Machine Learning with Census Data

MSAN 621

Luba Gloukhov, Monica Meyer, and Griffin Okamoto

December 5, 2014

Executive Summary

The data for this problem contains weighted demographic and employment information from the 1994 and 1995 current population surveys conducted by the U.S. Census Bureau. The target variable is whether a certain individual has an income of over $50,000 per year. There are 40 predictors, including age, race, sex, employment industry, occupation, education level, wage earning per hour, and more (see Appendix 2 for the full list). Our approach to modeling this data set began with data exploration and cleaning, which involved missing value imputation, creating dummy variables for categorical features, and univariate analysis of the features. Once this was handled, we moved forward onto the machine learning models. The data is extremely unbalanced, so our focus became accurately predicting the rare positive class, an individual earning an income of over $50,000 per year. We implemented six different machine learning algorithms: Choose Feature, Naive Bayes, K-Nearest Neighbors, Logistic Regression, Random Forest and Support Vector Machine. Implementations of these algorithms included data preparation, hyperparameter tuning, feature selection and cross validation of the training data. After extensive tuning, we evaluated our models' performance on the test data and compared various performance metrics. In particular, we care about the f1 score which combines the precision and recall for the positive class, helping us achieve an optimal balance. This paper is a discussion of our process from cleaning the data, tuning the models, performing cross validation and finally evaluating the models' performance on the test data.

Data Exploration and Cleaning

Missing Values and Imputation

Our data exploration revealed that the data set has many missing values. In fact, there are no examples that have a value for every single feature. Missing values in the original data set are represented as "Not in Universe" or "?", depending on the feature. Since all of the examples have missing values, we could not simply eliminate examples. We chose to impute these missing values in R using na.roughfix() in the randomForest package, which replaces missing values with the median for numeric variables and the most common category for categorical variables. To understand the implications of this imputation, for each feature we examined the univariate plot, how many missing values there are, and the imputed value. These plots can be found in Appendix 2. Many of the variables have no missing values, but there are 7 variables with over 90% missing values. In addition, none of the numeric variables have missing values. This brought up concerns of whether missing values in these columns were indicated with other codings we were not aware of. Imputing in this simple way may reduce the information contained in

each of our features by diluting or misconstruing the true values.

In addition to imputing missing values using na.roughfix(), we experimented by imputing missing values using a Random Forest classifier. For those variables with ≥ 90% observations missing, we fit the model on the the ≤ 90% non-missing observations and predicted values for the remaining ≥ 90% observations. As such, instead of missing values being replaced with the mean or median value, missing values were replaced with the value predicted by a Random Forest model. In each instance, the Random Forest model contained 10 maximal-depth trees and measured the quality of the split with the Gini index. The 5-fold cross validation scores (f1, accuracy) for the 7 models are listed in Appendix 3. We tested the performance of KNN and Logistic Regression under the strictly mean/median imputation and the aforementioned Random Forest partial imputation on 10% of the training data. In both cases, the performance was not improved significantly enough to warrant fitting and predicting on the full dataset seven additional times. The charts in Appendix 3 compare the performance on an equally sampled 10% sample of the data.

Data Normalization

Aside from the missing value imputation, we needed to convert the categorical variables to numeric dummy variables so that the data could be used with sklearn. For this task, we utilized R to convert the categorical variables to numbers representing their category, then sklearn's OneHotEncoder() function to write them to dummy variables in Python. This increased the number of predictors from the original 40 to 489, drastically increasing the dimensionality of the dataset. This is very restrictive on some algorithms. We decided that most feature selection or data normalization should be done on a case-by-case basis for each algorithm. Some algorithms benefit more from these preprocessing steps than others, whether it be for increases in accuracy (and other performance metrics) or for decreases in computation time.

Special Considerations

Another primary concern with this data set is the class imbalance. There are far many more negative examples (less than 50k) than positive examples (greater than 50k). In most cases, this will make it difficult for us to accurately identify positive examples, which are what we are primarily interested in. We can use stratified sampling to obtain balanced training samples, and some algorithms have hyperparameters that allow us to weight the classes differently. However, in all cases we chose to use stratified samples of various sizes from the training data to fit our algorithms. From there, we chose to optimize our hyperparameters on F1 score for the positive class. We still considered accuracy, as well as precision and recall for the positive class.

The Models

Choose Feature

Data Preparation

The choose feature algorithm predicts on the basis of the feature that is found to have the best weighted purity measured by the Gini index. It splits all features according to their mean value, then predicts a new example based on the mean value of the chosen feature and the majority class for that split. Due to the simplicity of this algorithm, preparation of the data is limited to cleaning up the missing values and then choosing a balanced data set to train the model. In this case, it is important to choose a balanced data set to train the model, because the data is heavily skewed towards the negative target class (or making less than $50,000). Without the balanced dataset, choose feature classified every prediction

as negative. Thus, we stratify our sample so that each target class is represented with equal weight.

Cross-Validation Scores

From 5-fold cross validation on our balanced sample of the full training dataset size, algorithm performance can be summarized by the following metrics:

F1	Accuracy	Precision	Recall
0.5440	0.7705	0.5402	.5551

NAIVE BAYES

Data Preparation

The Naive Bayes algorithm applies Bayes' theorem with the assumption of conditional independence between every pair of features. Naive Bayes classifiers have worked well due to the small amount of training data necessary to estimate parameters and due to how quickly they run compared to more sophisticated methods. Preparation of the data is limited to removing missing values and stratifying the training data so that we train on a balanced data set. The next step involved in running this algorithm is feature selection.

Feature Selection

After removing missing values in the data, and encoding non-numeric values to binary features, we were left with a selection of 489 different features. We ran a test using SelectKBest in sklearn to pick the k best features. This looped through k = (2:489) features choosing the k best features, then performing 5-fold cross validation on a balanced sample of the training data. This allowed us to find the number of features that produced the best results. This was tested on 10%, 20% and 100% size stratified samples of the data. The table below presents the top five performing number of features and their cross validated f1 scores by percent sample of training data chosen. From this information, we chose to perform training on a balanced sample of the same size as the training dataset and let SelectKBest choose the best 136 features for training the Naive Bayes model.

Features	10%	Features	20%	Features	100%
134	0.842	135	0.842	136	0.842
140	0.841	136	0.842	31	0.841
130	0.840	131	0.841	137	0.840
122	0.840	127	0.840	135	0.840
106	0.838	21	0.839	134	0.839

Cross-Validation Scores

Based upon 5-fold cross validation on our balanced sample of the full training dataset size, algorithm performance can be summarized by the following metrics:

F1	Accuracy	Precision	Recall
0.8413	0.8223	0.7601	0.9419

K-NEAREST NEIGHBORS

Data Preparation

k-Nearest Neighbors (KNN) is a computationally intensive algorithm due to the need to calculate the distance to all points upon predicting for a new point. In order to ease computational demands and work within the time constraints, 10% of the data was randomly sampled for model fitting in the process of hyperparameter tuning. The 10% sample was selected by sampling without replacement from the

full dataset ("naturally sampled") and by stratifying so that each target group is represented with equal weight ("equally sampled").

Hyperparameter Tuning

The primary hyperparameters for KNN are the number of neighbors (k) and whether or not the predictive decision is weighted by the distance of each neightbor to the point of interest (*weights*). We searched the space spanned by "natural" versus "equal" sampling, w set to either "uniform" or "distance", and k in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 30, 50, 100, 500, 1000, n] (n equal to the number of observations used in model fitting – 39904 for the full dataset under 5-fold cross validation).

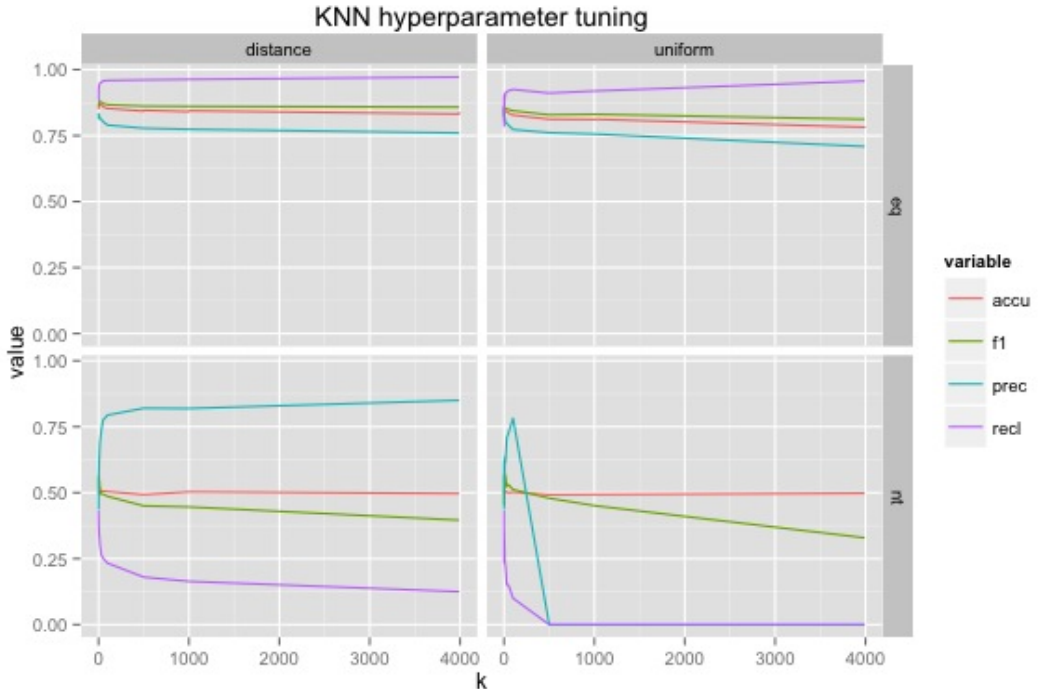


Figure 1

Figure 1 compares the performance across the space spanned by the aforementioned hyperparameter values. Note the separate plot for each combination of w (in the columns) and sampling (in the rows). The performance metrics for each one of accuracy, f1, precision and recall span the y-axis while the values of k span the x-axis. A 5-fold cross validation indicated optimal performance across at "equal" sampling, "distance" weighting and $k = 12$.

Cross-Validation Scores

Based upon 5-fold cross validation on our balanced sample of the full training dataset size, algorithm performance can be summarized by the following metrics:

F1	Accuracy	Precision	Recall
0.9389	0.9352	0.8885	0.9953

LOGISTIC REGRESSION

Data Preparation

Similarly as for k-Nearest Neighbors (KNN), for Logistic Regression, in order to ease computational demands and work within the time constraints, 10% of the data was randomly sampled for model fitting in the process of hyperparameter tuning. The 10% sample was selected by sampling without replacement from the full dataset ("naturally sampled") and by stratifying so that each target group is represented with equal weight ("equally sampled").

Hyperparameter Tuning

The primary hyperparameters for Logistic Regression are the type of regularization (p) and the inverse of regularization strength (c). Again, as with KNN, we searched the space spanned by "natural" versus "equal" sampling, p set to either "l1" or "l2", and c in [0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 1, 10, 20, 50, 100, 500, 1000, n] (n equal to the number of observations used in model fitting – 39904 for the full dataset under 5-fold cross validation).

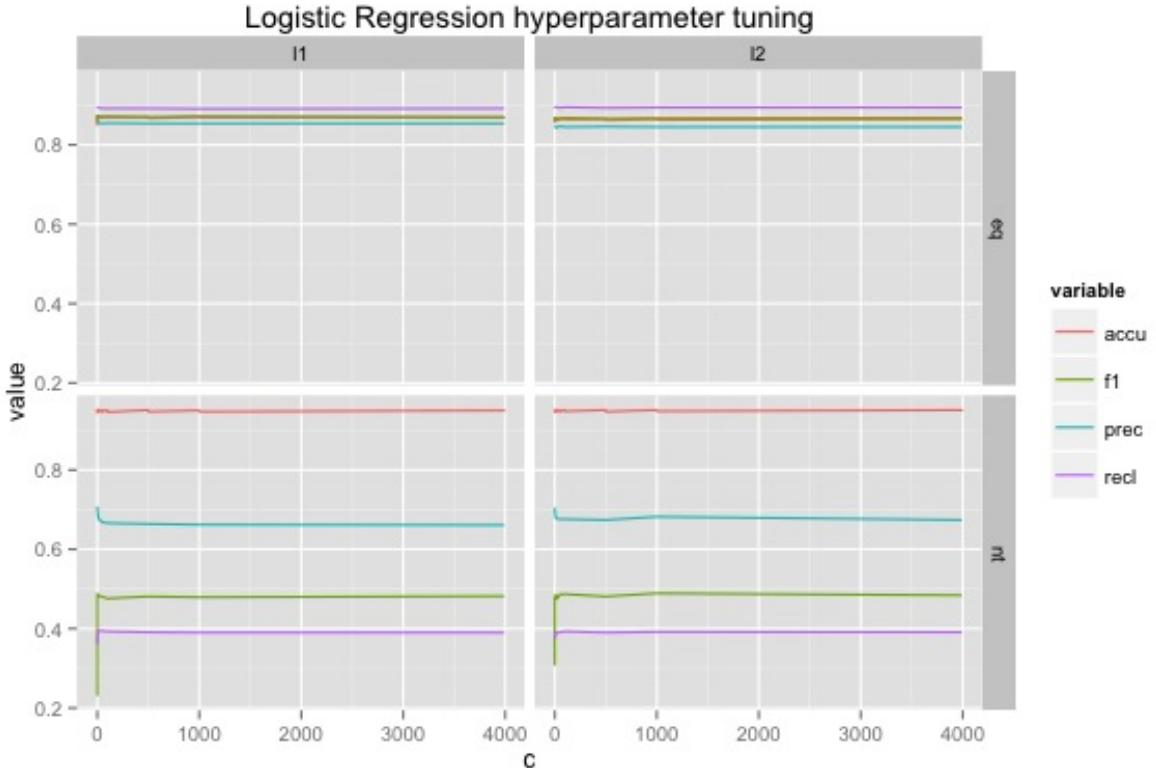


Figure 2

The chart in Figure 2 compares the performance across the space spanned by the aforementioned hyperparameter values. Note the separate plot for each combination of p (in the columns) and sampling (in the rows). The performance metrics for each one of accuracy, f1, precision and recall span the y-axis while the values of c span the x-axis. A 5-fold cross validation indicated optimal performance across at "equal" sampling, "l1" regularization with c=.25. Though the benefit of "l1" regularization over "l2" is small, "l1" can yield to a more parsimonious model by setting some of the coefficients to exactly zero.

Cross-Validation Scores

Based upon 5-fold cross validation on our balanced sample of the full training dataset size, algorithm performance can be summarized by the following metrics:

F1	Accuracy	Precision	Recall
0.433987	0.855566	0.286648	0.892984

RANDOM FORESTS

Data Preparation

Random forests is typically one of the most accurate machine learning algorithms of those we have learned, so we had high hopes for its performance on our data. Since the decision trees that are trained depend on node purity, having so few positive (over 50k) examples made it unlikely that any given leaf

would predict positive. Thus, we decided it would be best to use a balanced data set to train the model. In addition, we reduced the number of features using feature importances. This decreased the number of features from 489 to 72. Feature selection for random forests is more important than for single decision trees, because random forests only consider a random subset of features at any given split. However, there was no need to scale the data because splits do not concern the exact numeric value of the feature for a given example, just the class of similarly valued examples.

Hyperparameter Tuning

The primary hyperparameters for random forests are the number of trees and number of features to consider at a given split. The defaults for these hyperparameters in sklearn are 10 trees and $\sqrt{p}$ features. As such, we performed a grid search that optimized these parameters based on f1, precision, recall, and accuracy. The parameter space we searched over contained every combination of [10, 20, 50] trees and [$\sqrt{p}$, $log_2 p$, 0.2p, 0.4p, 0.6p] features. The cross-validation performances based on each combination were very similar for all metrics, but the optimal choice was 20 trees and $log_2 p$ features for all metrics except recall. Thus, these were the primary hyperparameters we selected.

Hyperparameter	Value
Number of Trees	20
Number of Features	$\sqrt{72} = 8$
Purity Criterion	Gini Index
Min. Examples for Split	2
Min. Examples for Leaf	1
Max. Number of Leaves	None

The other hyperparameters are those that follow from decision trees: purity criterion, maximum tree depth, minimum examples to perform a split, minimum examples in a leaf, and maximum number of leaves. In sklearn, the latter three hyperparameters default to be less restrictive, allowing trees to overgrow as much as possible. Since ensemble methods benefit from more variability among the individual estimators, this is typically the optimal choice of these hyperparameters. However, we performed a grid search to be sure, and confirmed this assumption. There was a negligable difference between Gini and Entropy, so we used the default, Gini. Thus, the final selection of hyperparameters is shown in in the table above.

Cross-Validation Scores

Based upon 5-fold cross validation on our balanced sample, our algorithm performance can be summarized by the following metrics:

F1	Accuracy	Precision	Recall
0.9812	0.9809	0.9643	0.9987

We note that F1, precision, and recall are for the positive (over 50k) class. Our precision for the positive class is the lowest, and our recall is the highest. In this case, we have optimized our recall at the expense of our precision, but this leads to the optimial F1 value among the random forest models we considered.

SUPPORT VECTOR MACHINE

Data Preparation

Since SVM takes an extremely long time to train, we wanted to reduce the number of examples to save time computationally. However, this leaves a strong risk of overfitting to fewer data points. So we verified that the test performance does not suffer significantly from training on only a subset of the data using learning curves. Figure 3 shows the learning curve for the algorithm with sklearn's default hyperparameters. This learning curve only shows very small subsets of the training data, and the training and validation scores still seem to converge quickly. Thus, the risk of overfitting is reasonably low, and we can proceed with optimizing hyperparameters and performing cross-validation on 25% of the original training set.

Like random forests, SVM typically performs very well "out-of-the-box." However, SVM usually benefits greatly from scaling the features. To test this, we performed 5-fold cross-validation on a random sample of 25% of the data, with and without scaling using SVM with the default penalty of C = 1 and RBF kernel. The average accuracy across folds did not improve noticeably (0.9454 to 0.9483), but the other metrics did. F1 increased from 0.2802 to 0.3346, precision increased from 0.6908 to 0.7536, and recall increased from 0.1758 to 0.2156, all with relation to the positive class (over 50k).

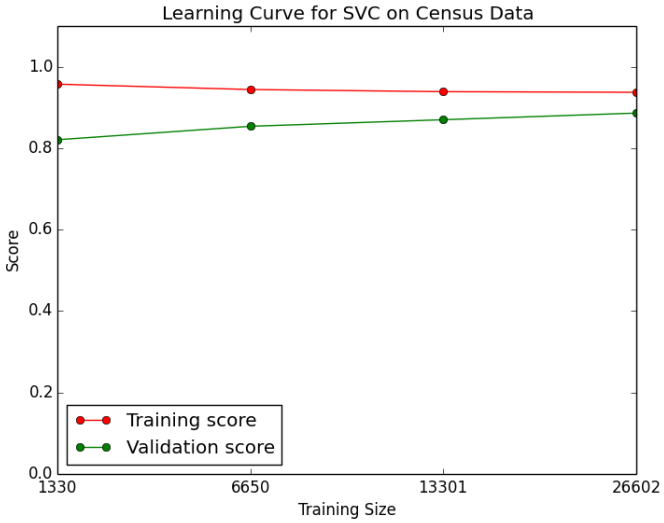


Figure 3

We were unsure whether SVM would perform significantly better with balanced data, compared to just adjusting the class weights. To find out, we performed the same cross-validation using the scaled data and either stratification or class weights. We selected class weights of 30 for the positive class and 1 for the negative class (since positives are over 30x rarer than negatives) for the sake of this rough test. For stratification, accuracy decreased, but all of the other metrics increased drastically. In contrast, for the weighted class approach, accuracy and precision decreased, and F1 and recall increased slightly. As such, we decided to stick with balanced data, as we have been using with the rest of our algorithms.

Hyperparameter Tuning

Since the RBF kernel has been chosen for us, the only hyperparameter to be tuned is the penalty parameter. The default for sklearn is 1, so we performed a grid search over the values [1, 2, 3, 5, 10]. Precision and accuracy were optimized for a penalty of 10, recall was optimized for a penalty of 2, and F1 was optimized for a penalty of 3. However, there was very little difference in the performance across all of the penalty values. Therefore, we chose a penalty of 3 for our final model.

Cross-Validation Scores

Based upon 5-fold cross validation on our balanced sample, our algorithm performance can be summarized by the following metrics:

F1	Accuracy	Precision	Recall
0.9203	0.9179	0.8938	0.9485

Like random forests, precision takes the hardest hit when optimizing for F1 score. The accuracy on the balanced set was surprisingly low, but it is hard to see how this will translate to the (likely) unbalanced test set.

EVALUATION

Finally, we applied the chosen algorithms to the test data. We imputed the test data in the same way as the training data. Each algorithm performed its own stratified sampling (of varying proportions of the test data), as well as feature selection and scaling/normalization, if any. The performance metrics for each of the algorithms on the test data are below:

Algorithm	F1	Accuracy	Precision	Recall
Choose Feature	0.235	0.627	0.135	0.923
Naive Bayes	0.248	0.641	0.142	0.955
K Nearest Neighbors	0.365	0.813	0.231	0.865
Logistic Regression	0.433	0.856	0.287	0.893
Random Forest	0.547	0.934	0.477	0.642
Support Vector Machine	0.423	0.850	0.278	0.891

As expected, all of the algorithms performed worse on the test data than our balanced training data. As a baseline, zeroR (not included in this report) would have achieved 0.9379 accuracy, 0 precision, 0 recall, and 0 F1 with respect to the positive class. Somewhat to our dismay, all of these algorithms performed worse in terms of accuracy than zeroR. However, all of them performed much better on the rest of the metrics. Random forest achieved the highest F1, precision, and accuracy scores, but the lowest recall. In contrast, choose feature and Naive Bayes scored the lowest on F1, precision, and accuracy, but highest on recall. SVM, K-Nearest Neighbors, and logistic regression performed moderately well in comparison.

These differences seem to be the result of our use of balanced samples. In many cases, we resampled from the positive examples to create a sample larger than the original number of positive examples. In doing so, we most likely produced duplicates that our algorithms learned very well, so some of the algorithms were able to learn these particular examples very well. This includes KNN, random forest, and logistic regression. However, algorithms that relied on the proportions and probabilities of each class in the training set were thrown off when presented a heavily unbalanced sample. This is not to say that taking stratified samples was wrong for these algorithms, but perhaps taking differently sized stratified samples would have been better than perfectly balanced ones.

CONCLUSION

The biggest hurdles in cleaning and transforming this data were imputing for missing values and changing the raw data into purely numeric data via dummy variables. While we dabbled in more sophisticated random forest-based imputation, it turned out that simple imputation based on medians and most common classes performed similarly. Creating dummy variables resulted in a much higher-dimensional data set than we originally expected, and necessitated feature selection and/or sampling for the computationally expensive algorithms.

The focus in our modeling of data set was handling the unbalanced classes, and optimizing our F1 parameter for the positive (over 50k) class. While we allowed great flexibility for each of the algorithms in terms of scaling, feature selection, and stratified sampling, tuning these on a case-by-case basis did not lead to ideal results. Overall, our algorithms performed sub-optimally on the test data compared to the training data. The balanced training samples affected the performance on the test data more for some algorithms than others. Random forest was most flexible in this regard, and choose feature and Naive Bayes were least so. Despite the promise of highly-regarded machine learning techniques, this exercise showed us that no algorithm is perfect "out-of-the-box" and that careful consideration of the specific quirks of the data is key to successful predictions.

APPENDIX 1: TEAMWORK

APPROACH

We each worked on two algorithms to split the bulk of the workload equally, and all contributed to the writeup of our data preprocessing and code aggregation. The more specific contributions we had are outlined below. We met 2 - 3 times a week, most often by video chat with Luba in Mountain View and Monica and Griffin in San Francisco. Our few in-person meetings were quick check-ups, most often directly before or after class. Our final meeting was a longer on-campus work session to coordinate code and our ideas for the writeup. We shared code by email because none of us have private repositories on Github. Since we mostly coded separate algorithms, this did not hinder our cooperation.

Luba

I worked on the data preparation, hyperparameter tuning, cross validation and corresponding coding and visualization of K-Nearest Neighbors and Logistic Regression. I initiated and completed the Random Forest imputation of sparse variables. I wrote the sections correpsonding to my work in this report and produced all corresponding plots.

Griffin

I worked on the random forest and SVM algorithms, including all scaling, feature selection, hyperparameter tuning, and cross validation. In addition, I worked a lot on the data exploration and cleaning, including producing the univariate plots and analysis. I wrote most of the R and Python code to clean up the training data and convert all of the features to numeric using OneHotEncoder. In the writeup, I wrote about my own algorithms and exploration of the data, and the majority of the evaluation and conclusion sections.

Monica

I worked on data preparation and cross validation of both the Naive Bayes and choose feature classifiers. I also wrote code to perform feature selection for the Naive Bayes model. I wrote about my algorithms and wrote the executive summary for this write up. In addition, I wrote the main code file which can run each model with the hyperparameters and any feature selection performed in tuning the algorithms and then will fit on the training data set, predict on the test data set, produce a confusion matrix for the predictions and print the accuracy, precision, recall and f1 score of the model's performance.

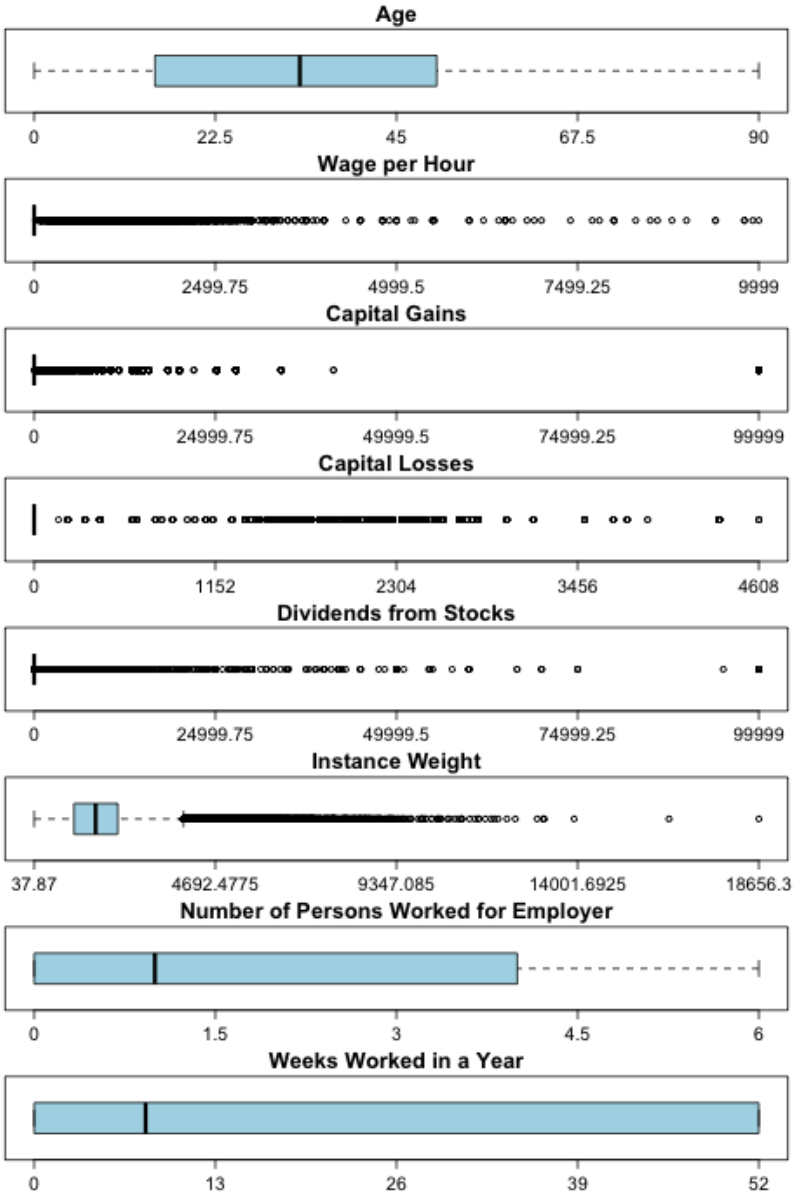
APPENDIX 2: DATA EXPLORATION AND CLEANING

Below is the full list of predictor features in the census data set, separated by numeric vs. categorical.

Numeric Features
Age
Wage per Hour
Capital Gains
Capital Losses
Dividends from Stocks
Instance Weight
Number of Persons Worked for Employer
Weeks Worked in a Year

Categorical Features
Class of Worker
Industry Code
Occupation Code
Education Level
Enrolled in Educational Institution Last Week
Marital Status
Major Industry Code
Major Occupation Code
Race
Hispanic Origin
Sex
Member of a Labor Union
Reason for Unemployment
Full or Part Time Employment Status
Tax Filer Status
Region of Previous Residence
State of Previous Residence
Detailed Household/Family Status
Detailed Household Summary in Household
Migration Code - Change in MSA
Migration Code - Change in Region
Migration Code - Move within Region
Live in this House 1 Year Ago
Migration Previous Residence in Sunbelt
Family Members Under 18
Country of Birth - Father
Country of Birth - Mother
Country of Birth - Self
Citizenship
Own Business or Self Employed
Fill Inc Questionnaire for Veteran's Admin
Veteran's Benefits
Year

Below are the univariate plots of the features, along with the proportion that were missing and the imputed value using na.roughfix().



Missing: 0.00%
Imputed Value: 33

Missing: 0.00%
Imputed Value: 0

Missing: 0.00%
Imputed Value: 0

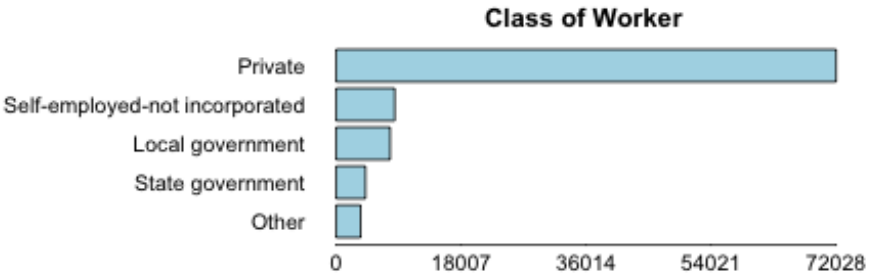
Missing: 0.00%
Imputed Value: 0

Missing: 0.00%
Imputed Value: 0

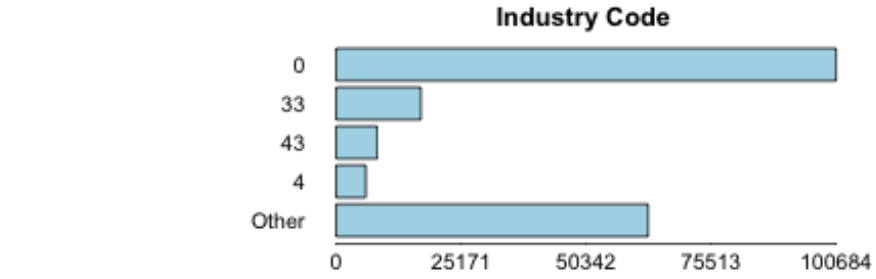
Missing: 0.00%
Imputed Value: 1618.31

Missing: 0.00%
Imputed Value: 1

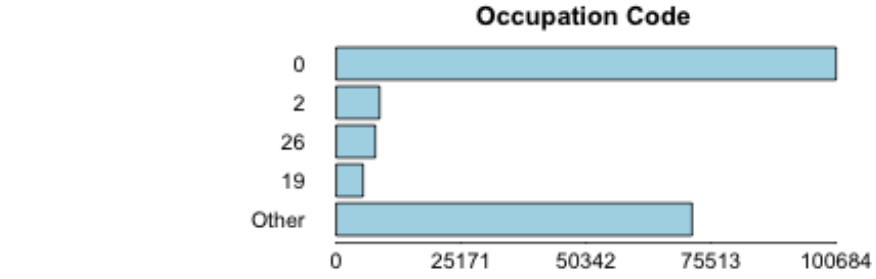
Missing: 0.00%
Imputed Value: 8



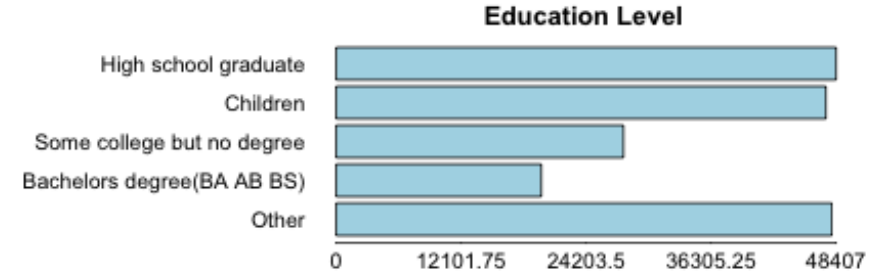
Missing: 50.24%
Imputed Value: Private



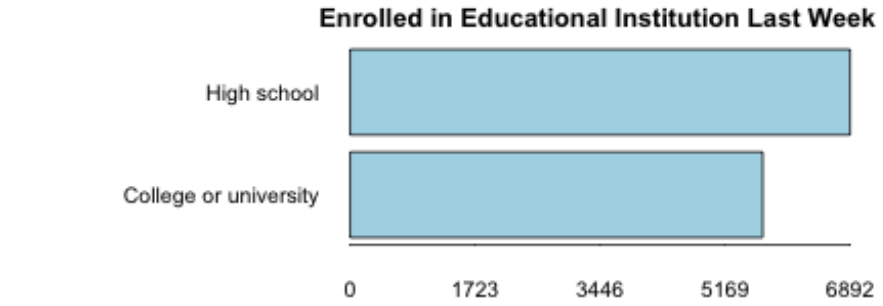
Missing: 0.00%
Imputed Value: 0



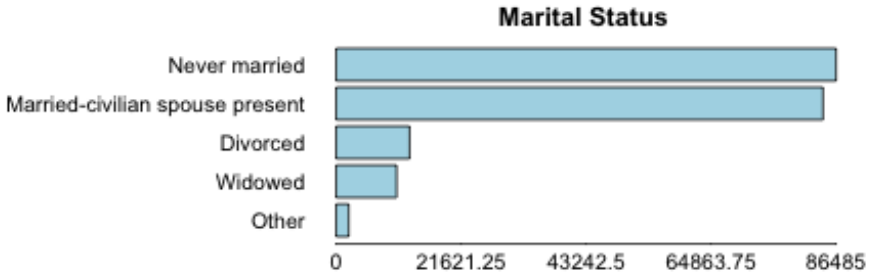
Missing: 0.00%
Imputed Value: 0



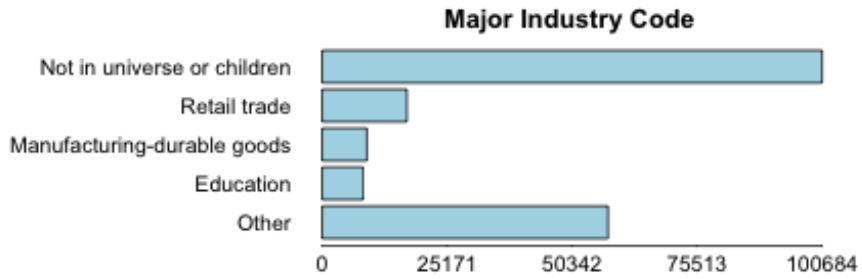
Missing: 0.00%
Imputed Value:
High School Graduate



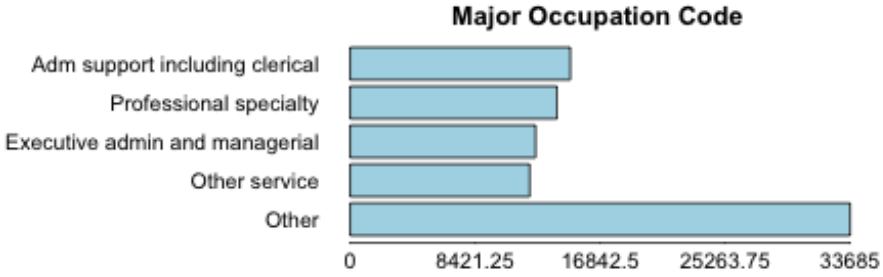
Missing: 93.69%
Imputed Value: High School



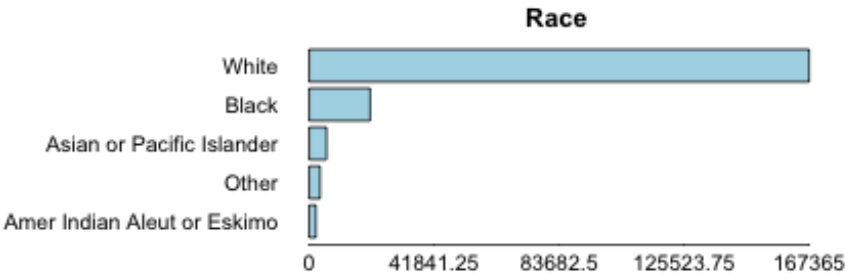
Missing: 0.00%
Imputed Value: Never Married



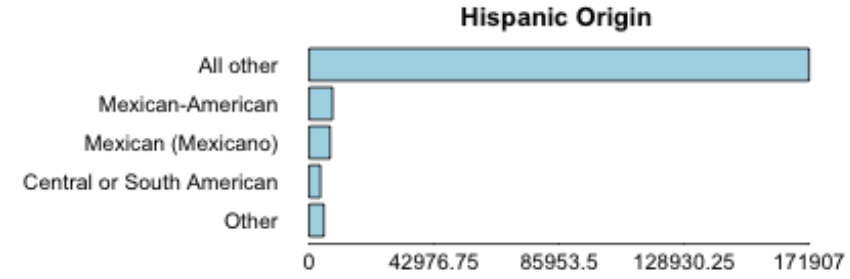
Missing: 0.00%
Imputed Value: Not in universe or children



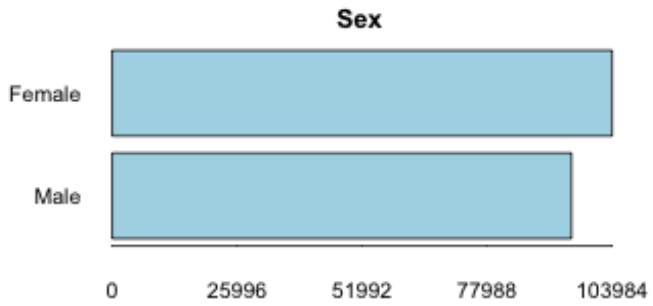
Missing: 50.46%
Imputed Value: Adm support including clerical



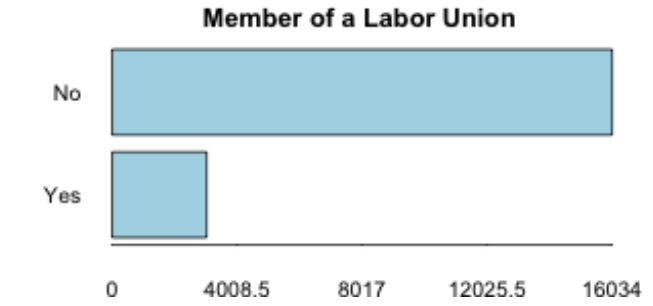
Missing: 0.00%
Imputed Value: White



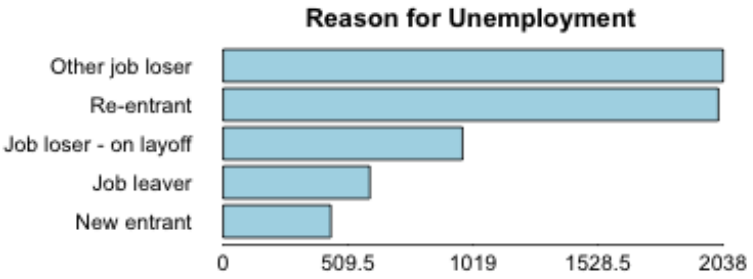
Missing: 0.00%
Imputed Value: All other



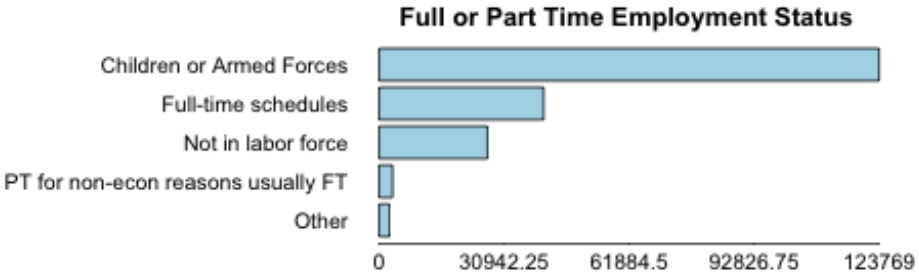
Missing: 0.00%
Imputed Value: Female



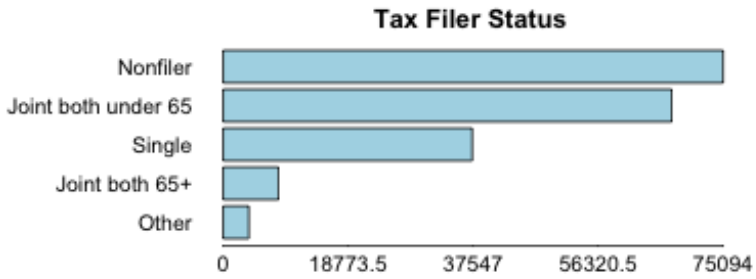
Missing: 90.445%
Imputed Value: No



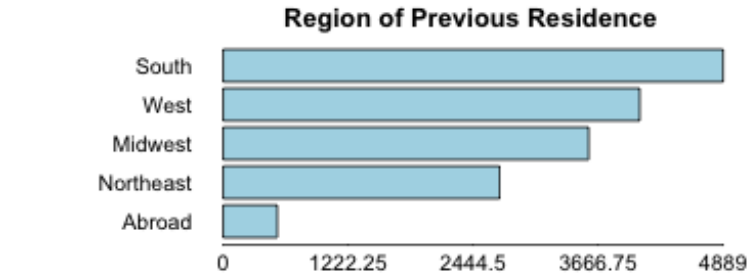
Missing: 96.96%
Imputed Value: Other job loser



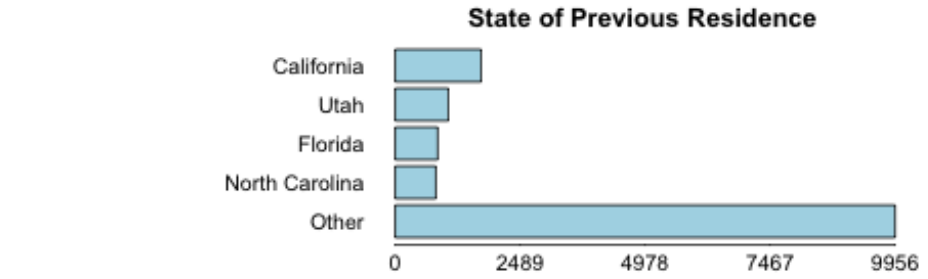
Missing: 0.00%
Imputed Value: Children or armed forces



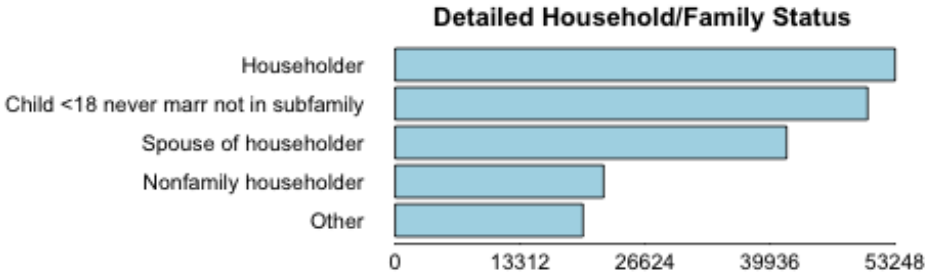
Missing: 0.00%
Imputed Value: Nonfiler



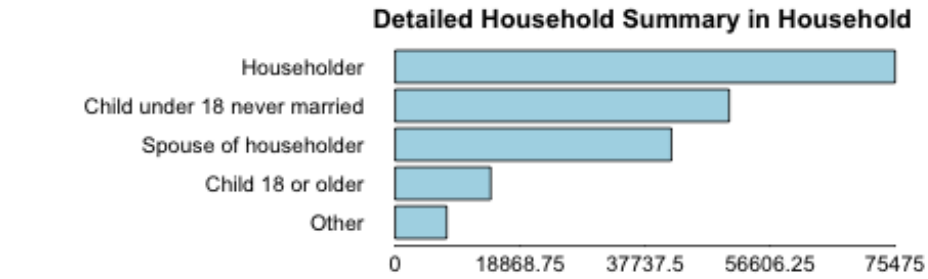
Missing: 92.09%
Imputed Value: South



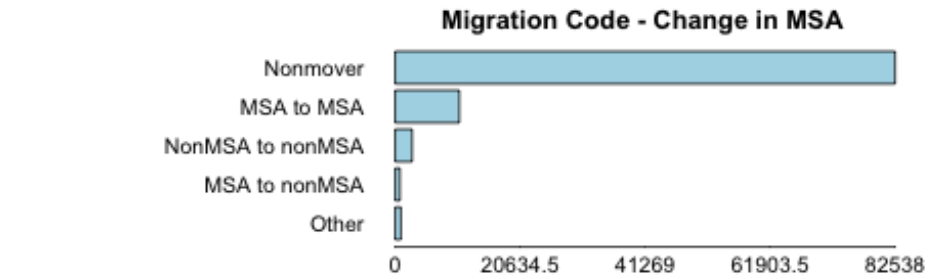
Missing: 92.45%
Imputed Value: California



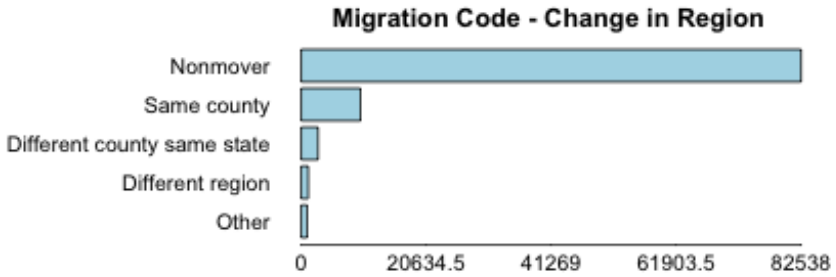
Missing: 0.00%
Imputed Value: Householder



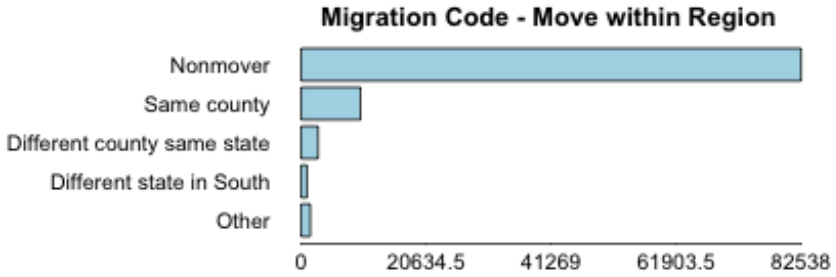
Missing: 0.00%
Imputed Value: Householder



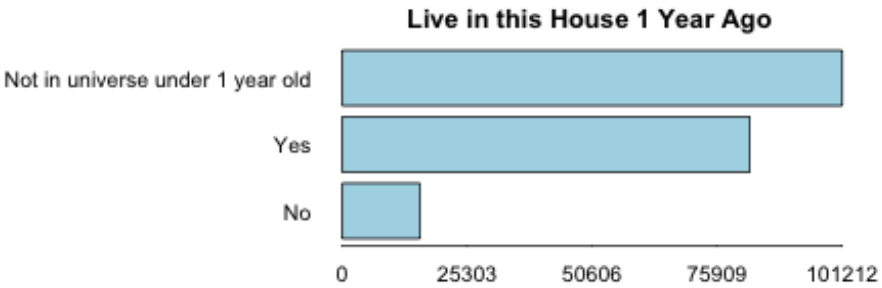
Missing: 50.73%
Imputed Value: Nonmover



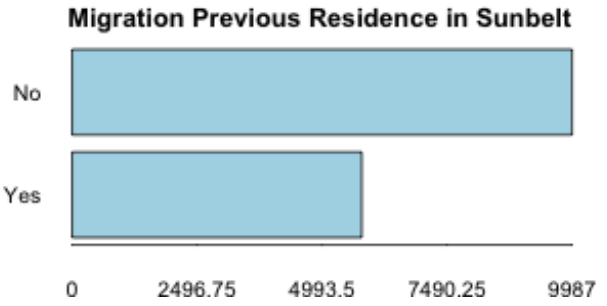
Missing: 50.73%
Imputed Value: Nonmover



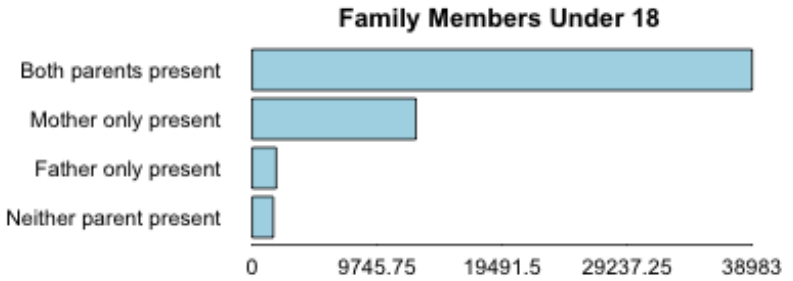
Missing: 50.73%
Imputed Value: Nonmover



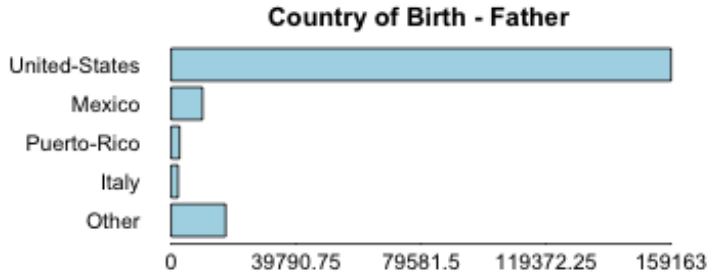
Missing: 0.00%
Imputed Value: Not in universe under 1 year old



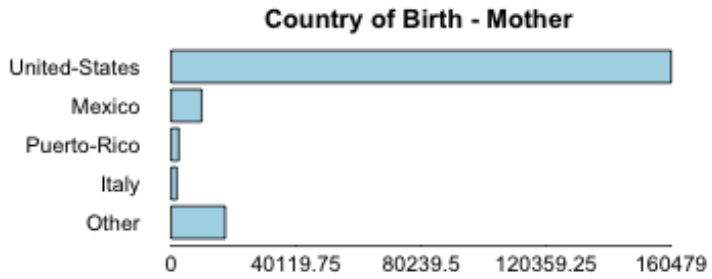
Missing: 92.09%
Imputed Value: No



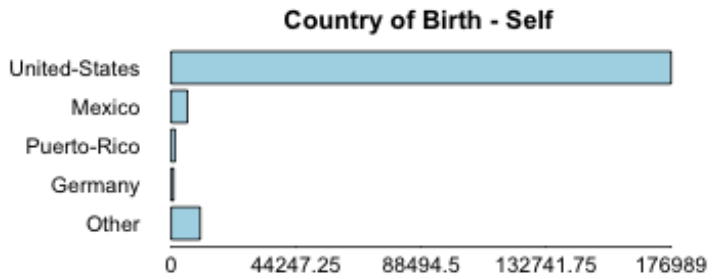
Missing: 72.29%
Imputed Value: Both parents present



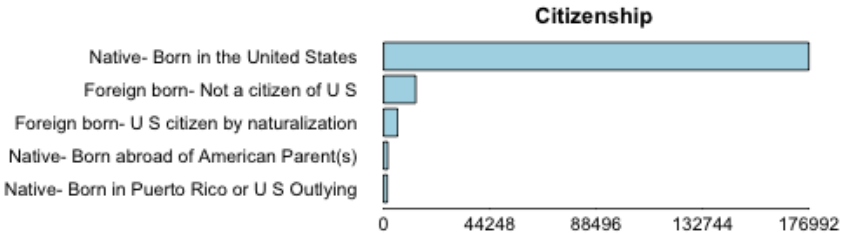
Missing: 3.36%
Imputed Value: United-States



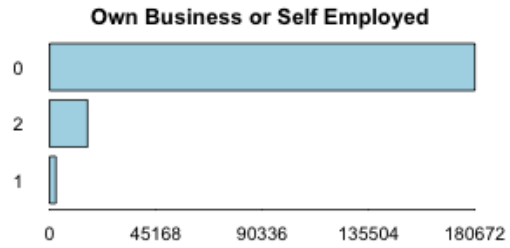
Missing: 3.08%
Imputed Value: United-States



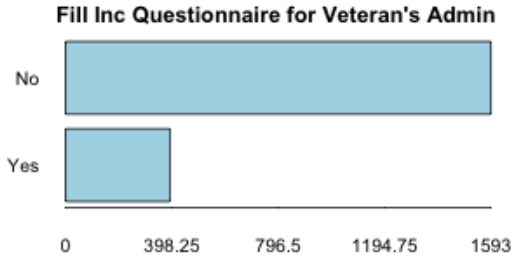
Missing: 1.70%
Imputed Value: United-States



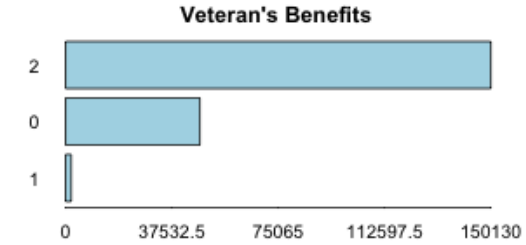
Missing: 0.00%
Imputed Value: Native- Born in the United States



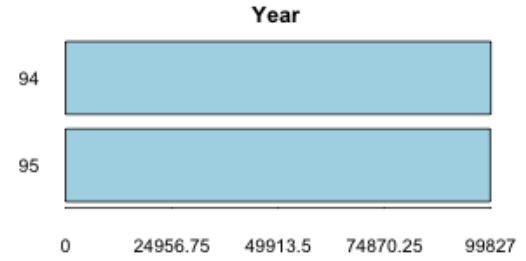
Missing: 0.00%
Imputed Value: 0



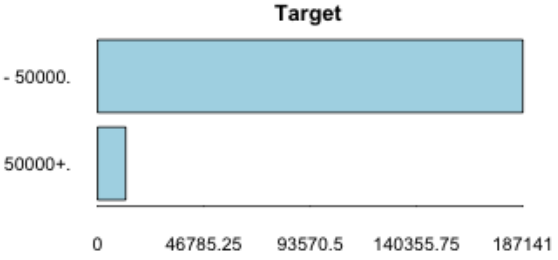
Missing: 99.01%
Imputed Value: No



Missing: 0.00%
Imputed Value: 2



Missing: 0.00%
Imputed Value: 94



Missing: 0.00%
Imputed Value: - 50000.

APPENDIX 3: RANDOM FOREST IMPUTATION

In imputing missing values we experimented by using a Random Forest classifier for those (seven) variables with ≥ 90% observations missing. The 5-fold cross validation (f1 and accuracy) scores for each variable's Random Forest predictions follow (computed on those observations not lacking values) along with a plot of the overall KNN and Logistic Regression performance with the Random Forest imputation versus the mean/median imputation. The performance did not warrant our use of this time and resource intensive technique in our final model evaluation.

Variable	F1	Accuracy
enrolled.in.edu.inst.last.wk	0.475	0.964
member.of.a.labor.union	0.922	0.862
reason.for.unemployment	0.546	0.547
region.of.previous.residence	0.499	0.504
state.of.previous.residence	0.181	0.195
migration.prev.res.in.sunbelt	0.778	0.696
fill.inc.questionnaire.for.veteran.s.admin	0.881	0.791

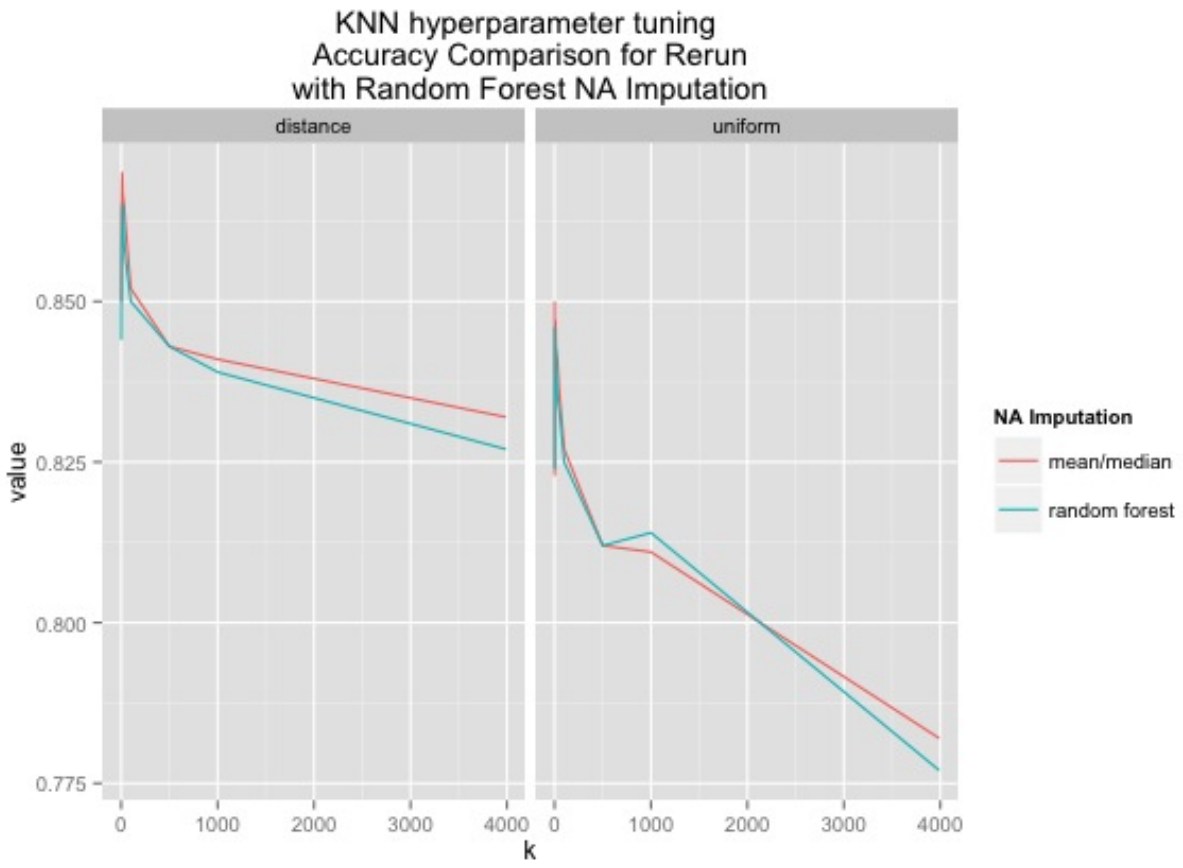


Figure 4


Logistic Regression hyperparameter tuning
Accuracy Comparison for Rerun
with Random Forest NA Imputation
l1
l2
value
0.870
0.865
0.860
0.855
0.850
0
1000
2000
3000
4000
C
NA Imputation
mean/median
random forest

Figure 5