992 and the Private Leaderboard score was **0.94929.** More significantly, the observed leaderboard ranking for the Private Leaderboard was much higher (the exact median of **19**/37) than the Public Leaderboard position of 26/37, nearly the bottom quartile of the ranking distribution. It’s possible that teams who had higher Public Leaderboard scores and much lower Private Leaderboard scores had ‘overfit’ their model to the Public Leaderboard test subset.

Conclusions

As stated prior, the author would have like to have cross-validated more often and also have programmed more testing and cross-validation into the overall model/algorithm. Also, since feature importances and lowering dimensionality was not incorporated until the last two weeks of the competition, its possible that more understanding of those techniques would have improved the author’s model and/or algorithm selection. In addition, if the author had less work and travel demands from their employer, it would have been interesting to work with a team.

Overall the author found the exercise of this Practical Modeling Competition to be worthwhile and very engaging. It reinforced the concepts discussed in class and encouraged the author to test a bunch of methods in the sklearn toolbox and read up, in great detail, on the sklearn documentation. It also exposed the participants to the excitement of Kaggle competition and the risks of ‘overfitting,’ as possibly demonstrated in the ‘leap-frogging’ of the final week(s) in the competition.

Acknowledgements

I would like to acknowledge and thank Professor Daniel Hsu and the course Teaching Assistants Edward, Robert, Jun Ho and Siddharth. Most significantly I would like to thank Kenzo, my tutor, for helping me with much of the understanding of the course mathematical concepts and notation, particularly in linear algebra and calculus and helping me become more fluent in better coding practices.

References

L. Breiman. "Arcing Classifiers", Annals of Statistics 1998.

L. Breiman. "Random Forests", Machine Learning, 45 (1), 5-32, 2001.

Hal Daumé III. *A Course in Machine Learning.* TODO. First printing, September 2015.

P. Geurts, D. Ernst., and L. Wehenkel. "Extremely randomized trees", Machine Learning, 63(1), 3-42, 2006.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics Springer New York Inc., New York, NY, USA, 2001.

Daniel Hsu. Predictive modeling project website, 2016. [http://www.cs.columbia.edu/~djhsu/coms4721-s16/project_new.html]

Kaggle competition website. "Columbia COMS 4721 Spring 2016 Competition." Private Leaderboard -. Web. 03 May 2016.

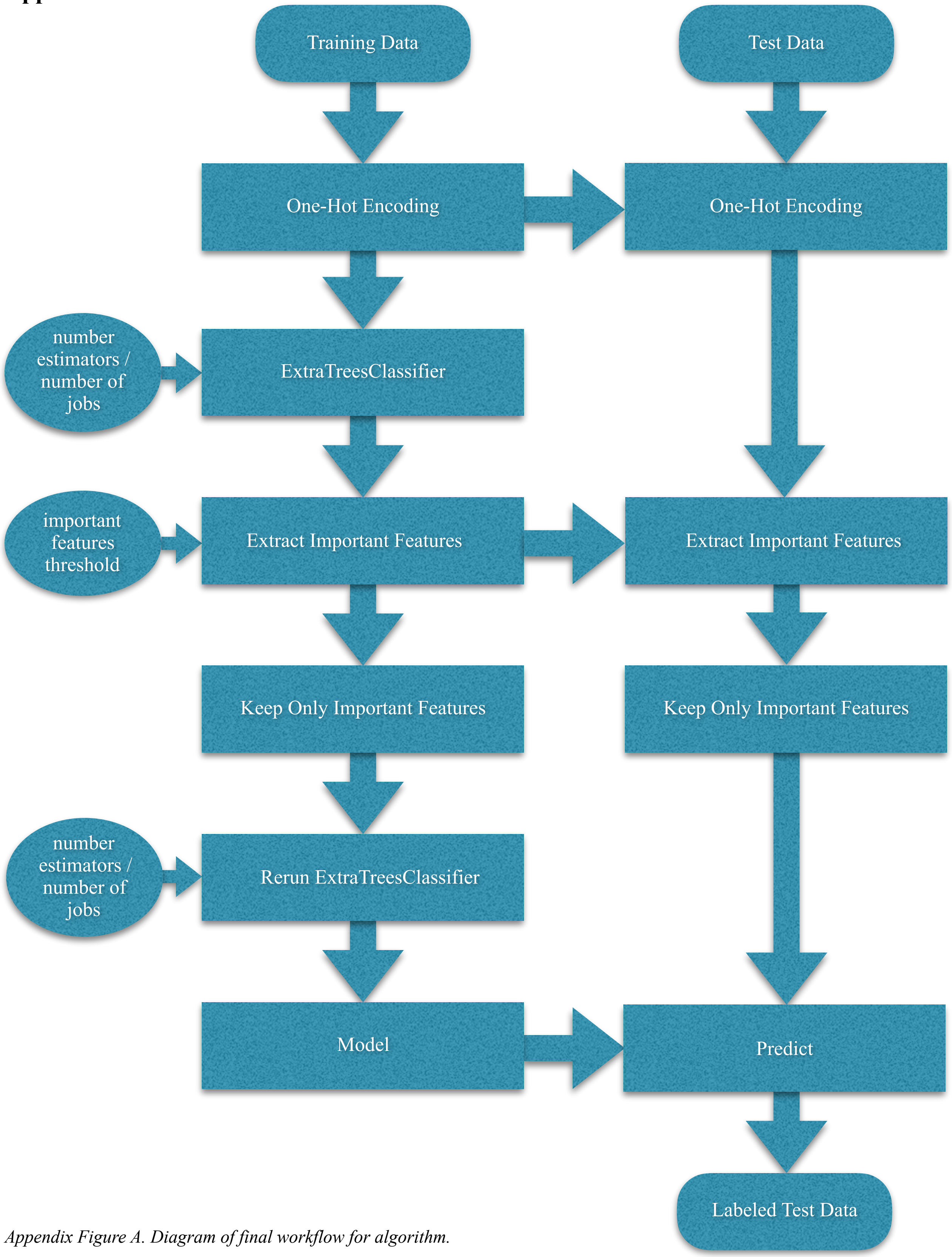
Violeta Mirchevska, Mitja Luštrek and Matjaž Gams. "Combining domain knowledge and machine learning for robust fall detection" DOI: 10.1111/exsy.12019. 2013.

skikit-learn. "Documentation of Scikit-learn 0.17." Documentation Scikit-learn: Machine Learning in Python — Scikit-learn 0.17.1 Documentation. N.p., n.d. Web. 03 May 2016. [http://scikit-learn.org/]

Daniel B. Suits. "Use of Dummy Variables in Regression Equations", Journal of the American Statistical Association. Vol. 52, No. 280 (Dec., 1957), pp. 548-551. 1957. [https://www.jstor.org/stable/2281705?seq=1#page_scan_tab_contents]

Xilinx. "HDL Synthesis for FPGAs Design Guide". section 3.13: "Encoding State Machines". Appendix A: "Accelerate FPGA Macros with One-Hot Approach". 1995. [http://www.xilinx.com/txpatches/pub/documentation/xactstep6/hdlsynth.pdf]

Appendix A.


Training Data
Test Data
One-Hot Encoding
One-Hot Encoding
number estimators / number of jobs
ExtraTreesClassifier
important features threshold
Extract Important Features
Extract Important Features
Keep Only Important Features
Keep Only Important Features
number estimators / number of jobs
Rerun ExtraTreesClassifier
Model
Predict
Labeled Test Data

Appendix Figure A. Diagram of final workflow for algorithm.