PROJECT REPORT

Project Report : Kaggle in-class Competition (COMS 4772 Spring 2016)

May 5, 2016

Team Name : BladeRunners
Members : Nitish Ratan Appanasamy, Vishal Juneja, Rohit Bharadwaj
Columbia University
COMS 4772 : Machine Learning for Data Science

Contents

1 INTRODUCTION

In this report we briefly describe our approach towards the problem of predicting entity coreference using labeled data from a spoken dialogue task. The features in the given dataset are a mix of categorical and numerical features. We approached this problem by first analyzing the features, referencing existing research for effective feature extraction [1] and conducting several experiments with different classifiers and combination of features to come up with an effective high performing model. In the next sections, we describe our feature and data pre-processing followed by model selection, final algorithm and results.

2 DATA PRE-PROCESSING AND FEATURE DESIGN

This section details the various methods employed by us in pre-processing the raw data and in generating the new features. During the feature analysis we found several interesting observations of the existing features, which we manipulated to boost the accuracy.

2.1 Encoding Categorical Variables

Categorical variables were encoded using two methods to effectively leverage the information present in them. In the next two sub sections, we briefly describe feature encoding techniques we employed.

2.1.1 LabelEncoder

This in-built converter in Python SkLearn is used to convert the categorical variables directly to numeric values based on an unique in built function map. However a major drawback of this method is that this encoding introduces a numeric relationship/similarity between the different variables of a given feature which might not exist. For eg. 'Yes' might be encoded to 1 and 'No' to 4.5 but however these numeric values imply that the 'No' is 4.5 times the value of variable 'Yes'.

2.1.2 One Hot Encoding

One Hot Encoding or one versus k was the other method employed in encoding categorical variables. However one of the drawbacks of employing this method was that employing One Hot Encoding on the raw data lead to an explosion in the number of features as a few categorical features in the data set had very large number of variables (number of variables >

1000). This resulted in an explosion of the total number of features which greatly increased run-time and decreased accuracy. We performed one hot encoding using Python SKlearn library. In the next section we explain our approach for feature reduction and selection which played a crucial role in boosting the accuracy.

2.2 Feature Reduction

It was identified that the categorical features having a very large number of variables were in fact combination of two other categorical features present in the data. Removing such redundant features reduced the total number of features after One Hot Encoding to 500.

On further experimentation with the dataset using feature selection methods (Lasso, Forward, ExtraTree Classifier feature importance) we realized that many variables within certain categorical features had very low frequency and therefore One Hot Encoding of that variable for that feature gives a highly sparse binary vector. In order to prune the features further we selected few sparse features with frequency below an empirical threshold and merged them to create a single Binary feature where the value was set to one if any one of the features is set for an instance. Another interesting observation we made is regarding few features for which the values were constant across the dataset (0 standard deviation). However, removing them reduced the cross-validation accuracy. We haven't performed more analysis but left them in the dataset unchanged.

Henceforth all referrals to the Data Matrix will refer to the matrix obtained from using One Hot Encoding and removal and manipulation of the features as specified in this section.

2.2.1 Principal Component Analysis

Using PCA in python we calculated the principal components of the data matrix excluding the feature columns '59' and '60' and any other features derived from them. We then transformed the data using these principal components for dimensionality reduction.

2.2.2 Non-negative Matrix Factorization

Non-negative Matrix Factorization is a decomposition that tries to approximate the input non negative matrix (all entries are non-negative) as a product of two matrices W and H obtained by minimizing the objective function as given in equation [1].

$$\begin{aligned} 0.5 * ||X - WH||_{Fro}^2 + \alpha * l1ratio * ||vec(W)||_1 + \alpha * l1ratio * ||vec(H)||_1 \\ +0.5 * \alpha * (1 - l1ratio) * ||W||_{Fro}^2 + 0.5 * \alpha * (1 - l1ratio) * ||H||_{Fro}^2 \end{aligned} \tag{1}$$

Here the Fro represents the Frobenius norm. The matrix W obtained from nmf is sparse. If the input matrix X is a document term matrix such that the the rows represent the words and the columns the documents then the matrix W can be interpreted as a sparse compilation of the different topics in the document and the matrix H as the matrix that contains the membership information for each document. We used Python SkLearn library to achieve this to find effective features.

2.3 Further Methods

For feature generation some of the other methods that were explored were BinaryICA/FastICA and Sparse Filtering. The idea behind exploring BinaryICA was it would help decorelating the signals and the idea behind Sparse Filtering was that it would help in achieving a sparse representation of the data matrix while optimizing for population sparsity and enforcing high dispersal as described in the paper [4]. This method is also specially effective for over complete representations. Out of these two methods, Sparse Filtering based classification used in our Blending procedure along with nmf generated features and PCA generated features showed the most promise. However since Sparse Filtering was not available in Sklearn and was available only from a third party developer we could not implement it on our own in time to be able to submit it for the competition. Using only the One-Hot Encoded categorical features of Data Matrix we applied nmf to obtain the H matrix which was used as a new set of features.

2.4 Polynomial Features and Binning

We experimented with introducing polynomial features for each of the numeric features we had and using feature selection we determined that creating polynomial features for the features '59' and '60' were the most effective. For these two features we created quadratic terms, square root, logarithmic and also used binning to create new features.

2.5 Training and Testing Datasets

In order to ensure consistency in transformations we processed the entire Data set (including Training data as well Test data) together. This was done in order to prevent any differences in the transformation of the training data and test data if preprocessed separately. For example if the right singular vectors obtained from the SVD decomposition of the training set might not be the same as one for the test set, hence for consistency sake, we performed all

transformations on the dataset formed by combining training and test sets.

3 MODEL/ALGORITHM

Several candidate algorithms such as SVM, Logitstic Regression, LDA, QDA, and Linear Regression with Elastic Nets were tried and tested. However the algorithms that performed best using cross-validation and hold-out set accuracy were 'RandomForests', 'Adaboost' with different base classifiers, and 'ExtraTree Classifier'. Our final model was based on a combination of these Algorithms using linear stacking or blending based on [2] and [3].

3.1 Final algorithm

We created two sets of data, each one of the data sets were processed exactly as described in the pre-processing section but with one data set having the PCA generated features and other data set having nmf generated features. The two Data set after pre-processing was divided into the Training set and Hold-out set. The Hold out set was set to be $\frac{1}{5}$ of the total training data set. The two training sets were then blended using the procedure as detailed below.

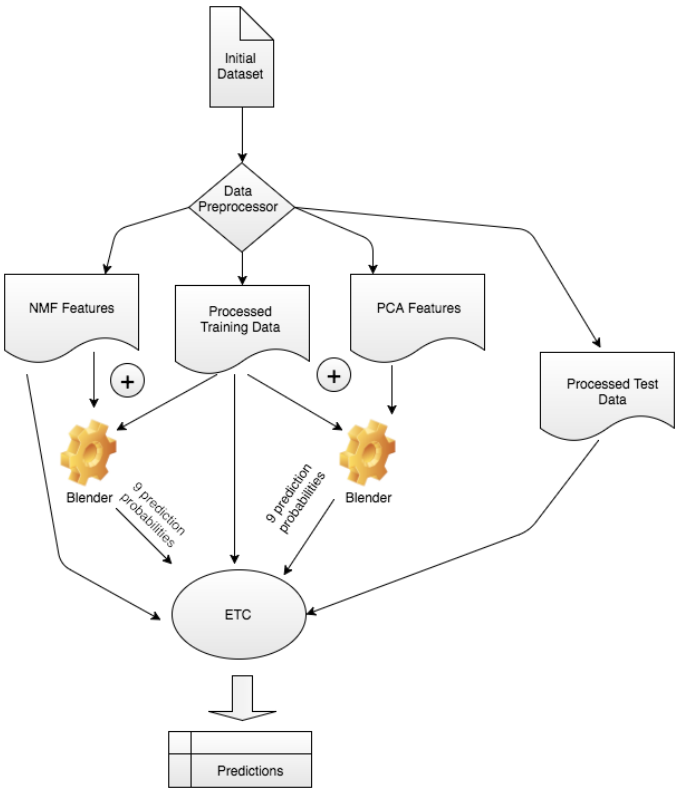


Figure 1: Final Algorithm

Blending Procedure

The blending procedure is a multi level prediction method consisting of two levels. level - 0 of the blending procedure is a set of Base Classifiers comprising of Adaboost, Extra Tree Classifiers, Gradient Boosting Classifier, LDA, QDA, KNN. The training data set is first divided into k folds. In order to produce the level-0 predictions for a

particular k^{th} fold of the data each of the level-0 classifiers were trained on the remaining $k-1$ folds of the data set (i.e all the training data except the fold for which predictions were being made). All level-0 classifier's prediction probabilities (i.e probability that data is labeled as '1') for all the k folds of the data were then used as new features. This above given generation of features were performed on both the training sets i.e the one having PCA generated features and the one having nmf generated features. These new features were then appended to the initial data set. The idea behind generating all these extra features were to provide the ml algorithms new perspectives on the data to learn from. This new merged data set was used to train an Extra Tree Classifier (with parameters set to 300 trees) to predict the final labels for the test data set. For replication of results the random number generator function in python was used. The figure below is a flowchart of our final algorithm.

4 MODEL/ALGORITHM SELECTION

The algorithms were chosen based on their performance on the Hold-out sets, Cross-Validation scores, and the scores on the public leader board. In general the performance of algorithms on the Hold-out set, the public leader board and on Cross-Validation closely matched each other.

Our extensive usage of Random Forests and Extra Tree classifiers on multiple stages of our blending procedure was predicated on our understanding of the high robustness of random forest type classifiers against over-fitting [5]. Our choice for the set of classifiers used for the level-0 of the blending procedure were solely based on our earlier experiments on which base classifiers gave the highest test and hold out scores and also based on the need to have distinct classifier types such that the level-0 results produced were uncorrelated with each other. This non-correlation of the level-0 predictions is important as these features serve as the input for the level-1 classifier.

5 PREDICTOR EVALUATION

Predictor Evaluation was performed using a Holdout set of size $\frac{1}{5}$ of the Data set. The individual base classifiers used were chosen based on their Hold Out scores, Cross- Validation scores and their public leader board scores.

6 RESULTS, ANALYSIS AND DISCUSSION

We initially began our analysis by trying out different basic classifiers such as Logistic Regression, SVM, LDA, QDA following which we moved to ensemble based classifiers like Adaboost, ETC, Random Forests. For these ensemble classifiers we ran a hyperparameter optimization loop with the hyperparameters being optimized were the number of base classifiers being combined, the depth of the trees (if trees where the base classifiers) and the learning rate (see Appendix for tuning results). For the Adaboost algorithm different base classifiers were tried including SVM, Logistic and Linear Regression, Trees, out of which Trees gave the best results. Among different ensemble based classifiers, Extra Tree Classifiers were the best. Based on these results and also in order to reduce correlation, different level-0 predictors for our blending algorithm were picked. Besides blending, linear stacking was also experimented with. In the table below, we present results of few models that we have experimented with. We tweaked our models based on the methods describe above which increased our scores gradually.

	algorithm	public	private
1	Simple Logistic Regression	0.62706	0.63202
2	Random Forest	0.89631	0.89239
3	Extra Tree Classifier (ETC)	0.92135	0.91611
4	Blending after removing redundant features	0.94122	0.93768
5	ETC after removing redundant features	0.94834	0.94702
6	ETC with polynomial and quantization of 59,60 cols	0.94916	0.94872
7	ETC removed sparse features after one hot encoding	0.95162	0.94992
8	ETC merged sparse instead of removing	0.95408	0.95194
9	Adaboost (with ETC) + log, sqrt, division features of 59,60 cols	0.95427	0.95194
10	ETC (300 instances) + Blended NMF features	0.95560	0.95332
11	ETC + Blended NMF and PCA features	0.95572	0.95446
12	Voting of top 3 classifiers	0.95585	0.95446

7 CONTRIBUTIONS

For the initial few days, every member of the team was experimenting on his own to perform feature analysis and identify better performing algorithms. After the initial set of experiments, we zeroed in on few methods where we took turns to implement the method and run the model. Below, we describe the contributions of each team member for the project. However,

everyone was also involved in analysis, brainstorming and code re-factoring/management.

Rohit Bharadwaj: PCA generated features, Label Encoding, Adaboost, Generalized Stacking, Preprocessing training and test data together, BinaryICA/ Fast ICA.

Vishal Juneja: nmf generated features, One Hot Encoding, Extra Tree Classifier, Voting, Introducing polynomial features and binning.

Nitish Ratan: Removal of redundant features, Removing and merging low frequency variables, Blending, Hold out, Sparse Filtering.

REFERENCES

[1] David Kofoed Wind *Concepts In Predictive Machine Learning* http://www.davidwind.dk/wp-content/uploads/2014/07/main.pdf

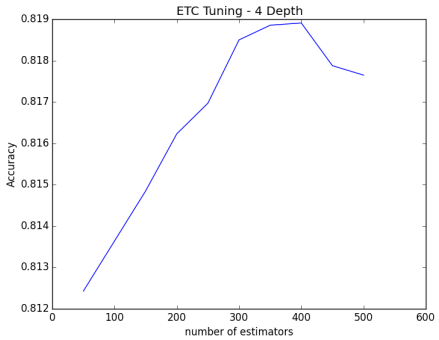
[2] David H. Wolpert. *Stacked Generalization.* Neural Networks 5(2):241-259 Âů December 1992.

[3] Kai Ming Ting, Ian H. Witten *Issues in Stacked Generalization.* Journal of Articial Inteligence Research 10 (19) 271-289.

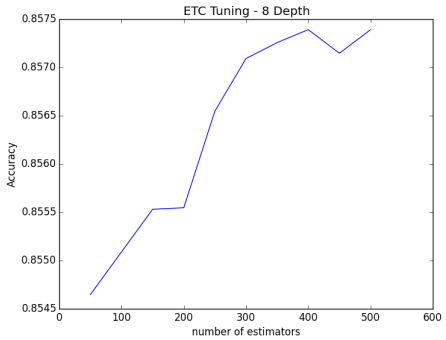
[4] Jiquan Ngiam, PangWei Koh, Zhenghao Chen, Sonia Bhaskar, Andrew Y. Ng *Sparse Filtering* https://papers.nips.cc/paper/4334-sparse-filtering.pdf.

[5] Leo Breiman *Machine Learning, 45, 5âĂŞ32, 2001, Kluwer Academic Publishers.*

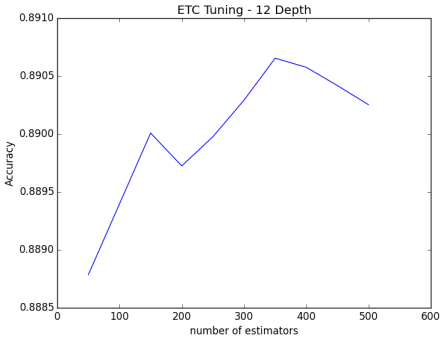
APPENDIX



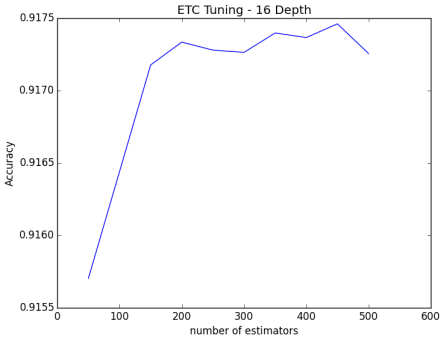
(a) depth = 4



(b) depth = 8

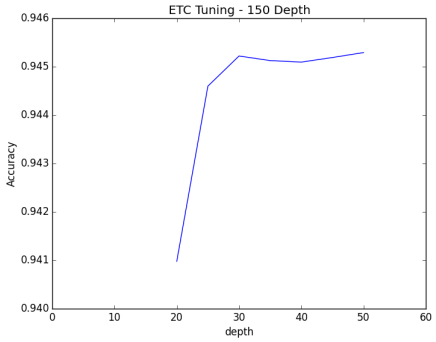


(a) depth = 12

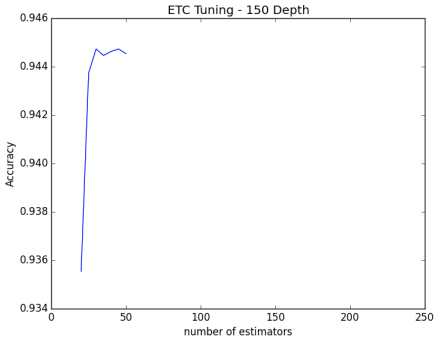


(b) depth = 16

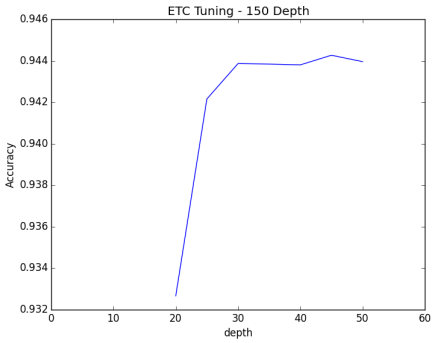
Figure 3: Tuning for number of estimators in ETC



(a) all features



(b) square root features



(c) log2 features

Figure 4: Tuning for depth and feature subset with 150 estimators in ETC