Machine Learning Engineer Nanodegree

Capstone Final Report

Michael Basca March 5th, 2017

I. Definition

Project Overview

The practice of predictive modeling defines the process of developing a model in a way that we can understand and quantify the model's prediction accuracy on future, yet-to-be-seen data. In short, predictive modeling is the process of developing a mathematical tool or model that generates an accurate prediction (Kuhn and Johnson 2013). Prediction models are used in a wide range of industries such as finance, healthcare, public policy and others.

Polling is an integral part of our political system. A poll is a survey or questionnaire used on a random sample of people in hopes that the random sample is representive of a large population. In assuming that the sample is representative, polling minimizes the resources required to gather data by only using a fraction of the population. Polling can be asset in the realm of politics because they gauge the views of the public on the current political issues. Politicians can use polling information along with predictive modeling to cater their message to the constituencies that they are trying to influence, in order to get elected or aid their decision making in governance.

The history of using machine learning in polling is limited. Some of the literature that was researched have modeled political affiliation based on text sentiment analysis of tweets or political speeches [1, 2]. The text can be processed into a bag of words vector that closely resembles binary features for this project. A variey of supervised linear and non-linear learning algorithms were used (Support Vector Machines, Logistic Regression, Linear Discriminant Analysis and Naïve Bayes) as well as unsupervised methods as well(Principle Component Analysis, K-Means Clustering). Another experiment details the use of Decision Trees to predict party affiliation of Congress members by analyzing how each member voted on particular legistlation [3, 4]. These yes or no votes also reflects the structure of the binary data that is used for this capstone.

Problem Statement

In this project I will be participating in a **Kaggle**[5] project that uses data from a mobie application called ***Show of Hands***[6]. In this competition ***Show of Hands***[TM] provides a questionnaire to thousands of data in a form of approximately one hundred **Yes** or **No** questions along with their political affiliation (**Democrat** or **Republican**). I will use this data to create a model that predicts political affiliation based unlabeled test data.

The problem is to predict whether the person answering the questionnaire is either a **Democrat** or **Republican**. Kaggle has the actual test labels to compare your predictions with and will give you a score based on accuracy. The goal is to achieve an accuracy that acheives a score at 75 percentile (or more) of all participants.

This problem is a classification task. While there are many learning algorithms (linear and nonlinear) that can produce a classification model, the focus of this assignment will be the preprocessing of data that will enhance the signal while reducing the noise to aid in proper generalization. I will perform vizualizations of the data that will guide which processing will be required.

The preprocessing will include:

- Transformations
  - MinMax scaling
  - Box Cox Transforms
- Feature Extraction and Dimensionality Reduction
  - Tree Feature Importance
  - L1 Regularization
- Dealing with missing Values
  - Deletions of Samples via threshold
  - Imputation
- Create meta-features
  - additive/multiplicative features

There are a variety classification learning models that we can choose from:

- Linear Models
  - Logistic Regression
  - Linear Discriminant Analysis
  - Partial Least Squares Linear Discriminant Analysis
- Non-linear Models
  - Support Vector Machines

    - Decision Trees
- Ensemble Models
    - Random Forests
    - Boosting

Each model has their strengths and weaknesses. In general I will explore common models with the data minimal processing to provide a baseline score. I will then perform extra processing (feature reduction as well as feature creation) to see whether the score can be improved.

Metrics

While classification models can be evaluated in a variety of metrics (AUC, sensitivity, specificity, kappa, etc.), Kaggle will evaluate our predictions based on accuracy. We may use the previous metrics a guide to provide better accuracy. Accuracy is defined as:

$$\frac{TP + TN}{N}$$

Where:

- **TN** are Number of True Negatives
- **TP** are Number of True Positives
- **N** are Number of Samples

An explanation of why accuracy makes sense is best provided by a counter example:

Predicting rare disease - Let' assume that the goal is to accurately diagnose/predict a rare disease based on information given by the patient which will be used as predictors. There are two type of errors that we can make: - False Positives - Identifying a person that doesn't have a disease as having the disease. - False Negatives - Diagnosing a person that does have a disease as healthy.

Let's assume that if we diagnose a person that has the disease, that person would just go through extra tests and no additional hard would occur for that person. We also assume that not correctly diagnosing the person with a disease results in death. Based on this we clearly value avoiding False negatives and are willing for our model to identify all the people with the disease (increasing recall) with sacrificing diagnosing healthy people incorrectly (decreasing precision). In this case we care more about sensitivity rather than accuracy which places equal focus on both false positives and false negatives.

In our project we value both parties equally (A democrat is not more valuable than a Republican and vice versa). Also the training data exhibits balanced classes, hence accuracy is a suitable metric.

II. Analysis

Data Exploration

A data set of 6960 samples were gathered from ***Show of hands***. The data included the following features:

1. Date of birth - an Interval variable
2. Gender - a Nominal/Binary variable
3. Income Bracket - an Ordinal Variable
4. HouseHold Status - a Nominal Variable
5. Educational Level - an Ordinal variable
6. One hundred and one **Yes** or **No** questions - a Nominal/Binary Variable

The outcome is: 1. Party affiliation **Democrat** or **Republican** - a Nominal/Binary Variable

Kaggle has provided csv files for the training set with labels as well as the testing set (80/20 split) without labels to test your model against.

Here is a portion of the data frame:

Figure 1 - Partial slice of the DataFrame

	YOB	Income	EducationLevel	HouseholdStatus	Gender	Interact.with.someone.dislike.daily	Parents.Fight.i
USER_ID							
1	1938.0	NaN	NaN	Married (w/kids)	Male	No	NaN
4	1970.0	over $150,000	Bachelor's Degree	Domestic Partners (w/kids)	Female	NaN	Yes
5	1997.0	$75,000 - $100,000	High School Diploma	Single (no kids)	Male	NaN	Yes
8	1983.0	$100,001 - $150,000	Bachelor's Degree	Married (w/kids)	Male	No	Yes
9	1984.0	$50,000 - $74,999	High School Diploma	Married (w/kids)	Female	No	Yes

It was decided to remove samples where participants were born before 1933 (some samples were stated that their YOB was 1900 which indicated that the sample's data validity was questionable) as well as participants born after 2000 where the person might be too young to comply with some of the questions or answer them seriously. The result was a loss of 7% of the samples that were considered outliers.

Here are some example statistics for some selected feature columns after the removal of these outliers:

Figure 2 - Descriptive Statistics

	YOB
mean	1979.63
std	14.95
min	1935
25%	1970
50%	1983
75%	1993
25%	1970
max	1999

Income	count
Income	**count**
under $25,000	729
$25,001 - $50,000	692
$50,000 - $74,999	805
$75,000 - $100,000	714
$100,001 - $150,000	744
over $150,000	701

EducationLevel	count
Current K-12	720
High School Diploma	662
Current Undergraduate	745
Associate's Degree	366
Bachelor's Degree	1162
Master's Degree	6150
Doctoral Degree	183

Gender	Count
Male	3112
Female	1984

Work.Min.Wage	count
Yes	196
No	2906

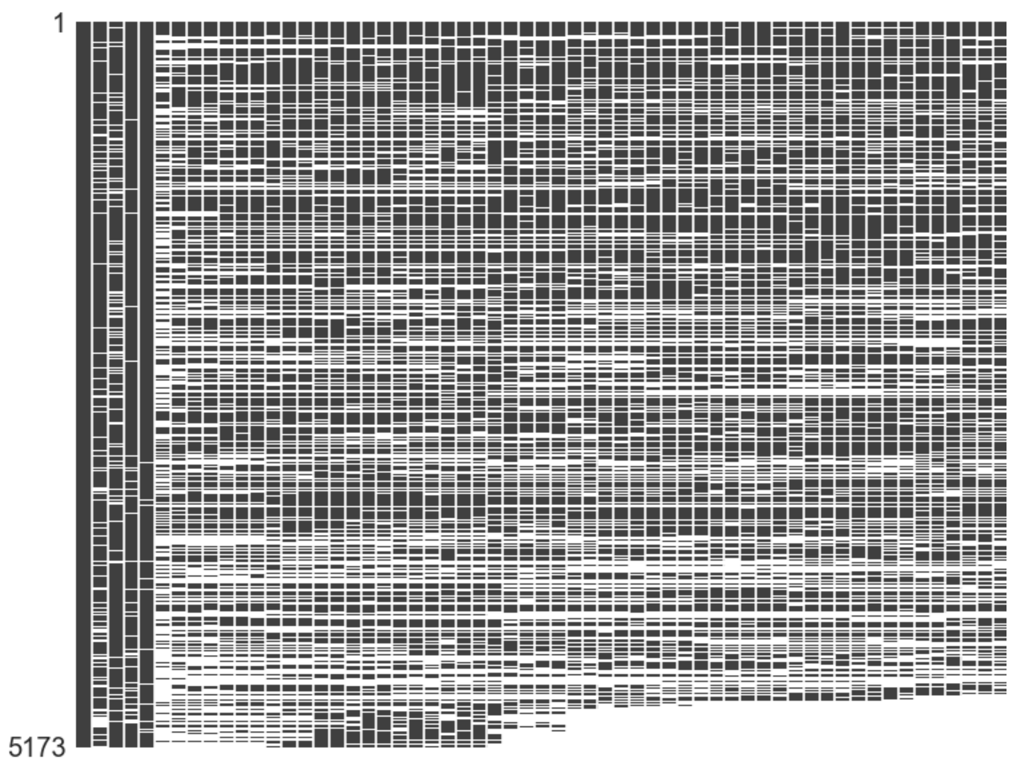
The target breakdown is the following:

Party	count
Republican	2423
Democrat	2750

Which indicates relatively balanced classes.

The data is somewhat sparse. Let's look at this graphically:

Figure 3 - Sparsity Matrix of the DataFrame

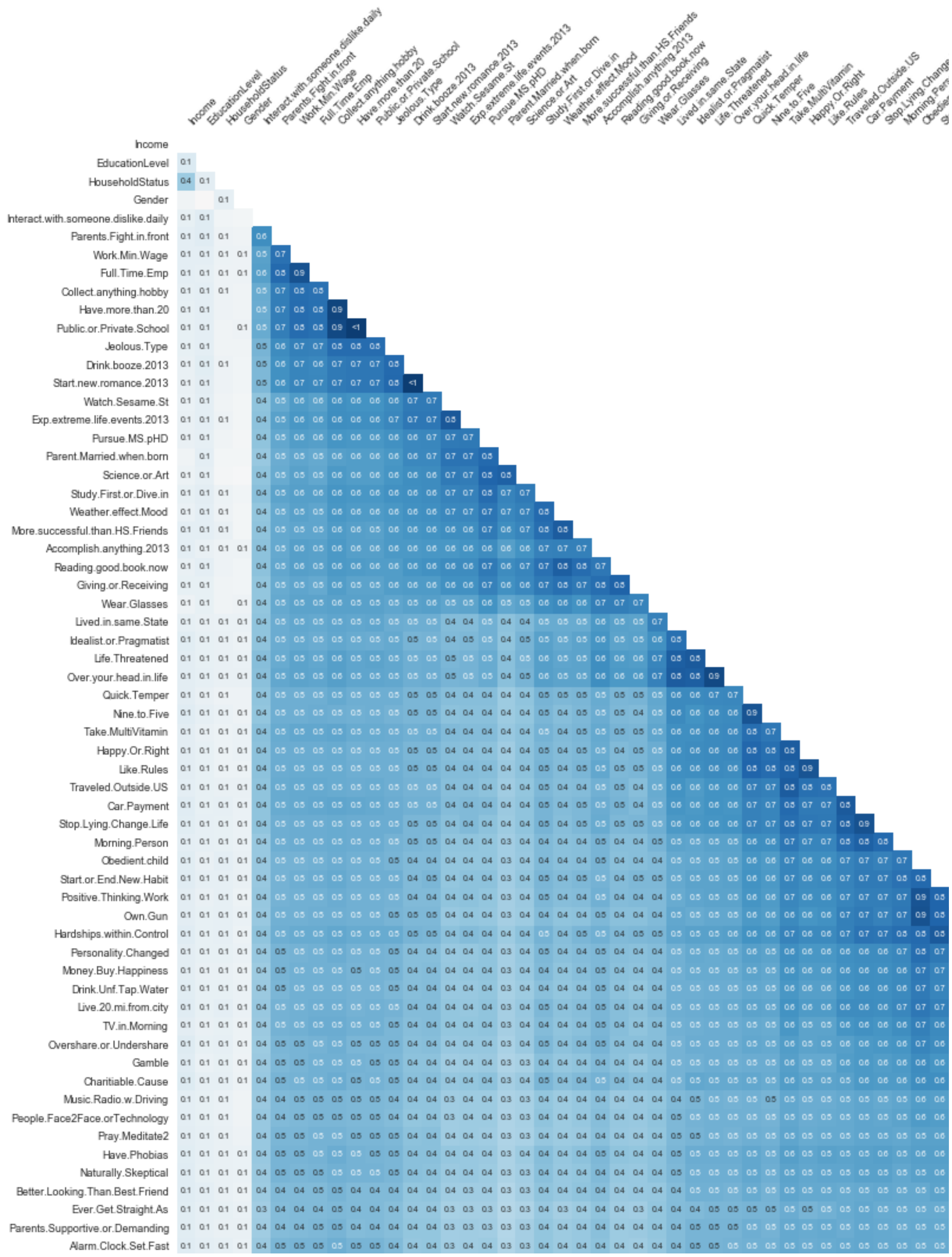


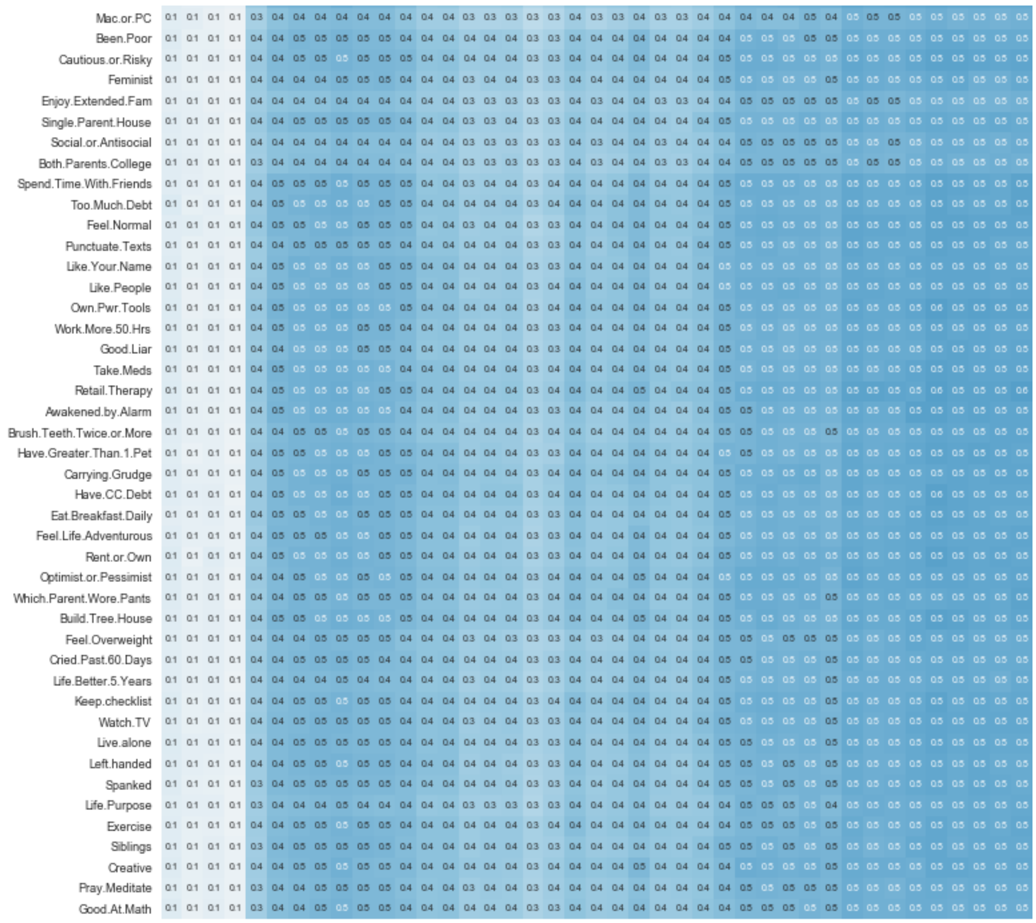
We will devise a method of deleting more samples that have a minimum sparsity threshold with imputation on the rest of the samples.

Exploratory Visualization

Let's take a look at the correlation plot.

Figure 4 - Correlation Matrix





Using a threshold of +/- 0.7 we see that there are very few features that highly correlated.

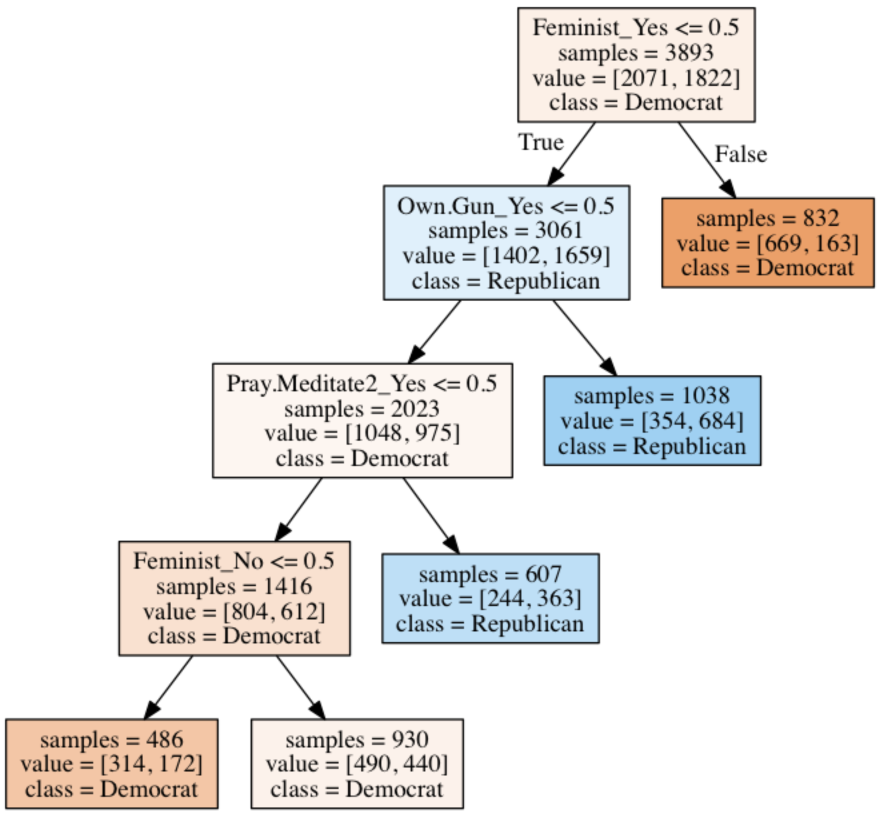
Let's take a look univariate comparison between our categorical vs. our target. We will take the odds ratio for every binary categorical and list the highest or lowest (the lowest will be inverted in the table i.e. 1/score) odds ratios. These features are considered strong indicators of what the target value will be.

The target breakdown is the following:

Question/Answer Options	Odds Ratio	Party
Are you a Feminist?	5.43	Democrat
Own Gun?	2.37	Democrat
Pray/Meditate?	2.19	Republican
Pray/Meditate2?	2.11	Republican
Does life have a purpose?	1.90	Republican
Did your parents spank you?	1.61	Republican
Which parent wore the pants (Mom/Dad)?	1.55	Democrat

We can also create a Decision tree to have an idea of which features are important.

Figure 5 - Graphical Representation of the Decision Tree



The tree reconfirms that features from the odds ratio analysis correspond to strong predictive features.

Based on the sparsity of the data set let's see if it's possible remove very sparse samples from the data set to ensure that the data to be trained is representative, but we need to ensure that not too many samples are removed so that the model has sufficient data to train on. Based on this, I applied the following strategy:

1. Remove samples from the data set based on a sparsity threshold, with threshold percentage defined as the percentage of features that are filled.
2. Impute using MICE [7] imputation scheme.
3. Cross validate using stratified shuffle split sampling, and area under ROC curve as scoring metric. This is too ensure that class imbalancing due to sampling will not lead to misleading score.

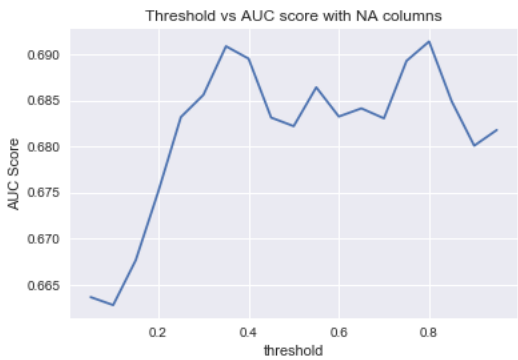
Plots of score vs. threshold value we generatdd for the following models using dummy variables that included whether the sample had an NA as feature and dummy variables that did not.

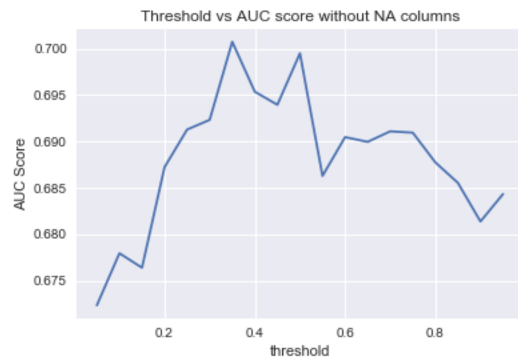
- Logistic Regression
  - L1, L2, elastic net regularization
- Suport Vector Machine using RBF kernel
- Random Forest (50 trees)

- Gradient Boosting (50 trees)

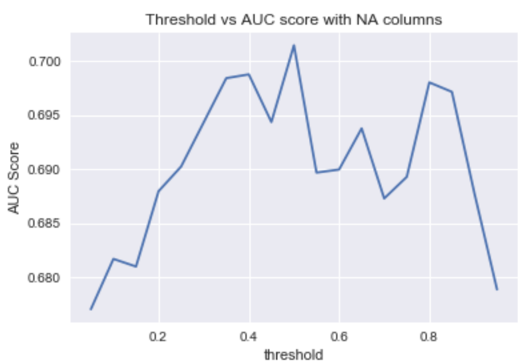
Figure 6 - Prediction Scores vs Sparsity Thresholds

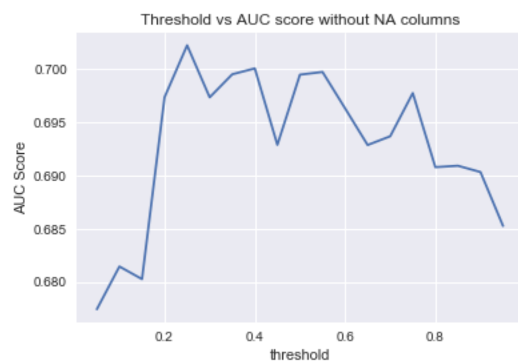
Support Vector Machine



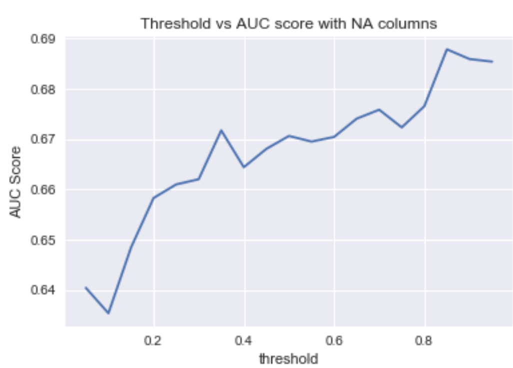


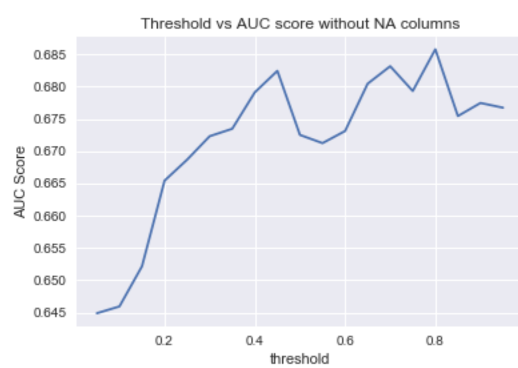
Gradient Boosting



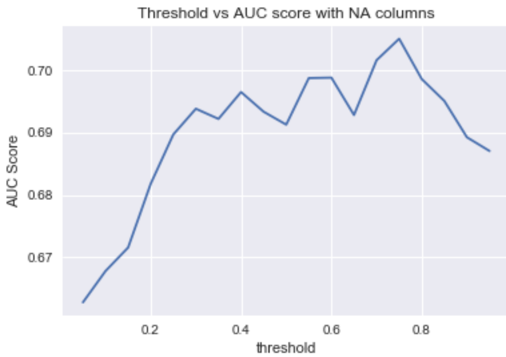


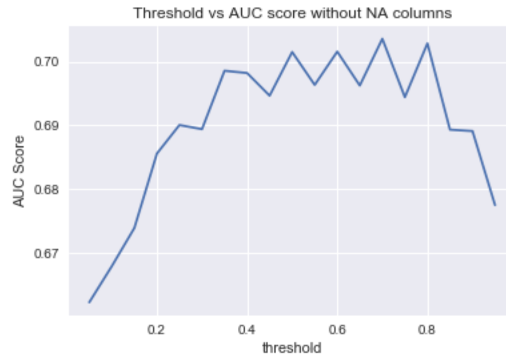
Random Forest



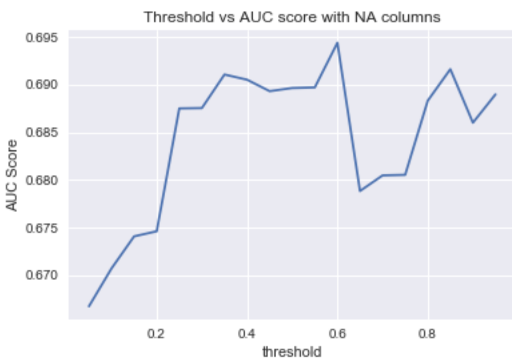


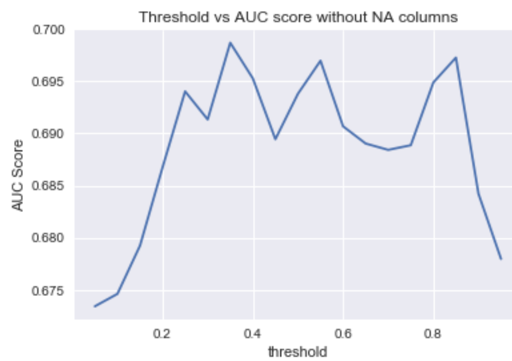
Logistic Regression L1 Regularization



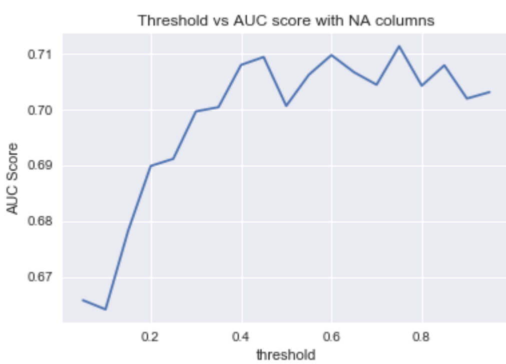


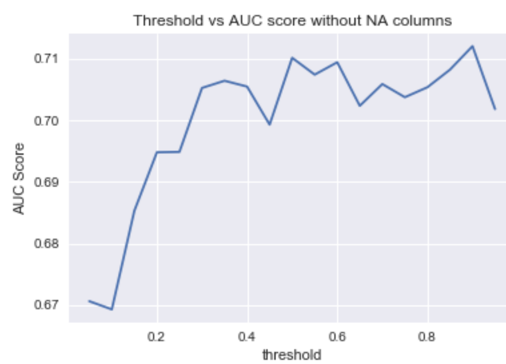
Logistic Regression L2 Regularization





Logistic Regression Elastic Net Regularization





From the analysis above I deduce the following:

- There is not a major difference in performance from:
- Model to Model
- Using dummy variables including NAs vs not inlcuding NAs.
- There performance is generally worse for thresholds of less than 0.3

Based on the following going forward we will the following processing for our training and tuning each model.

1. Utilize data set that includes dummy variables with NAs
2. Reduce samples to 0.3 threshold
3. Impute NAs via MICE algorithm
4. BoxCox transform continuous and ordinal features then rescale using MinMax to ensure that all values are between 0 and 1 (only for non tree models)

Algorithms and Techniques

Here are is brief description of the algorithms used in this project.

Linear models - Given a number of data points **N** and a number of features **p** where **N > p**, one can calculate a model that best fit the target values $f(X)$.

$$f(X) = \beta_0 + \sum_{j=1}^{p} \beta_j$$

Where the β_js represent the weights of the features. To obtain the weights where certain features can be reduced to zero (hence we're in effect performing feature selection) we apply the following algorithm.

$$\text{Minimize: } \sum_{i=1}^{N} (Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Where λ is a regularization term that is varied to apply generalization to the model. The higher value chosen the more regularized the model. Since there is no closed form solution for the lasso version of linear regression and algorithm called stochastic gradient descent is used. Here is the summary of the algorithm:[8]

- Choose an initial vector of parameters β and learning rate η .
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For $i = 1, 2, \ldots, n$ do:
    - $\beta := \beta - \eta \nabla Q_i(\beta).$

Where $\nabla Q_i(\beta)$ is the gradient of a single example.

Tree models

A decision tree is a graph that uses the branching method to show each possible outcome of a decision. A decison tree takes each attribute and finds which attribute splits the data to achieve the greatest "purity" i.e. for classification which attribute best splits the data where each decision split has the most samples of a particular class. Node purity can be determined by different metrics such as entropy or information gain. The formula for entropy in a binary classification is:

$$\sum_{i=1}^{c} -p_i log_2(p_i)$$

Which ranges from 0 (nodes pure or only 1 class) to 1 (classes split evenly)

A popular algorithm that is used is called the ID3 algorithm. Here is the algorithm in summary:[9]

1. Calculate the entropy of every attribute using the data set S
2. Split the set S into subsets using the attribute for which the resulting entropy (after splitting) is minimum (or, equivalently, information gain is maximum)
3. Make a decision tree node containing that attribute
4. Recurse on subsets using remaining attributes.

Ensemble models

Ensemble models are essentially numerous B number of models aggregrated to ultimately decide what the output should be for a sample based on some criteria (either through a voting or majority vote). There are numerous algorithms. Here is a quick summary for a random forest algorithm:[10]

For $b = 1, \ldots, B$: Sample, with replacement, B training examples from X, Y; call these X_b, Y_b. Train a decision or regression tree f_b on X_b, Y_b **on a partial subset of features chosen randomly*** After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x')$

Usually the choice of how many features to randomly choose dependent on p e.g. $\sqrt{p}$ or $\frac{p}{3}$

Boosting are somewhat similar except that they train on trees with few or even one feature (stumps) a number of times weighting more the trees that are incorrect. Here is a summary of the algorithm for boosting (James, Witten, Hastie Tibshirani 2013):

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat: (a) Fit a tree $\hat{f}_b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r). (b) Update ˆ f by adding in a shrunken version of the new tree:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_b(x)$$

   (c) Update the residuals,

   $$r_i \leftarrow r_i - \lambda f_b(x_i)$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}_b(x)$$

where λ is the shrinkage rate which is analagous to the learning rate in stochastic gradient descent.

Linear models such as logistic regression require the number of samples to be greater than the number of features **N > p**. Assuming the samples are not duplicates (full rank), we then can produce model based on inverting the data matrix to get model weights. Most times this can be much faster in computation than other techniques that require more computation. For our project the number of features **p = 394** and the number of samples, for a **threshold of 0.3, N = 3893**, we have the ability of using linear models such as logistic regression.

If the data is not linearly separable or if **p > N** we can use non-linear models such as support vector machines and/or tree models. Ensemble tree models tend to do better if the data has a lot of noise, since trees have the ability to be pruned to avoid overfitting to noise and that fact that multiple models are used and averaged together one single model that does fit to the noise have a neglible affect when averaged with other models. Gradient boosting tends outperform random forests since they have less bias (but may overfit), but random forests are simpler to tune in terms of hyperparameters.

Benchmark Model

The Kaggle competition creator provided code in the R language perform a simple logistic model to achieve a baseline score:

```
# KAGGLE COMPETITION - GETTING STARTED

# This script file is intended to help you get started on the Kaggle platform, and to show you how to make a submission to the competition.


# Let's start by reading the data into R
# Make sure you have downloaded these files from the Kaggle website, and have navigated to the directory where you saved the files on your computer

train = read.csv("train2016.csv")

test = read.csv("test2016.csv")

# We will just create a simple logistic regression model, to predict Party using all other variables in the dataset, except for the user ID:

SimpleMod = glm(Party ~ . -USER_ID, data=train, family=binomial)

# And then make predictions on the test set:

PredTest = predict(SimpleMod, newdata=test, type="response")

threshold = 0.5

PredTestLabels = as.factor(ifelse(PredTest < threshold, "Democrat", "Republican"))

# However, you can submit the file on Kaggle to see how well the model performs. You can make up to 5 submissions per day, so don't hesitate to just upload a solution to see how you did.

# Let's prepare a submission file for Kaggle (for more about this, see the "Evaluation" page on the competition site):

MySubmission = data.frame(USER_ID = test$USER_ID, Predictions= PredTestLabels)

write.csv(MySubmission, "SubmissionSimpleLog.csv", row.names=FALSE)

# You should upload the submission "SubmissionSimpleLog.csv" on the Kaggle website to use this as a submission to the competition

# This model was just designed to help you get started - to do well in the competition, you will need to build better models!
```

The accuracy on the test data was:

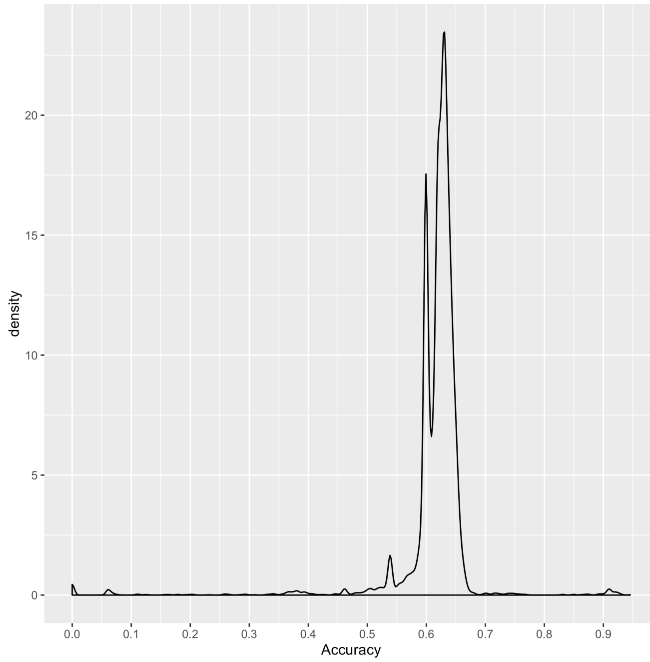
Public Score = **0.59914**

Private Score = **0.57902**

Contestants were expected achieve an accuracy better than this.

A density graph of the public scores was also created from the Kaggle scores generated from 8336 competitors:

Figure 7 - Scores of all the Kagglers that Participated in the Competition



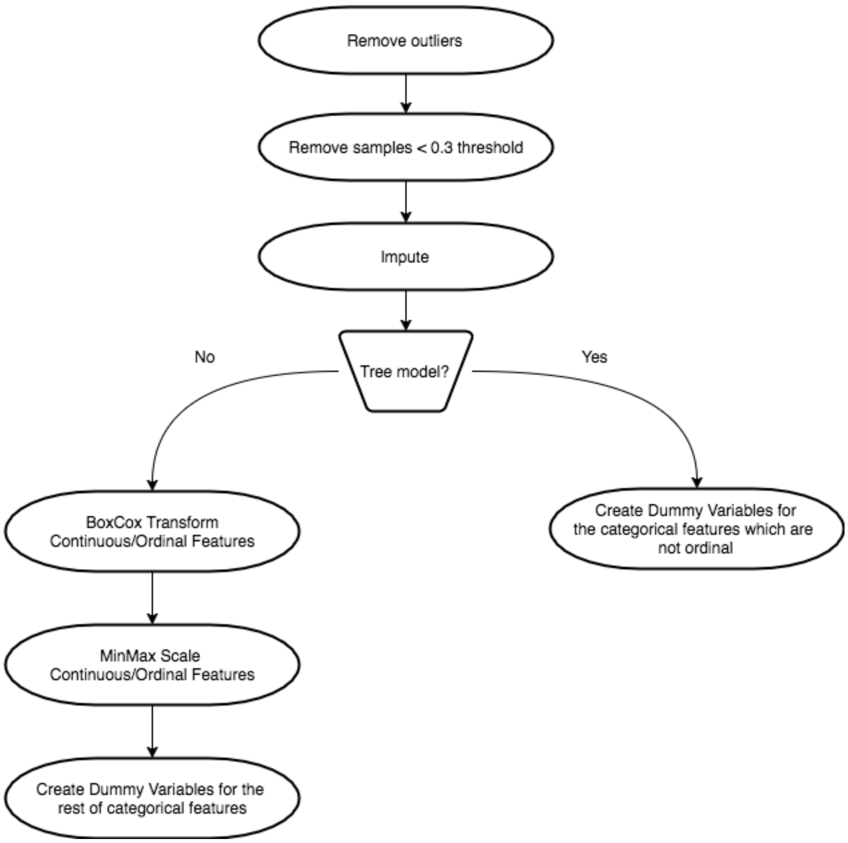
The first hump in the bimodal density graph represents the benchmark. While the second hump represents the students improvement on the data where the median is approximately **0.625**. The goal is to achieve an accuracy score in the neighborhood of the second hump.

III. Methodology

Data Preprocessing

As mentioned above the preprocessing steps are layed out in the diagram.

Figure 8 - Preprocessing Flowchart



Note again that the MICE imputation will be utilized.

Implementation

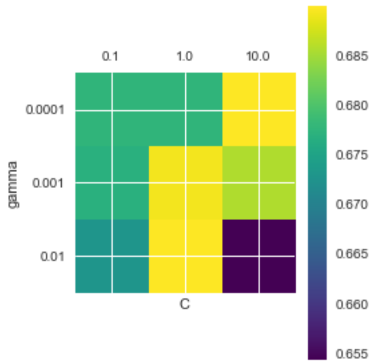
The following algorithms and parameters were used:

- Logistic Regression, Specifically `ElasticNetCV` function Perform a GridSearch on the following:
  - alpha values: ranging from $6 X 10^{-5} and 6 X 10^{-1}
  - L1 ratios: .1, .5, .7, .9, .95, .99, 1. Where value of 0 corresponds to Ridge regression and value of 1 referring to Lasso regression
- Random Forest Classifier Perform a GridSearch on the following:
  - number of trees: 1000, 1500, 2000.
- Support Vector Machine Perform a GridSearch on the following:
  - Gamma: $1 X 10^{-4}, 1 X 10^{-3}, 1 X 10^{-2}
  - C: 0.1, 1, 10
- Gradient Boosting Perform a GridSearch on the following separately in this order:
  - GridSearch 1
    - Minimum Samples per split: 1200, 1400, 1600, 1800, 2000
    - Minimum samples per leaf: 2, 4, 6, 8, 10
    - Apply best parameters to GridSearch 2
  - GridSearch 2
    - Subsample: 0.6, 0.7, 0.8, 0.9, 0.1
    - Max Depth: 4, 5, 6
      - Apply best parameters to GridSearch 3
  - GridSearch 3
    - n_estimators: 50, 60, 70, 80, 90, 100, 110, 120, 130, 140

An issue that I encountered tuning parameters was picking a parameter that increased the complexity of the model which increased the computational time drastically. For example, in the SVC model, including the C parameter of 100 increased the model time by an order of magnitude but didn't yield a substantial increase in cross validated score.

An example of picking the model with the highest prediction scores is shown below:

Figure 9 - An example of a parameter grid heat map



All of the code for processing the data and running the models are documented in the Capstone.ipynb notebook

Refinement

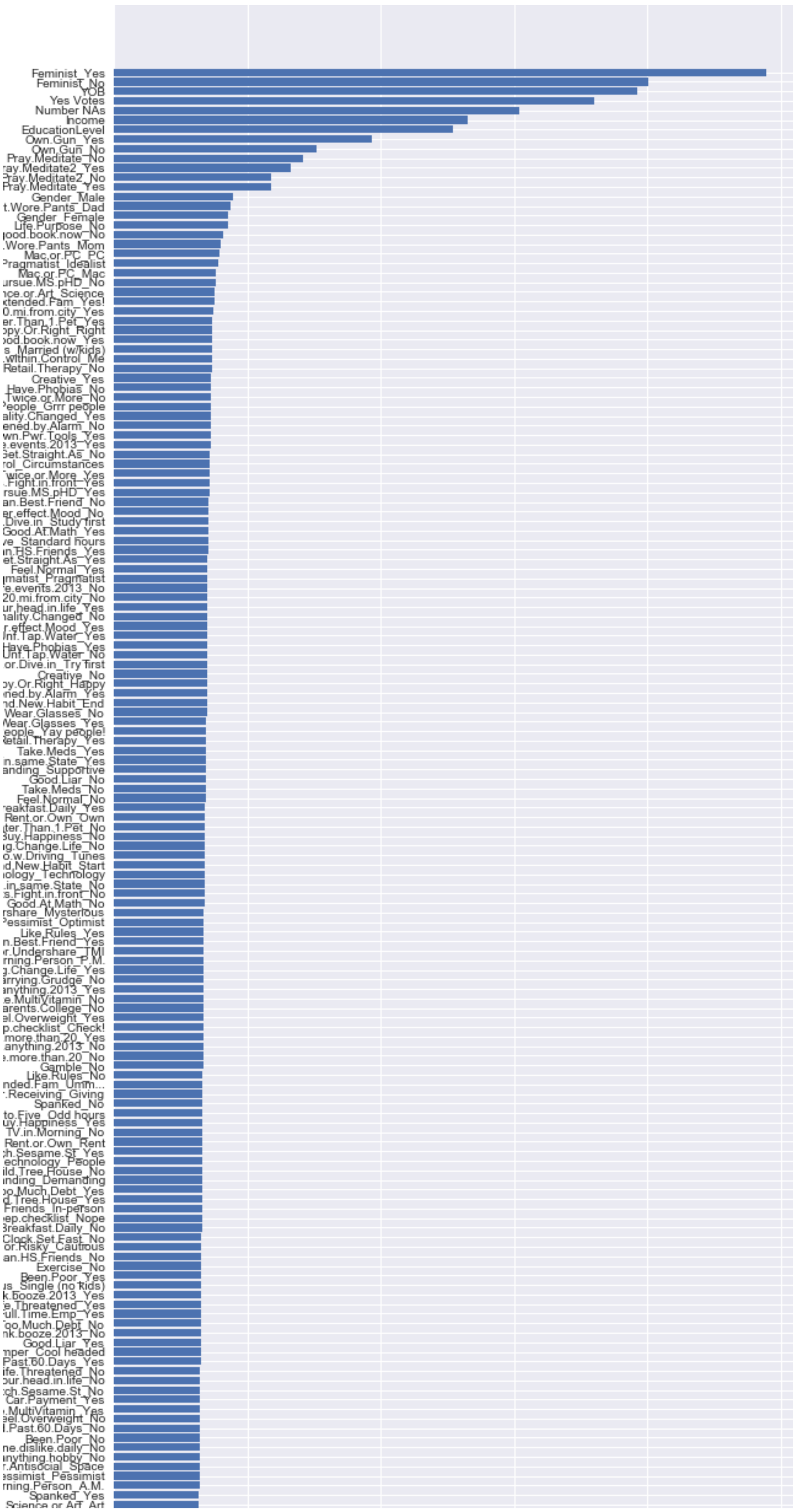
The practice of creating meta-features is a common practice in machine learning and data mining. For instance if you read several jupyter notebooks online on how to approach prediction with the Titanic Data set.[11] Several solutions contain numerous ways to create new features out of the existing data set. For example, one person took a feature called *Names* which is the manifest of people that have boarded the Titanic boat, extracted single women from married woman based on their salutation (Mrs. vs. Miss) and weighted them differently based on their marital status. An extra column with weights for all individuals was created based on this information, which added gave a better prediction score.

The initial models above were ran and achieved similar results (to be discussed in the evaluation section). In order to increase the prediction score, two meta-features were created:

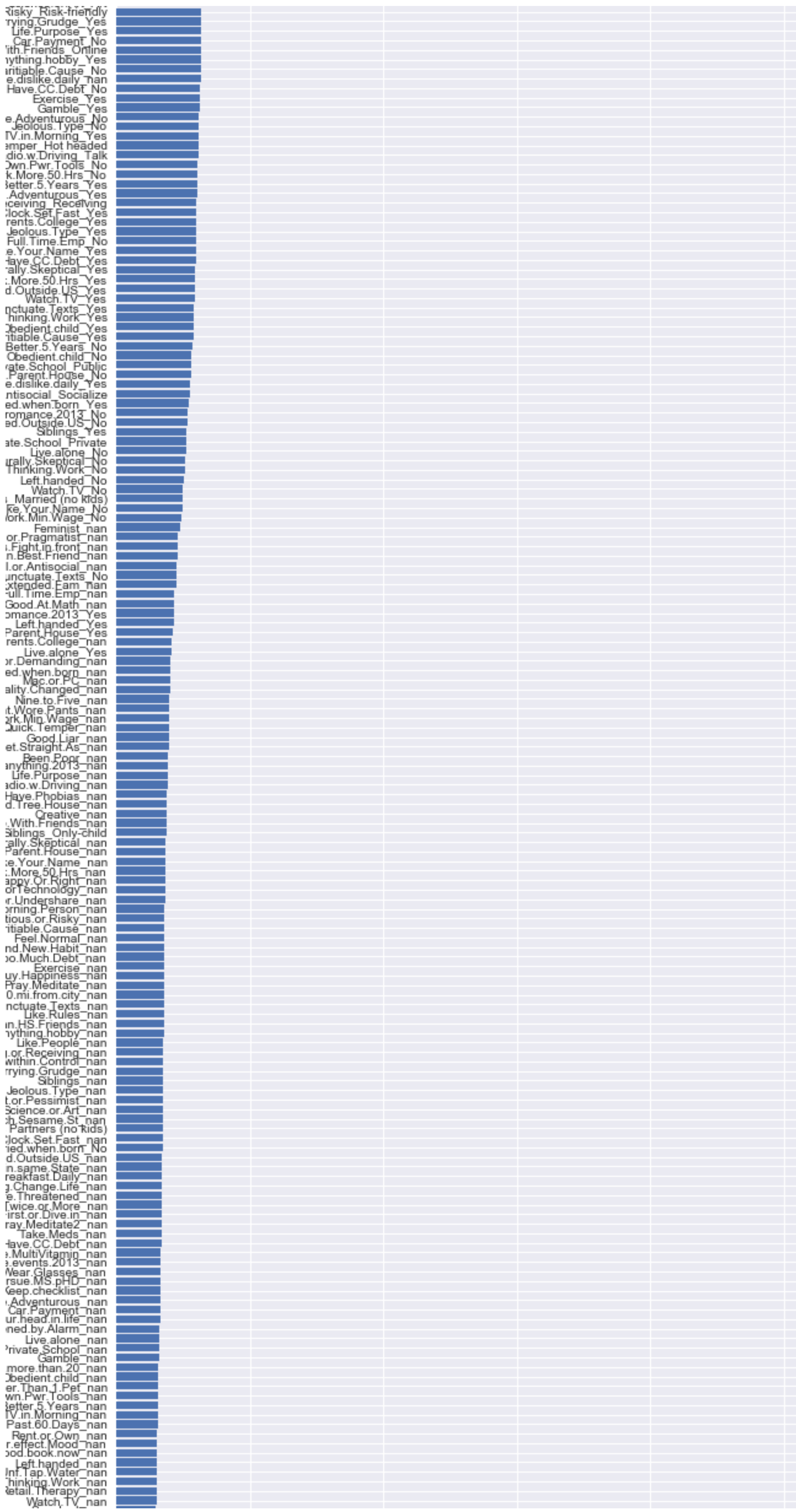
- Number of Questions answered 'Yes'
- Number of Questions left blank

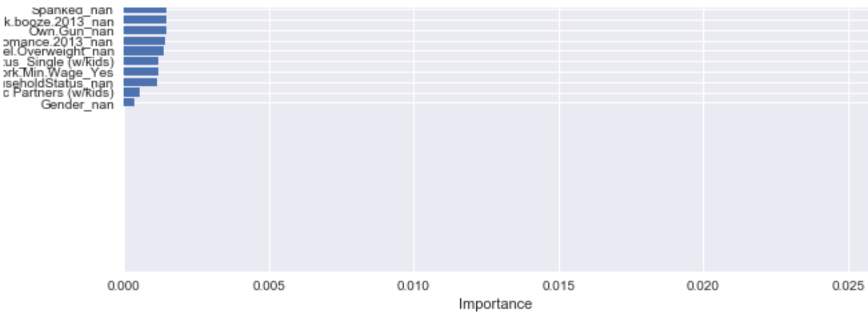
These two meta-features were created, because they were obvious and didnt really require upfront domain knowledge (i.e. quick and dirty). The the new data set were ran again on the Random Forest and Logistic Regression models using the same hyperparameters derived from their grid searches. Here is the importances of the new data set with meta-features. The importance plot shows that Yes Votes and Number NAs are considered among the most important features for prediction. Although they ranked very important the meta-features did not overall did not increase the prediction score and would probably need meta-features that were derived using domain knowledge.

Figure 10 - Ranked importances of the Random Forest Model



Feminist_Yes
Feminist_No
YOB
Yes Votes
Number NAs
Income
EducationLevel
Own.Gun_Yes
Own.Gun_No
Pray.Meditate_No
ray.Meditate2_Yes
Pray.Meditate2_No
Pray.Meditate_Yes
Gender_Male
t.Wore.Pants_Dad
Gender_Female
Life.Purpose_No
good.book.now_No
.Wore.Pants_Mom
Mac.or.PC_PC
Pragmatist_Idealist
Mac.or.PC_Mac
ursue.MS.pHD_No
nce.or.Art_Science
xtended.Fam_Yes!
0.mi.from.city_Yes
er.Than.1.Pet_Yes
ppy.Or.Right_Right
ood.book.now_Yes
s_Married (w/kids)
.within.Control_Me
Retail.Therapy_No
Creative_Yes
Have.Phobias_No
Twice.or.More_No
People_Grrr people
ality.Changed_Yes
ened.by.Alarm_No
wn.Pwr.Tools_Yes
e.events.2013_Yes
Get.Straight.As_No
rol_Circumstances
wice.or.More_Yes
.Fight.in.front_Yes
rsue.MS.pHD_Yes
an.Best.Friend_No
er.effect.Mood_No
.Dive.in_Study first
Good.At.Math_Yes
ve_Standard hours
in.HS.Friends_Yes
et.Straight.As_Yes
Feel.Normal_Yes
gmatist_Pragmatist
e.events.2013_No
20.mi.from.city_No
ur.head.in.life_Yes
ality.Changed_No
r.effect.Mood_Yes
Unf.Tap.Water_Yes
Have.Phobias_Yes
Unf.Tap.Water_No
or.Dive.in_Try first
Creative_No
py.Or.Right_Happy
ened.by.Alarm_Yes
nd.New.Habit_End
Wear.Glasses_No
Wear.Glasses_Yes
eople_Yay people!
Retail.Therapy_Yes
Take.Meds_Yes
n.same.State_Yes
anding_Supportive
Good.Liar_No
Take.Meds_No
Feel.Normal_No
reakfast.Daily_Yes
Rent.or.Own_Own
ter.Than.1.Pet_No
Buy.Happiness_No
g.Change.Life_No
o.w.Driving_Tunes
d.New.Habit_Start
ology_Technology
.in.same.State_No
s.Fight.in.front_No
Good.At.Math_No
rshare_Mysterious
Pessimist_Optimist
Like.Rules_Yes
n.Best.Friend_Yes
r.Undershare_TMI
rning.Person_P.M.
g.Change.Life_Yes
arrying.Grudge_No
anything.2013_Yes
e.MultiVitamin_No
arents.College_No
el.Overweight_Yes
p.checklist_Check!
more.than.20_Yes
anything.2013_No
.more.than.20_No
Gamble_No
Like.Rules_No
nded.Fam_Umm...
Receiving_Giving
Spanked_No
to.Five_Odd hours
uy.Happiness_Yes
TV.in.Morning_No
Rent.or.Own_Rent
h.Sesame.St_Yes
echnology_People
ild.Tree.House_No
nding_Demanding
oo.Much.Debt_Yes
d.Tree.House_Yes
Friends_In-person
ep.checklist_Nope
Breakfast.Daily_No
Clock.Set.Fast_No
or.Risky_Cautious
an.HS.Friends_No
Exercise_No
Been.Poor_Yes
s_Single (no kids)
k.booze.2013_Yes
e.Threatened_Yes
Full.Time.Emp_Yes
Too.Much.Debt_No
nk.booze.2013_No
Good.Liar_Yes
mper_Cool headed
Past.60.Days_Yes
ife.Threatened_No
our.head.in.life_No
ch.Sesame.St_No
Car.Payment_Yes
.MultiVitamin_Yes
eel.Overweight_No
l.Past.60.Days_No
Been.Poor_No
ne.dislike.daily_No
anything.hobby_No
r.Antisocial_Space
essimist_Pessimist
rning.Person_A.M.
Spanked_Yes
Science.or.Art_Art





IV. Results

Model Evaluation and Validation

Figure 11 - Results based on model and parameters

Model	Parameters	Public Score
Logistic Regression (elastic net)	alpha = 0.0067; L1 ratio = 1	0.62500
Logistic Regression with meta-features(elastic net)	alpha = 0.0067; L1 ratio = 1	0.61638
Random Forest	2000 trees, random features selected = sqrt features	0.62644
Random Forest with meta features	2000 trees, random features selected = sqrt features	0.61925
Gradient Boosting Machine	max_depth = 3; min_samples_leaf = 6; min_samples_split = 1300; n_estimators = 80; subsample = 1.0	0.62213
Gradient Boosting Machine with meta features	max_depth = 3; min_samples_leaf = 6; min_samples_split = 1300; n_estimators = 80; subsample = 1.0	0.62213
Support Vector Machine	C = 10; gamma = 0.001	0.61063
Support Vector Machine with meta features	C = 10; gamma = 0.001	0.62069

As a reminder the baseline scores were the following:

Public Score = **0.59914**

Private Score = **0.57902**

On average accuracy increased from the baseline score by approximately 2%. Although this seems minor it was on par with what other Kagglers seemed to achieve on average. It seems that model performance is agnostic of model type and whether we add these two additional features. Since model parameters were cross validated via stratified k-fold (w shuffling), I have high confidence that model generalizes to data unseen from the model. The scores in the table justify this notion as all the scores are relatively consistent. Based on this I would pick the logistic model using L1 regularization (corresponding L1 ratio =1) with additional meta features as the winning model from a speed performance standpoint. The threshold test shown earlier that the model is robust to perturbations in the original set as long as not too much data is removed,hence the threshold of 0.3 value.

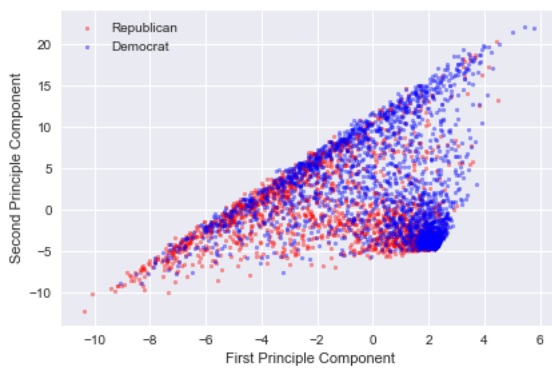
V. Conclusion

Reflection

We've shown that with straightforward preprocessing (removing samples based on thresholding and imputation) and gridsearch of parameters via cross-validation that the prediction score can be increased by approximately two extra percent. Given the Kaggle scores of previous submissions this score is on par of what other participants achieved on average. Given the sparsity of the data, there were numerous ways that the data could have been processed. Multiple imputation schemes as well as deleting entire features altogether. In terms of feature selection multiple types of subselection schemes could have been applied such as recursive feature elimination and univariate correlation with the target.

On sort of dimension reduction technique is an off shoot of PCA called Partial Least Squares. In short it's somewhat of a supervised version of PCA. Here is the plot below for two dimensions:

Figure 12 - PLS Regression of first two Principle components



As you can see there are separation in certain areas but overlap in others. When a classifier was performed on it produced similar results to the previous models. When performed on higher dimensions the results were still in the same ballpark.

Figure 13 - Scores for each Number of Principle Components used

Number of Dimensions	Score
10	0.641398078287
20	0.638302702006
30	0.640043178256
40	0.638690969857
50	0.638688278225
60	0.636563795873
70	0.634825114044
80	0.630378202271
90	0.627479427243
100	0.627477632822

The elastic net L1_ratio set to 1 regularization (complete LASSO) is a method to select the best features. It's result still seemed to be on par with the rest of the models that included all of the features.

Improvement

Although the previous processing and gridsearch validation did increase the score from the baseline. It did not improve it drastically. I feel that in order to increase prediction score even further more features need to be created with use of **domain knowledge**. For example weighting certain features and combining them either additavely or multicatively based on domain knowledge to create features that the model hasn't previously seen would aid in separating Republican samples from Democrat samples further. Most likey the top Kagglers had used this strategy.

References

[1]*Party Predictor: Predicting Political Affiliation http://cs229.stanford.edu/proj2013/EwonusMcCannRoth-PartyPredictorPredictingPoliticalAffiliation.pdf*

[2]*Predicting the Political Alignment of Twitter Users https://pdfs.semanticscholar.org/ccaf/a80db5f4b19886d6bbe9a2a37e2048d52a28.pdf*

[3]*Decision Trees and Political Party Classification https://jeremykun.com/2012/10/08/decision-trees-and-political-party-classification*

[4]*http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records*

[5]*https://inclass.kaggle.com/c/can-we-predict-voting-outcomes*

[6]*https://www.showofhands.com*

[7]*https://stat.ethz.ch/education/semesters/ss2012/ams/paper/mice.pdf*

[8]*https://en.wikipedia.org/wiki/Stochastic_gradient_descent*

[9]*https://en.wikipedia.org/wiki/ID3_algorithm*

[10]*https://en.wikipedia.org/wiki/Random_forest*

[11]*https://www.kaggle.com/c/titanic*