Spring 2016 COMS 4721: PREDICTIVE MODELING PROJECT

Alvarado-Guzmán, José A. jaa2220@cumc.columbia.edu
Bailey-Assam, Jade jeb2245@columbia.edu
Rusnak, Christopher cjr2176@columbia.edu

Editor: None

Abstract

This paper describes the data processing, feature reduction, Random Forest model selection trough cross validation and finally presents the model with minimal prediction error on the HCRC (Human Communication Research Centre) Map Task corpus dataset. This study was conducted on a Spring 2016 Machine Learning course (COMS 4721) predictive modeling project context, where the final predictions were submitted to Kaggle for students competition. We attempted to reduce the feature dimensionality by applying a combination of descriptive statistics and feature research for a manual feature selection and Lasso Regression and PCA as reduction algorithms. The manual feature selection turned out to be most successful for this task. After running a cross-validated random search our final model consisted of a Random Forest with 167 decision trees that obtained a prediction accuracy of 93.9% on the holdout dataset, 93.7% and 93.9% on the public and private Kaggle validation datasets respectively.

1. Introduction

Our dataset was derived from experiment results from the HCRC (Human Communication Research Centre) Map Task corpus (Thompson et al., 1993). In this experiment, 128 pairs of people were each given a map, one with a route and one without. The task was for one person with the map that has no route, the Instruction Follower, to draw that route based on discussions with the person with the map that has the route, the Instruction Giver. Among the independent variables that were measured in this experiment were whether or not the participants were making eye contact; whether or not the participants knew each other beforehand; and the role of each participant in the experiment (Instruction Giver vs. Instruction Follower). Their conversations were recorded and broken apart into over 150,000 individual words. Those words, their form of grammar, and the experiment's independent variables were used as features for the dataset. The binary label for the dataset is whether or not a pair of items mentioned by the participants in part of their conversation are co-references. That is, are both participants referring to the same thing? This question is a useful one to answer for demonstrating how much progress each pair of participants makes in completing its task. Creating a binary classifier for this dataset would hopefully show which conditions are necessary for a successful, cohesive dialogue.

2. Data Preprocessing/Feature Design

Before analyzing the data, we needed to perform some initial preprocessing and feature engineering to aid us down the line. We decided to exclude features from our training data set in an effort to reduce the size of our set to make it more manageable for us to analyze.

We created a Python class to read the data files into a .py data structure, maintained a dictionary with the distribution of each feature and normalized the data. Based on the aforementioned individual feature distributions and on further research we were able to reduce the size of the data set by excluding several features from both the training and test data sets.

Certain features were excluded because they either didn't tell us anything useful (ie all entries were 1) or they were redundant. An example of a feature that doesn't tell us anything is a feature where all entries have the same value. An example of a feature that is redundant is one that is represented in multiple features throughout the data set. We only need to account for this data once in our model. There were 52 features in our original training set (+1 column of labels). We dropped the following 14 of the 52 original features, leaving us 38 feature vectors to describe the training data set.

- features 0 & 9: concatenation represented in feature 18
- features 2 & 11: concatenation represented in feature 20
- features 7 & 16: concatenation represented in feature 25
- features 56 & 57: concatenation represented in feature 58
- features 5 & 14: concatenation represented in feature 25
- features 29, 31, 32, 35: all entries were 1. There's nothing much we can learn from this

We applied one-hot encoding in both the training and the testing sets to the following features: 8, 17,18, 20, 23, 25, 26 and 58. Since these features take on values that are present in one dataset but not in the other, we removed the binary features of values that were not present in both the training and the testing datasets. This ensured parity amongst both data sets; essentially we could make an 'apples to apples' comparison between training and test data.

Principal component analysis was also attempted in one of two ways:

- Run PCA on the training dataset, preserving features whose subspace contains a fixed proportion (80% or 90%) of the data's variance, and then build a predictive model from that reduced dataset.
- Using Python's scikitlearn pipeline method, choose the predictive model hyperparameters and the dimension of the PCA-reduced feature subspace via cross validation on the training dataset.

In both cases, we successfully reduced the dimension of the feature subspace of the holdout dataset by applying the same PCA from the training dataset (i.e., the PCA-reduced training dataset and the PCA-reduced holdout dataset had the same number of features). However, fitting the same PCA feature mapping from the training dataset to the testing dataset resulted in the testing dataset having one additional column than the PCA-reduced training set. Since we could not determine which feature, if any, was extraneous, we decided to not follow through with PCA as a method of feature reduction.

Lasso Regression with a ten fold cross validation was attempted on a normalized version of the data, including binary conversion of categorical features, but only three features had coefficient greater than 0 so we decided not to utilize this algorithm as a dimension reduction either.

3. Model/Algorithm Description

We used Python's scikit learn package to try a random forest classifier, a type of ensemble method that uses bagging. Random forest classifiers are composed of multiple decision trees. These trees are nonparametric models which are constructed using a greedy algorithm that maximally reduces Gini index uncertainty (the default criterion in scikit learn) at every split. The stopping criterion is when all leaves are pure (default scikit learn parameter value). Each split of each tree occurs over a random subset of $\sqrt{d}$ fields (default scikit learn parameter value), where d is the total number of fields in the dataset. The random seed was set at a specific value of 42 so that the results of the model were reproducible. For each data point, the predicted class label was chosen by majority vote over all of the decision trees.

4. Model/Algorithm Selection

The number of decision trees to include in the random forest was a hyperparameter to be chosen via 10-fold cross-validation. We chose 30 different possible integer values, uniformly at random, between 5 and 200. In general, grid search over model hyperparameters is computationally wasteful and not likely to choose optimal parameter values. Instead, we optimized this value using cross-validated random search over the hyperparameter space (James et al., 2012), and selected the model with the highest overall binary classification accuracy rate. This model was composed of 167 decision trees.

5. Predictor Evaluation

The original dataset was randomly partitioned into two sets: 70% of the data was used for training data and the remaining 30% was used as the holdout dataset. The cross-validated random search was applied over the training data. All models were evaluated using the overall accuracy rate for binary classification:

$$\frac{TP+TN}{TP+FP+TN+FN} = 1 - errorrate, where:$$

TP = number of true positives (total coreferences correctly predicted),
FP = number of false positives (total non coreferences incorrectly predicted),
TN = number of true negatives (total non coreferences correctly predicted),

and FN = number of false negatives (total coreferences incorrectly predicted).

The model with the optimal hyperparameter values from the random search was determined by choosing the model with the highest accuracy rate. That model was fitted on the training set by using those optimal hyperparameter values and then evaluated on the holdout dataset.

6. Results of Evaluation & Analysis

The accuracy rate on the holdout dataset was 93.92%. Our performance on the public data points from the quiz set was 93.7%, while the model had a score of 93.9% on the private data points. In general, ensemble methods work well because they aggregate the predictions of multiple models. This setup allows for the insights of each model (i.e., the correct predictions) to be represented in a single predictor. An ensemble method also has lower model variance than its component models. In particular, bagging not only reduces model variance (Daume, 2015, chap. 11), but also reduces the model error by a factor of M, the number of bootstrap samples, assuming that the component models have uncorrelated errors (Bishop, 2012, chap. 14). As such, a random forest would be less sensitive to data perturbations then a single decision tree (Daume, 2015, chap. 11). Further, since each of the decision trees of a random forest are split over a random subset of features; each of these trees are constructed independently; and a large number of trees composed the final model, it is much more likely that there are more individual trees of the forest that are trained on a good combination of features for more accurate predictions.

7. Future Directions

We feel there are future opportunities available for this analysis. For data preprocessing, we would perform additional attempts at Principal Component Analysis. Additionally, we would like to implement some ensemble methods, namely AdaBoost as well as look at more trees in random forest (although this may have resulted in overfitting). Lastly, we would have liked to resolve some of the missing values in the data set (throwing them out, nearest neighbors, correlations, etc.)

8. Individual Contributions

- Alvarado-Guzmán, José A. - conducted the Lasso Regression and created a Python class that takes several arguments like the file location, delimiter, header, numeric, columns to convert to binary, ext. as an input and provide access to the following data structures:
  - List of list and Dataframe containing the original data contained on the file.
  - Numpy 2D array containing the numeric features and the categorical features converted to binary.
  - Numpy 2D array containing numeric features normalized and the categorical features converted to binary.

  - Numpy 2D array containing the numeric features normalized and the categorical features normalized as well.
  - Numpy 1D array containing the labels.
  - Numpy 1D array containing the labels normalized.
  - Dictionary where the keys are the features and the values are a list containing frequency and relative frequency for categorical variables and min, max, median, mean and quartiles for the numeric variables.

- Bailey-Assam, Jade - Conducted research around interpreting the data and helping the team understand what it meant. This helped us significantly with feature reduction. I figured out that the data set represents conversations between two people describing a map's route, which helped us understand that the features with 'f' & 'g' entries represent the person's role in the conversation (instruction follower or giver). By analyzing images of the maps from the experiment I was able to understand the features with '0' & '1' entries indicate whether or not a specific landmark was present on a certain map.

- Rusnak, Christopher - Applied principal component analysis as a data pre-processing step. I outlined the post-processing experimental design (partitioning the data, cross-validated random search over random forest model parameters, final model selection, model evaluation, outputting the predictions), as well as the code for doing so. I also setup a private Github repository to organize our scripts, datasets, resources, and notes.

References

James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research, 13:281-305,2012.

Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer Science+Business Media, New York, New York, 2006.

Hal Daume III. A Course in Machine Learning, v. 0.9,2015.

Henry S. Thompson, Anne Anderson, Ellen Gurman Bard, Gwyneth Doherty-Sneddon, Alison Newlands, and Cathy Sotillo. The HCRC Map Task Corpus: Natural Dialogue for Speech Recognition. In Human Language Technology: Proceedings of a Workshop, ARPA/ISTO., pages 25-30,San Francisco and Washington DC, 1993.