Prediction of Adult income

M.S. Data Analytics
Project report CISC 6930: Data Mining

Fall 2017

Submitted by:

Matias Berretta
Vaibhav Dixit
Rohini Mandge

Professor & Advisor: Yijun Zhao
Fordham University

Index

1. Overview

Project Description

This Data Mining project uses the real-world dataset extracted from 1994 census bureau database to predict annual incomes for adults in the United States, given a set of attributes like employment details, demographic information etc.

The income level is classified in two classes – less than or equal to 50,000 (0) and greater than 50,000 (1).

This code for this project was written in Python and R.

Summary of results

For this project, we created three ensemble classifiers and tested them on three processed data sets, each of which underwent a different imputation method. For our optimal model we chose our third finalist model, Random Forest-imputed data paired with Ensemble C, which consisted of all five classifiers (K-nearest neighbors, Random Forest, SVM, Logistic Regression and Naïve Bayes), provided the highest accuracy score. However, our second finalist, KNN-imputed data paired with Ensemble B, which consisted of three classifiers (K-nearest neighbors, Random Forest and Naïve Bayes), came pretty close to the optimal model in terms of accuracy, actually yielding the best results for precision, F1-score, false positive rate, and true negative rate.

2. Problem Statement

Using 1994 census bureau data, the aim is to build a predictive model that determines income level for adults. Income level is a binary target variable which indicates whether an individual makes less than or equal to $50K or greater than $50K on annual basis.

This project requires you to explore classification algorithms on a real-world dataset, and write a report explaining your experimental results. The language of implementation can be anything — the only requirement is that your program be able to interpret the given data, and be able to classify instances and produce interesting statistics.

The algorithm should be based on the classification algorithms learned during the course. Usually a straight forward implementation of one method will not lead to satisfactory performance. Also, the algorithm can be a combination of methods and should incorporate one or more data mining techniques when the situation arises. These techniques include (and certainly not limited to):
– Handling imbalanced dataset
– Proper imputation methods for missing values
– Different treatment of various type of features: continuous, discrete, categorical, etc.

3. Exploratory Data Analysis

3.1 Variables:

Our census data contained 15 variables of three distinct types: continuous, categorical and ordinal. We only had one ordinal value, *education,* which was originally named *education-num* and described as continuous. We deleted the original *education* variable, which was recorded in String format, opting instead to use its neighboring column, *education-num,* as the new *education* variable because it translated its neighbor's String values into their corresponding integer values. Since our new *education* variable consisted of discrete integers whose order was determined by level of education, we chose to treat it as an ordinal variable (i.e. "HS-grad" translated to an integer value of 9, whereas "Bachelors" translated to an integer value of 13).

Continuous Variables

- age
- fnlwgt
- age
- capital-gain
- capital-loss

Categorical Variables

- workclass
- education (DELETED)
- marital-status
- occupation
- relationship
- sex
- race
- native
- country

Ordinal Variables

- education-num

3.2 Missing values:

After a preliminary exploration of the census data, we found that both training and testing data sets contained missing values. For training and test data alike, all of the missing values were confined to three categorical values: *native_country, workclass and occupation*. ~7.4% of training data instances, that is, 2399 rows, contained missing values whereas ~7.5% of test data instances, that is, 1221 rows, contained missing values.

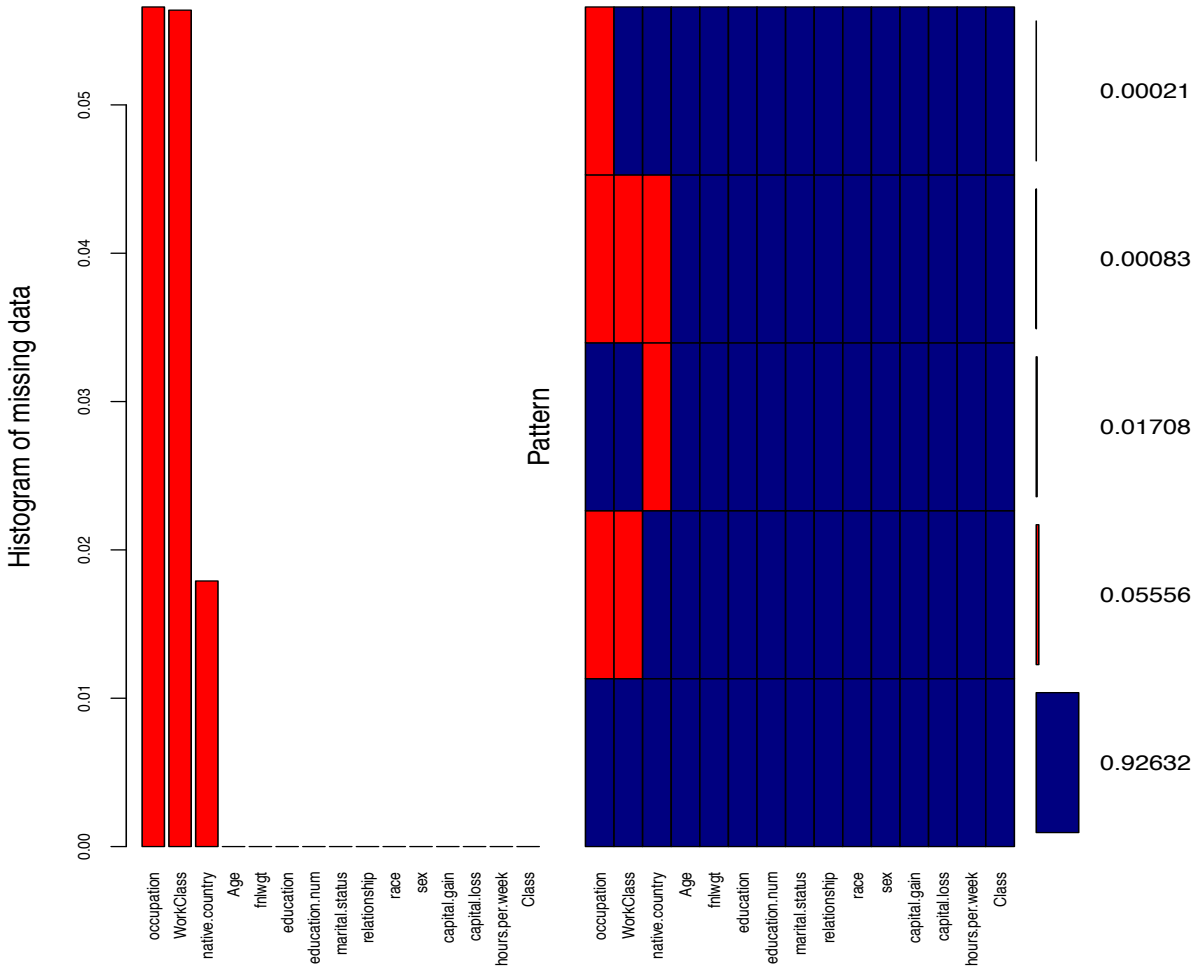


Figure 1 Missing Values

3.3 Unbalanced data:

The training data was unbalanced with a negative skew, where 75.9% of the instances were classified as negative and only 24.1% were classified as positive.

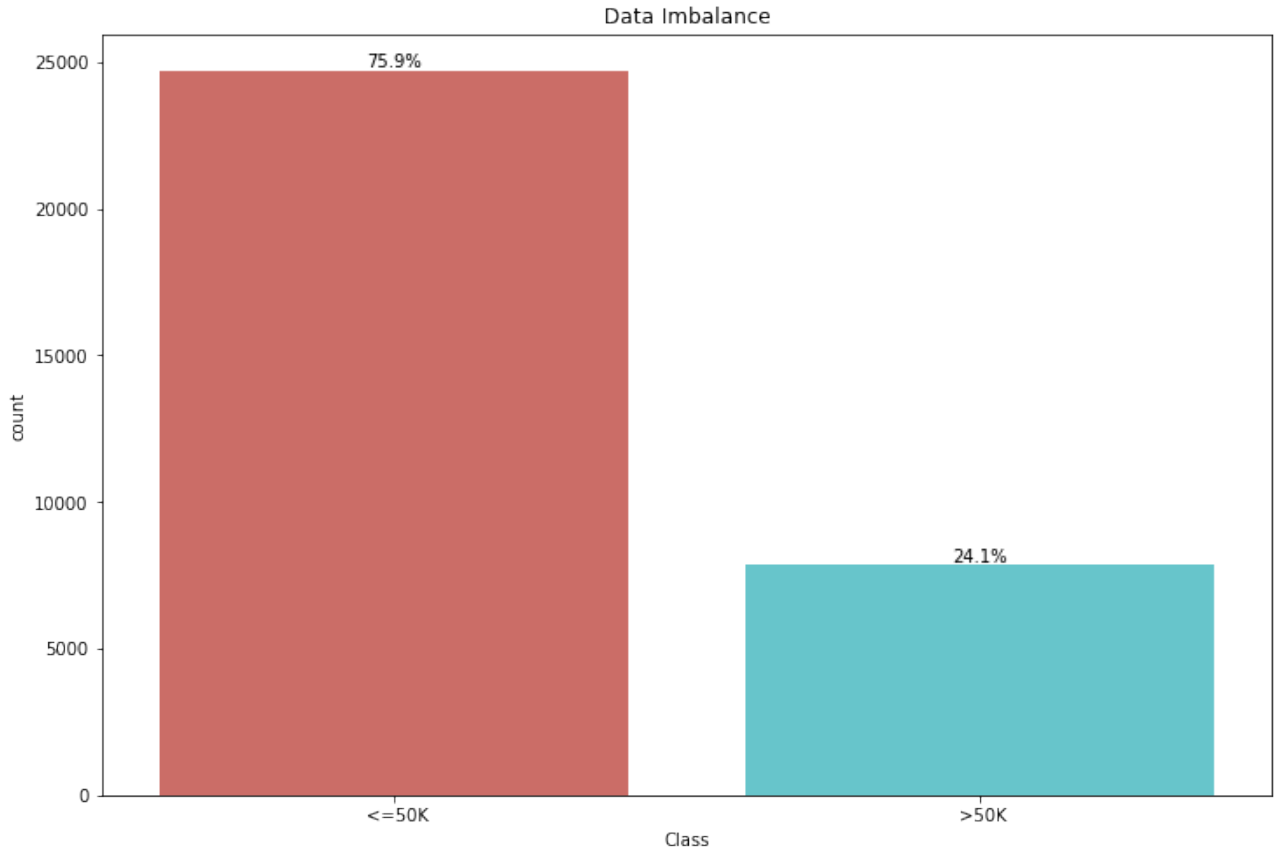


Figure 2 Unbalanced Data

3.4 Feature correlation

Figure 3 shows the correlation between the features and class variables.

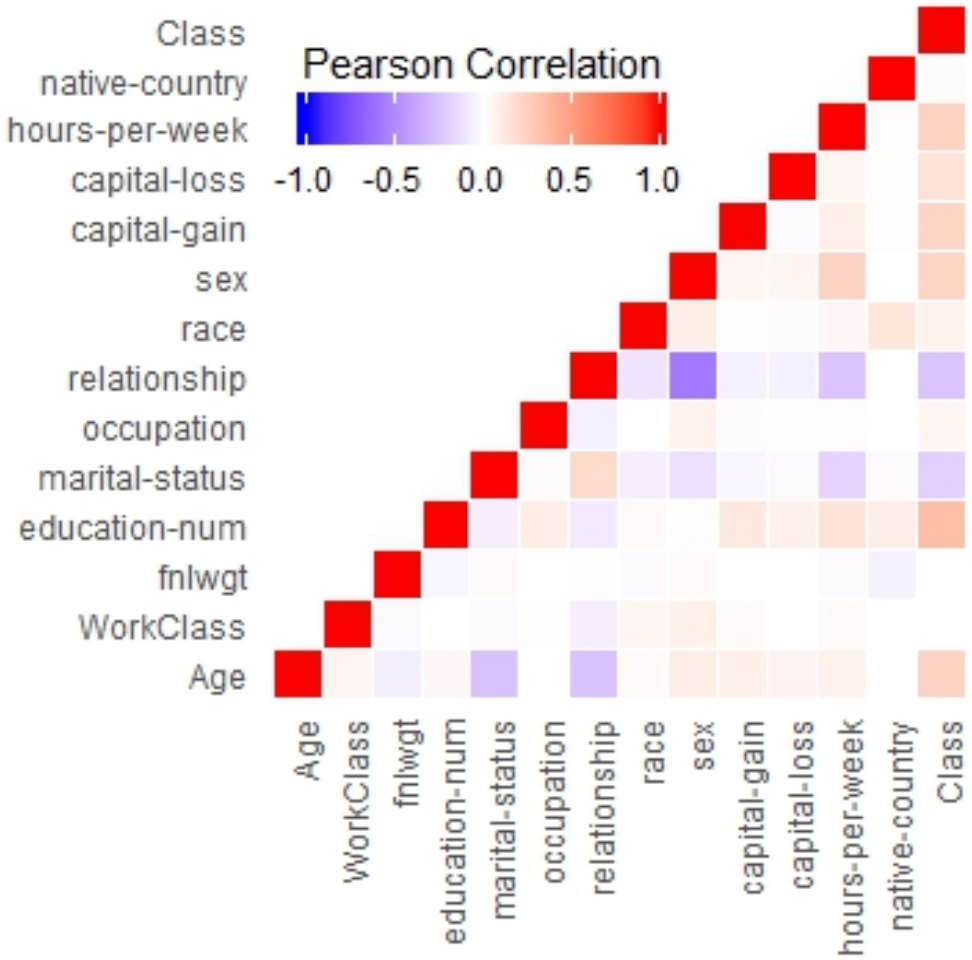


Figure 3 Feature Correlation

4. Ensemble Model

For this project, we took the Ensemble method approach to building our predictive models; the aggregate opinion of several algorithms is more varied and less biased than that of any single algorithm. To this effect, we sought to reduce the danger overfitting. In other words, the Ensemble approach yields a robust and high performing prediction model.

For this project, we built and tested three different ensemble classifier models. Each of our ensemble models was comprised by a different combination of the five supervised learning classifiers we learned about over the span of our Data Mining course. Furthermore, each of these combinations consisted of an odd number of algorithms so that every prediction would be decided by a clear majority vote.

We chose from a set of five classic classifying algorithms: *KNN, Random Forest, Naïve Bayes, SVM and Logistic regression*.

Ensemble A
Ensemble A consisted of *KNN, Logistic Regression and Random Forest*.
Since these models are highly correlated algorithms, (Random Forest and Logistic Regression were the least correlated algorithms at 58%) we expected Ensemble A to perform comparatively worse than the other algorithms.

Ensemble B
Ensemble B consisted of *Random Forest, Naïve Bayes and KNN*.
Since Naïve Bayes is not very highly correlated with Random forest and KNN (40% and 43% respectively) we expected it to produce better results than Ensemble A.

Ensemble C
Ensemble C consisted of *KNN, Random Forest, Naïve Bayes, SVM and Logistic Regression*. Since this model contained the most algorithms, we expected it to produce some of our best results.

	Pearson Coefficient Correlation (Mode Training Data)				
	KNN	**LR**	**NB**	**RF**	**SVM**
KNN	1	0.62	0.43	0.67	0.67
LR		1	0.45	0.58	0.84
NB			1	0.41	0.46
RF				1	0.61
SVM					1

Table 1: Results of Algorithm Correlations

5. Trial Run

In this approach, all missing values are dropped. The data is z-score normalized and ignored the unbalanced nature of the data in order to run preliminary tests and set a baseline for the classifier performance.

The results obtained for trial run are as follows

Data	Ensemble	Accuracy	Precision	Recall	F1-score
DropNa	A	86.26%	48.04%	93.69%	63.51%
DropNa	B	86.55%	50.12%	92.32%	64.97%
DropNa	C	86.31%	48.14%	93.87%	63.64%

Table 2: Trial Run Results table

6. Data Pre-processing

6.1. Encoding

As we found during our initial data analysis, there were few categorical features in the given dataset. There are many machine learning algorithms which can support categorical feature in computation without any manipulations but there are many more which do not support. Machine learning algorithm use for this project does not support the categorical feature directly and requires further manipulation in the data. Therefore, we had to figure out how to turn these categorical features into numerical features for algorithm processing.

There are many ways to encode the categorical features into numerical. As with many other aspects of data science, there is no best approach for categorical data encoding. Every approach has its trade-off and potential

impact on analysis outcome. Therefore, we tried multiple ways to encode our data and measured its outcome by running logistic regression classifier with cross validation technique.

For our implementation, we used a built-in python package "**CategoryEncoders**", which provides different techniques to encode the categorical data - One-Hot, Ordinal, Binary, Backward Difference, BaseN, Hashing, Helmert, Leave-One-Out, Polynomial, Sum encoding and default dummy encoding. We then used the encoded data to run logistic classifier with cross validation to measure the encoding technique impact. Under mentioned is the output of our observation with random forest imputed dataset with z-score normalization.

Encoding Type	Dimensionality	Accuracy	Precision	Recall	F1-score
BackwardDifferenceEncoder	89	81.70%	28.13%	83.42%	42.08%
BaseNEncoder	29	85.53%	60.76%	73.40%	66.49%
BinaryEncoder	29	85.41%	60.66%	73.00%	66.26%
HashingEncoder	14	84.31%	48.62%	76.36%	59.41%
HelmertEncoder	89	85.34%	64.53%	70.81%	67.53%
LeaveOneOutEncoder	13	85.36%	56.21%	75.57%	64.47%
OneHotEncoder	96	85.28%	64.17%	70.80%	67.32%
OrdinalEncoder	13	84.51%	48.31%	77.68%	59.57%
PolynomialEncoder	89	85.28%	63.10%	71.28%	66.94%
SumEncoder	89	85.42%	64.30%	71.19%	67.57%
DummyEncoder	89	85.51%	64.12%	71.58%	67.65%

Table 3 Different types of Encoding

According to the above observation, BaseNEncoder type encoding is giving us the highest performance. However, when running the same model on different versions of dataset, DummyEncoder results in consistent performer over other encoder types. Therefore, we choose Dummy Encoder to encode our categorical features.

6.2. Imputation

To handle the missing training data, there are two categories of techniques exists, model based and non-model based approaches. Non-model based techniques includes mean imputation and hot-deck imputation. These techniques generally decrease the variance estimates in statistical procedures.

Furthermore, these techniques also result in standard errors and bias in results. On the other hand, model based approaches includes data mining algorithm techniques to predict the missing values. (For ex - Regression model, decision tree, NB etc). This approach results decreasing the variance as well as bias.
For our project, we used three methods to impute the missing data which includes model based and non-model based techniques.

1. Mode

This is the example of a non-model based approaches. According to this approach, we fill the missing feature values with the most frequent data for the respective feature.

2. Random-Forest Based Imputation

This is another example of a model based approaches where multiple D-trees are built which contains information corresponding to attributes in the given dataset. This information is used to follow a given set of input attribute, depending on categorical nodes. After which, "Random forest" algorithm is used to generalize ensembles of D-trees through bagging which combines multiple random D-trees to aggregate the prediction for missing data.
For our implementation, we used a built-in package of R (missForest) for Random Forest based imputation. In missForest package, for each feature (Having missing values), it builds the random forest based on the given observations and then predicts the missing values. The algorithm continues to repeat these two steps until stopping criteria is met or the user specific maximum of iterations is reached.
According to default stopping criteria, after each iteration the difference between the previous and the new imputed data matrix is assessed for the continuous and categorical features. The default stopping criteria is defined such that the imputation process is stopped as soon as both differences have become larger once. In case of only one type of variable the computation stops as soon as the corresponding difference goes up for the first time. However, the imputation last performed where both differences went up is generally less accurate than the previous one. Therefore, whenever the computation stops due to the stopping criterion the before last imputation matrix is returned. In our case, we only had the missing values in categorical features, therefore second procedure applies.

Additionally, missForest algorithm provides the user with an estimate of the imputation error. This estimation is based on out-of-bag (OOB) error estimate of random forest.

https://stat.ethz.ch/education/semesters/ss2013/ams/paper/missForest_1.2.pdf

3. KNN Based Imputation

This is the example of a model based approaches. K nearest neighbor method is based on the given training observations, an aggregation of the k values of the nearest neighbors is used as the imputes values. (In this method, the aggregation type depends on the type of variable).

For our implementation, we used a built-in package of R (VIM) for KNN based imputation. In VIM package, the distance computation for defining the nearest neighbor is based on an extension of the grower's distance algorithm, which can handle distance variable of type binary, categorical, ordered, continuous and semi-continuous. In our project, all the missing value features were categorical type, therefore we used grower's distance function for to compute the distance between the observations.

Under this approach, there are two methods to impute the missing data using k nearest neighbor observations. First method of aggregation is to use the category with the most occurrences in the k nearest neighbors, if this results in a tie, a category from the tied categories will be randomly chosen. Second method of aggregation is to sample the category from the categories in the k nearest neighbors with probabilities equal to the occurrences in the k values.

For our implementation, we used the first method is aggregation to impute the missing values with the K value as "5". We also tried the imputation with different values of K as 3,5,7,10 and 15 while measuring the model performance using logistic regressions classifier with cross validation. However, there were no significant gain by increasing the K value. Therefore, we fix our KNN imputation model with K value as "5".

https://cran.r-project.org/web/packages/VIM/VIM.pdf

The most successful imputation approach is selected and data is balanced with the Bagging Classifier Method.

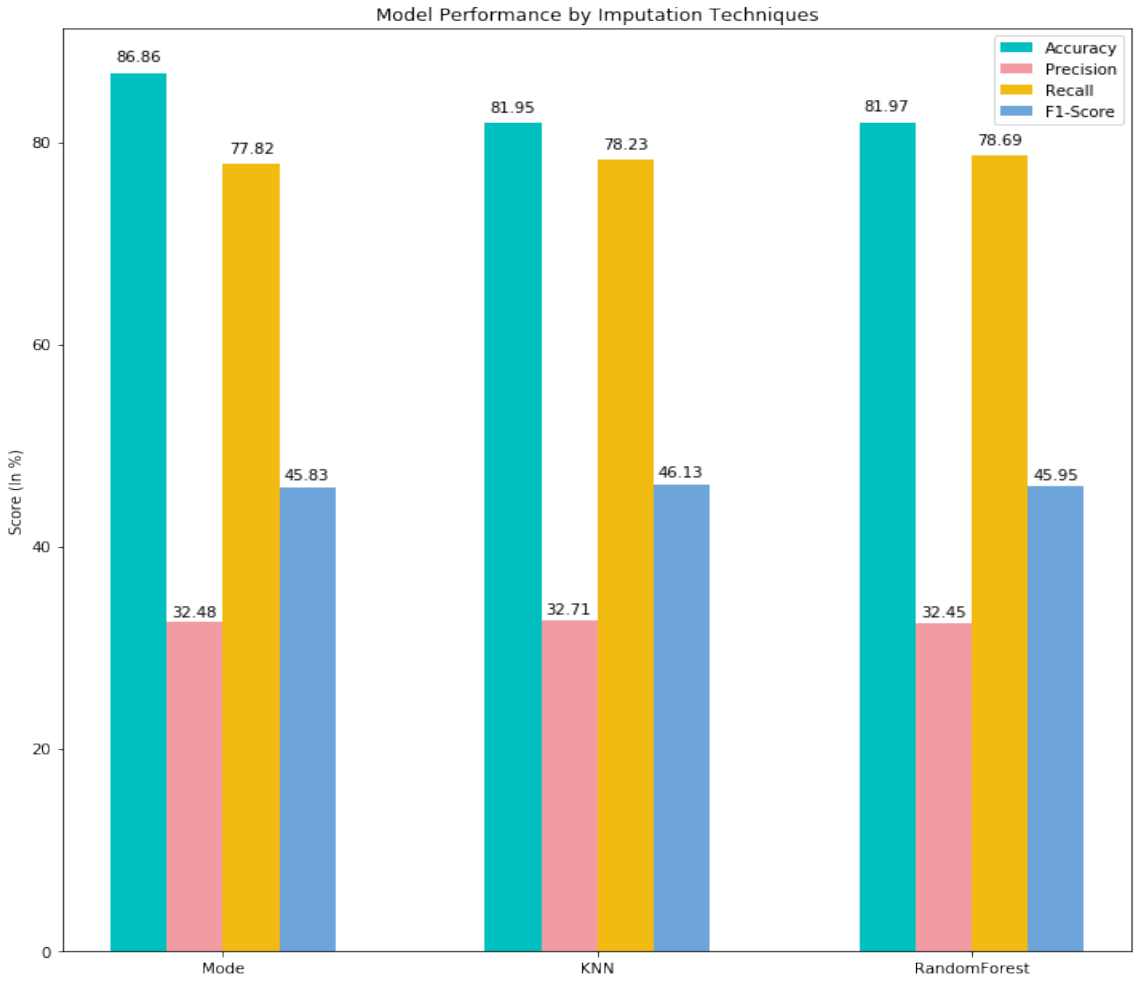
6.3. Balancing the Data with Bagging

We balanced our data with the bagging approach (also called as Bootstrap aggregating). It also reduces variance and helps to avoid overfitting. Bagging is a special case of the model averaging approach.

6.4. Z- Score Normalization

In normalization, giving scores a common standard of zero mean and unity standard deviation facilitates their interpretation. This is a common procedure in statistics because values that roughly follow a standard normal distribution are easily interpretable. We use Z-score normalization which replaces the measurement unit with "number of standard deviations" away from the mean. Hence, it's a convenient tool when someone wants to compare two variables that are measured in different units.
Following plots (Figure 4 and 5) shows what difference the normalization made in our datasets.

Figure 4: Results Before Normalization



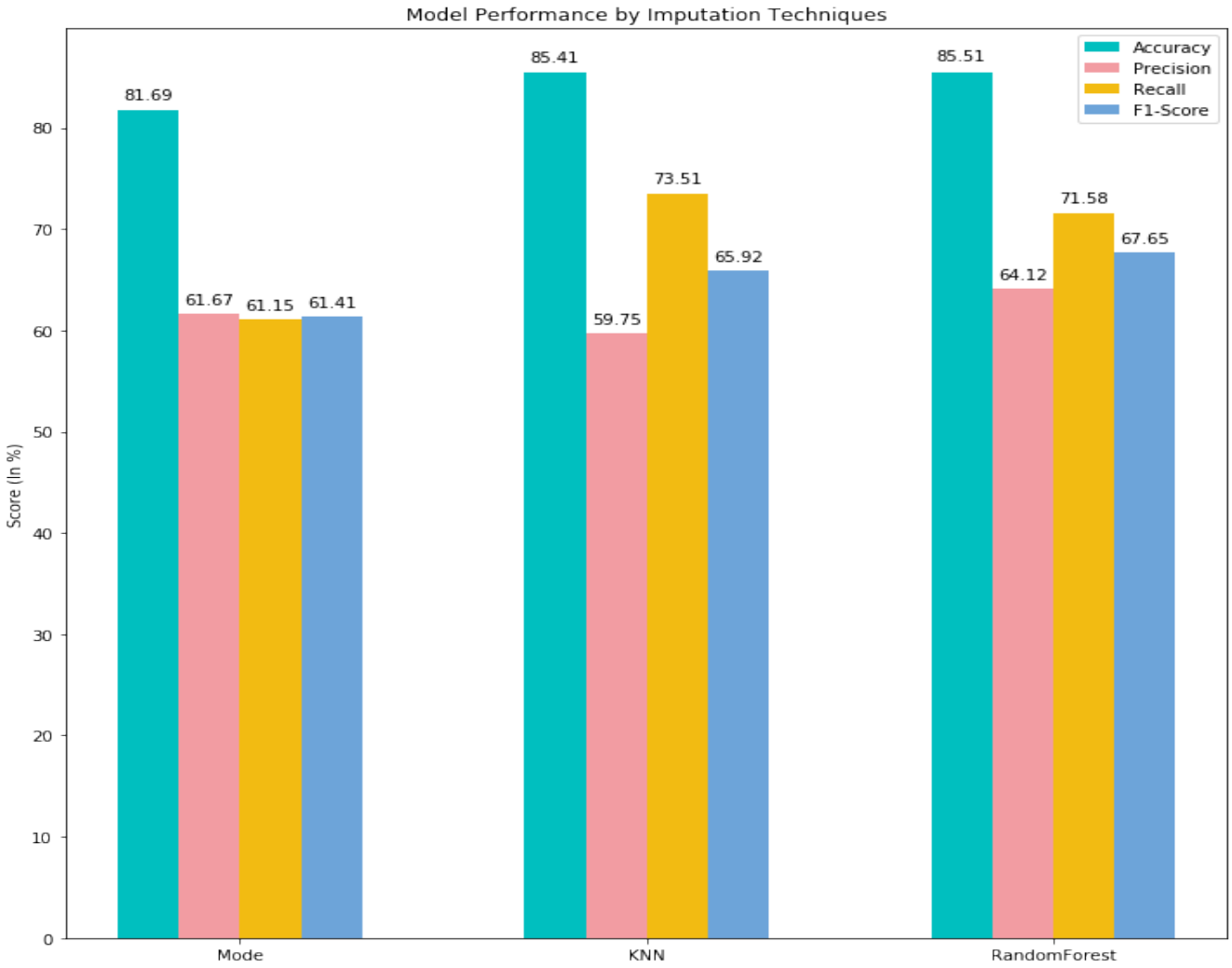


Figure 4: Results After Normalization

6.5. Parameter Optimization

After completing the bagging and normalization stages of the preprocessing pipeline we selected the highest performing ensemble model for each of the three imputed data sets. We then set out to optimize the parameters for the algorithms contained within each of these highest-performing ensemble classifiers. In order to find the optimal parameters for each different ensemble and data pairing, we split the training data and performed cross validation while iterating through different values for adjustable parameters of each individual algorithm contained within the ensemble.

For example, after bagging and normalization, Ensemble B yielded the best performance for our Mode-imputed data. Since Ensemble B consisted of three algorithms — KNN, Random Forest and Naïve Bayes — we iterated through

different values of number of neighbors for KNN and number of estimators for Random Forest (excluding Naïve Bayes from this process since it does not provide any adjustable parameters) using the normalized Mode-imputed training data. We then concluded that the optimal number of neighbor for KNN was 30 and that the optimal number of estimators for Random Forest was 50.

Ensemble	Data	KNN	Random Forest	Logistic Regression	SVM	Naïve Bayes
B	Mode	n_neighbors = 30	n_estimators = 50	N/A	N/A	N/A
B	KNN	n_neighbors = 50	n_estimators = 90	N/A	N/A	N/A
C	RF	n_neighbors = 15	n_estimators = 90	C = 10	C = 10, gamma = 0.1, kernel = 'rbf'	N/A

Table 4: Result table after Parameter Optimization

The results can be visualized in following figure 6

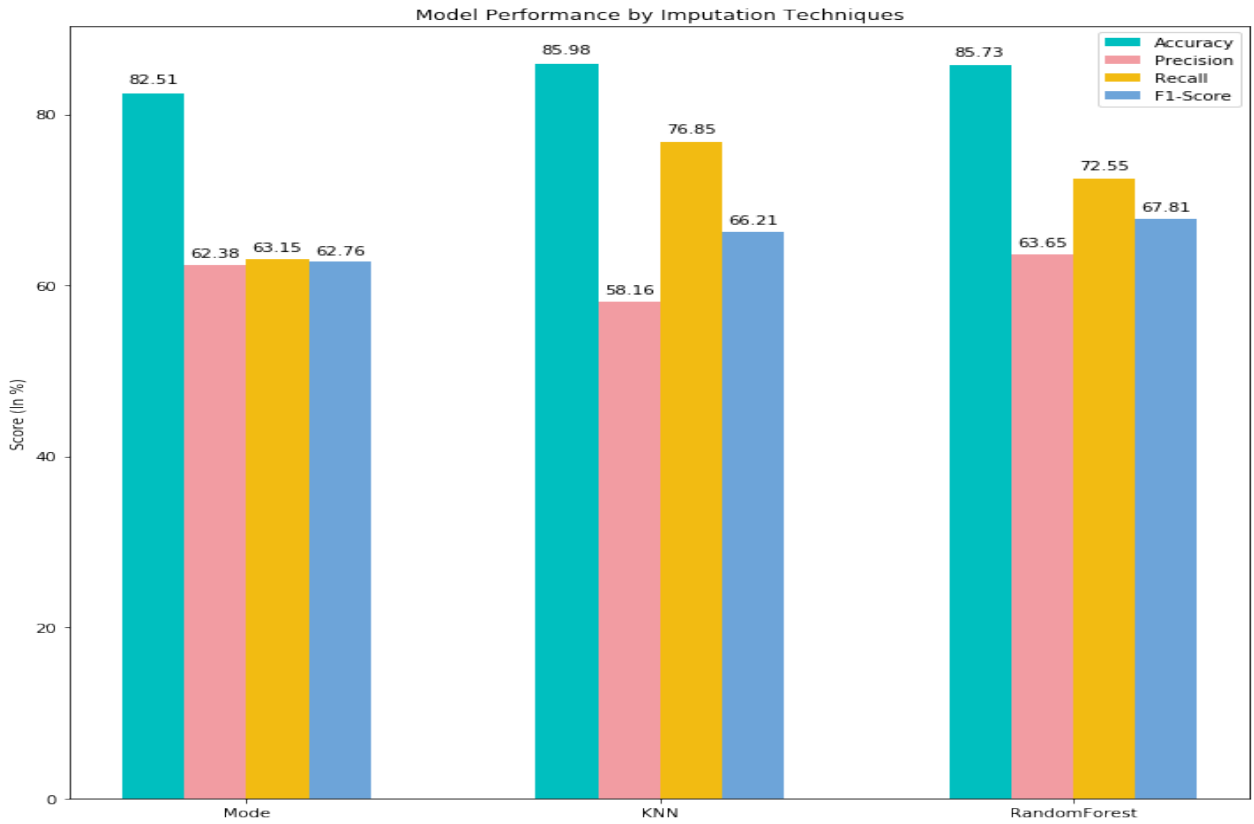


Figure 5: Plot of results after parameter optimization

6.6. Feature Selection

For our feature selection stage, we took the Filter Method approach, which ranks features by how closely correlated they are to the target variable (absolute value of Pearson Correlation Coefficient). Apart from evaluating each optimized model's performance by how well it fared when predicting with all features, we ran three additional tests to determine whether curating the amount and correlational relevance of each feature would improve our results. Each of these additional runs had a preset number of features as ranked by their absolute correlation value to the target variable: (1) test model with the first 20 most correlated features, (2) test model with the first 40 most correlated features, (3) and test model with the first 60 correlated features. These results were then compared to the performance results generated by using all features.

Reducing the number of features only improved the performance for the KNN Data, Ensemble B pairing, with the optimal feature number being 60.

Data	Method	Features	Accuracy	Precision	Recall	F1-score
Mode	Filter	20	81.59%	58.09%	61.71%	59.84%
Mode	Filter	40	82.48%	63.13%	62.87%	63.00%
Mode	Filter	60	82.38%	63.21%	62.59%	62.90%
Mode	Regular	All	82.51%	62.38%	63.15%	62.76%
KNN	Filter	20	85.19%	66.54%	69.48%	67.98%
KNN	Filter	40	84.98%	67.86%	68.34%	68.10%
KNN	Filter	60	84.91%	68.54%	67.89%	68.21%
KNN	Regular	All	85.98%	58.16%	76.85%	66.21%
Random Forest	Filter	20	85.34%	61.78%	72.17%	66.57%
Random Forest	Filter	40	83.42%	14.54%	20.92%	17.16%
Random Forest	Filter	60	85.41%	63.83%	71.39%	67.40%
Random Forest	**Regular**	**All**	**85.73%**	**63.65%**	**72.55%**	**67.81%**

Table 5: Result table after performing Feature Selection

The results can be visualized in following figure7.

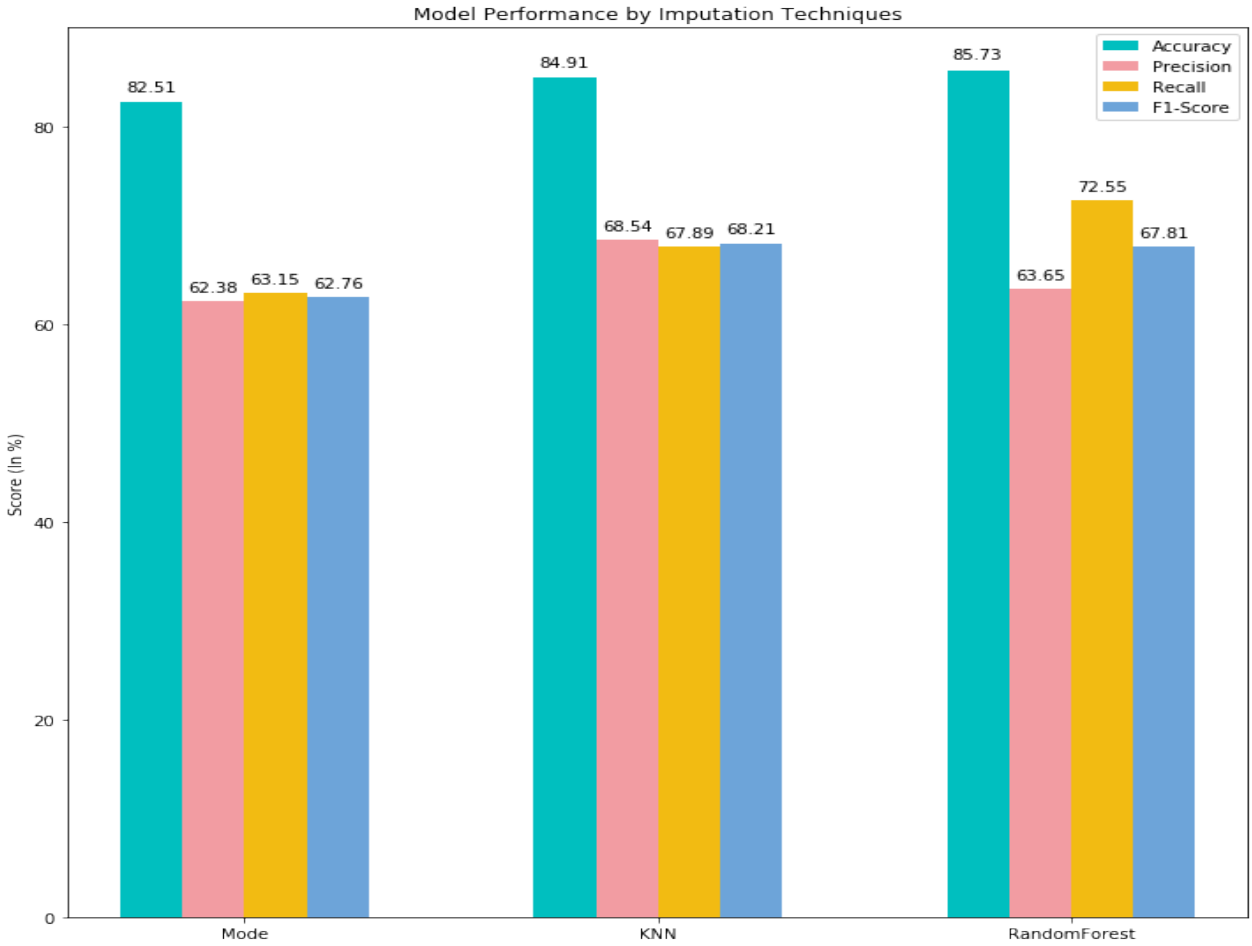


Figure 6: Plot of results after performing Feature Selection

7. Optimal Model

We ended up with three finalist models, one for each kind of processed data set (defined by method of imputation): Mode, KNN, and Random Forest data sets. The first optimal model paired Ensemble B with Mode data; the second paired Ensemble B with KNN data; and the third optimal model paired Ensemble C with Random Forest data.

We chose our third finalist model, which paired Ensemble C with Random Forest Data, as our optimal model since it yielded the highest results for accuracy (85.73% vs. 84.91% and 82.51%). However, this determination comes with a caveat.

Our second model, which paired Ensemble B and KNN imputation with 60 features, came pretty close to the optimal model in terms of accuracy, actually yielding the best results for F1-score, precision, false positive rate and true negative rate. Since in this project our task is just to predict the income, we chose to prioritize accuracy as a performance measure. If the real-world application of our project results were to determine whether an individual should be taxed more heavily depending upon their annual income, we would have chosen to prioritize precision and false positive rate over accuracy, since we wouldn't want anyone making less than $50,000 receiving an undue and unmerited increase on their tax bill based on an error in our predictive model. Similarly, if our task where to determine whether food was edible (1) or poisonous (0), we would rather err on the side of false negatives than false positives, since the price of wasted food is much lower than that of food poisoning and possible death.

From the negligible performance difference between our second and third finalist models, we gleamed the importance of algorithm diversification when building ensemble models. Although we expected Ensemble C to have a clear advantage over Ensemble B, the difference was not only small but also biased by our preference for accuracy as a performance measure. The fact that a three-algorithm ensemble classifier performed at the same level as a five-algorithm ensemble classifier indicates that the key to creating a powerful ensemble model is diversity of opinion among the voting algorithms. A ten-algorithm ensemble whose predictions always arise from a unanimous decision will fare far worse than an ensemble of three carefully curated algorithms whose predictions sometimes differ from one another.

On the subject of imputation, both the Random Forest and KNN approach displayed a clear advantage over the Mode method. Although the latter was the simplest and least computationally expensive approach to imputation, it fell short when it came across all performance values. Furthermore our comparison of Ensemble C and Ensemble B is not entirely fair since it is mediated by a difference in imputation method (Random Forest imputation vs. KNN imputation respectively). Still, Ensemble B's performance merits reflection and future exploration.

Legend Key	
Ensemble A	KNN + Logistic Regression + Random Forest
Ensemble B	KNN + Random Forest + Naïve Bayes
Ensemble C	KNN + Logistic Regression + Random Forest + Naïve Bayes + SVM
Finalist Models	Regular
Optimal Model	**Bold**

	Stages	Ensemble	Bagged	accuracy	precision	recall	f1-score
Mode	1) Bagging	Ensemble B	Yes	81.86%	32.48%	77.82%	45.83%
Mode	2) Z-score Normalized	Ensemble B	Yes	81.69%	61.67%	61.15%	61.41%
Mode	3) Params. Optimized	Ensemble B	Yes	82.51%	62.38%	63.15%	62.76%
Mode	4) Feature Selection: All	Ensemble B	Yes	82.51%	62.38%	63.15%	62.76%
KNN	1) Bagging	Ensemble B	Yes	82.10%	30.81%	82.35%	44.81%
KNN	2) Z-score Normalized	Ensemble B	Yes	85.41%	59.75%	73.51%	65.92%
KNN	3) Params. Optimized	Ensemble B	Yes	85.98%	58.16%	76.85%	66.21%
KNN	4) Feature Selection: 60	Ensemble B	Yes	84.91%	68.54%	67.89%	68.21%
Random Forest	1) Bagging	Ensemble B	Yes	82.10%	30.81%	82.35%	44.81%
Random Forest	2) Z-score Normalized	Ensemble C	Yes	85.51%	64.12%	71.58%	67.65%
Random Forest	3) Params. Optimized	Ensemble C	Yes	85.73%	63.65%	72.55%	67.81%
Random Forest	4) Feature Selection: All	Ensemble C	Yes	**85.73%**	**63.65%**	**72.55%**	**67.81%**

Table 6(A): The Final result table (Best Combinations At Each Stage for each Data Set)

Table 6(B): The Final result table

Data	Stages	Ensemble	Bagged	FPR	TNR
Mode	4) Feature Selection: All	Ensemble B	Yes	11.59%	88.41%
KNN	4) Feature Selection: 60	Ensemble B	Yes	9.76%	90.24%
Random Forest	4) Feature Selection: All	Ensemble C	Yes	10.83%	89.17%

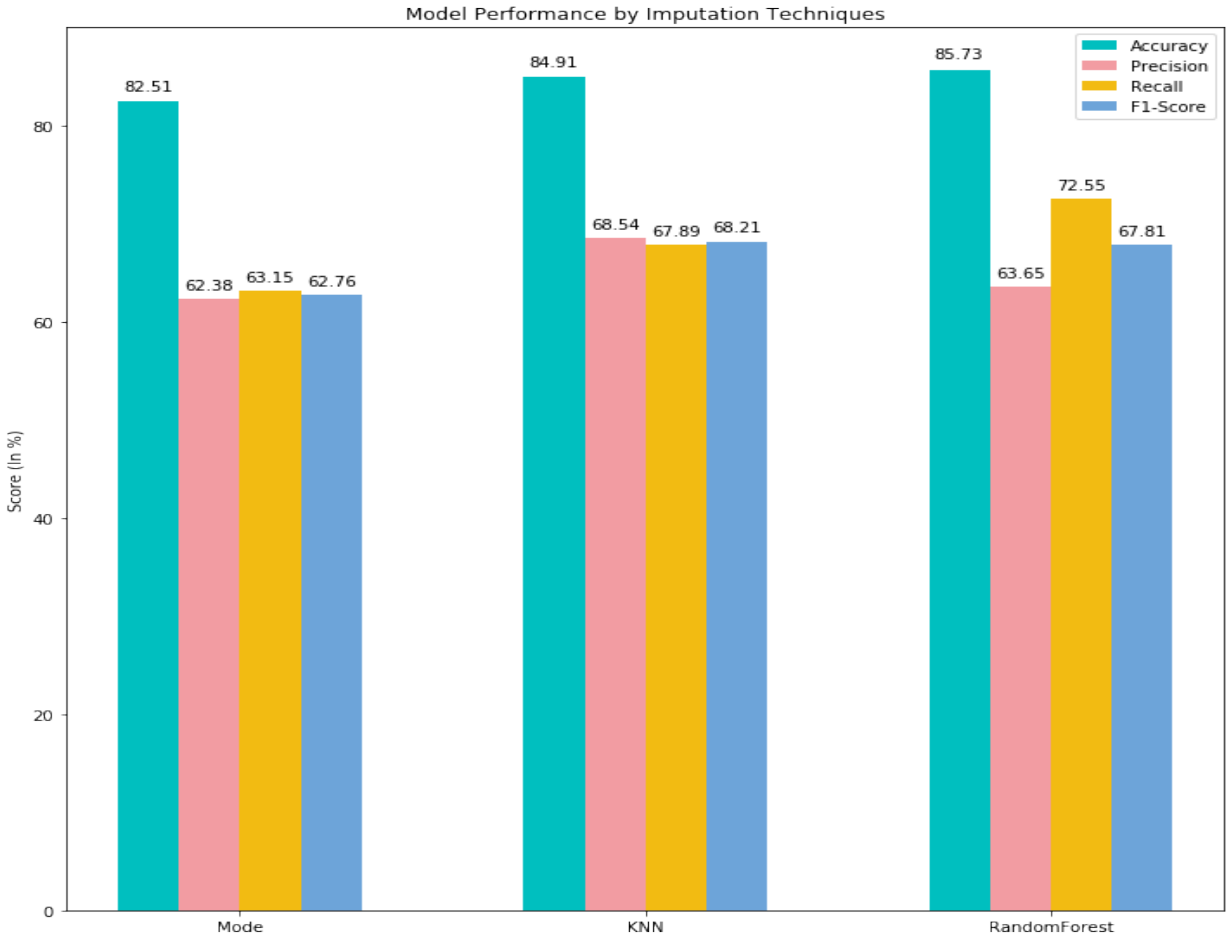


Figure 7: Final Results Plot

8. Future Work

In the future, we would like to try out a new way of balancing the data such as B(L)aging, which is short for Balanced Bagging. Furthermore, we would also like to approach feature selection with the Wrapper Method. Last but not least, we would hope to make use of neural networks in hopes of achieving higher performance values across the board.

9. Reference Links

- *https://github.com/mberrett/Data_Mining_Project*
- https://cran.r-project.org/web/packages/VIM/VIM.pdf
- https://link.springer.com/chapter/10.1007/978-3-642-14834-7_56
- https://www.youtube.com/watch?v=u8XvfhBdbMw
- https://github.com/statistikat/VIM
- https://www.analyticsvidhya.com/blog/2016/03/tutorial-powerful-packages-imputing-missing-values/
- https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=1946&context=gradreports
- https://svds.com/learning-imbalanced-classes/
- https://link.springer.com/chapter/10.1007/978-3-642-03156-4_6
- https://arxiv.org/ftp/arxiv/papers/0812/0812.2412.pdf