Prediction of Adult income

M.S. Data Analytics

## Project report CISC 6930: Data Mining

Fall 2017

Submitted by:

Professor & Advisor: Yijun Zhao

Fordham University

**Index**

|  |  |
| --- | --- |
| Content | Page |
| 1. [Overview](#Overview) | 3 |
| 1. [Problem Statement](#Problem_Statement) | 4 |
| 1. [Data Exploratoryanalysis](#Data_analysis) |  |
| * 1. [Missing Values](#missing_value) |  |
| * 1. [Unbalanced Data](#Unbalanced_data) |  |
| * 1. [Correlation](#Feature_correlation) |  |
| * 1. [Feature Types](#Types_of_variables) |  |
| 1. [Ensemble Models](#Ensemble_Models) |  |
| 1. [Trial Run](#trial_run) |  |
| 1. [Data Pre-processing](#Data_Preprocessing) |  |
| * 1. [Data Encoding](#data_encoding)   2. [Imputation](#imputation)   3. [Bagging](#bagging)   4. [Normalization](#Normalization) (w Bagging Iteration)   5. Parameter Optimization   6. Feature Selection |  |
| 1. Optimal Model |  |
| 1. [~~Training the model using data mining algorithms~~](#Algortithms) |  |
| 1. Reference documents links |  |
| 1. Reflection (Future Work) (B(l)agging) |  |

1. ***Overview***

**Project Description**

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

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

The project uses data various data mining algorithms and are written in Python and R.

**Summary of results**

1. ***Problem Statement***

Using 1994 census bureau data, the aim is to build a predictive model that determines income level for adults. Income levels are classified in two classes below $50K and above $50K annually (given in the dataset).

1. ***Exploratory Data Analysis***
2. 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

1. 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*.

7.45% 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 Missing Values

1. Unbalanced data:

The training data was unbalanced with a negative skew, where 77% of the instances were classified as negative and only 23% were classified as positive.

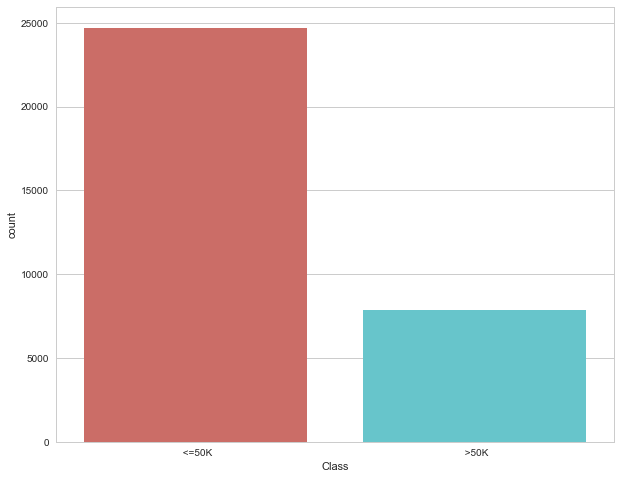


Figure Unbalanced Data

1. Feature correlation

The plot shows the correlation between the features and class variables.

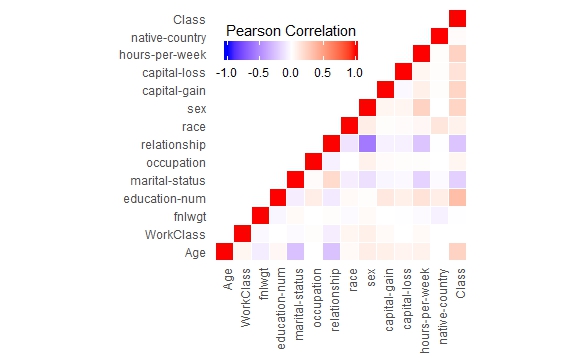


Figure Feature Correlation

1. ***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.

There are two major benefits of Ensemble models: better prediction results and a more stable model.

In order to increase our predictive power, we built and tested three ensemble classifier models. For each ensemble classifier, we picked a different odd-number combination of algorithms out of the set of classic supervised learning algorithms we learned about over the span of this course. Each combination 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 consists of *KNN, Logistic Regression and Random Forest*.

Since these models are highly correlated algorithms, we expected Ensemble A to perform comparatively worse than the other algorithms.

**Ensemble B**

Combining multiple predictions generated by different algorithms would normally give you better predictions. It is due to the diversification and combined nature as compared to each other, the model gives a better prediction. The key to creating a powerful ensemble model is diversity. An ensemble with two techniques that are very similar in nature performs poorly than a more diverse model.

So, in Ensemble B we combine *Random Forest, Naïve Bayes and KNN* to test the result. And due to their diverse nature Ensemble B performs better than ensemble A.

**Ensemble C**

Finally, we combine all the big five classifier algorithms to build the Ensemble C.

1. ***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

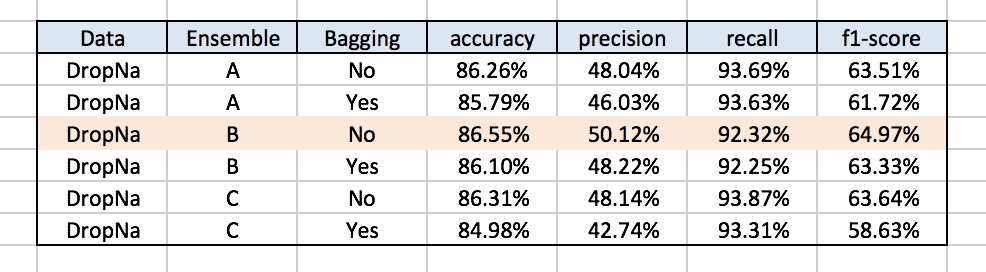


Table : Trial Run Results table

1. ***Data Pre-processing***
   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 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.

* 1. 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.

1. **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>

1. **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.

* 1. 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.

* 1. 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 shows what difference the normalization made in our datasets.

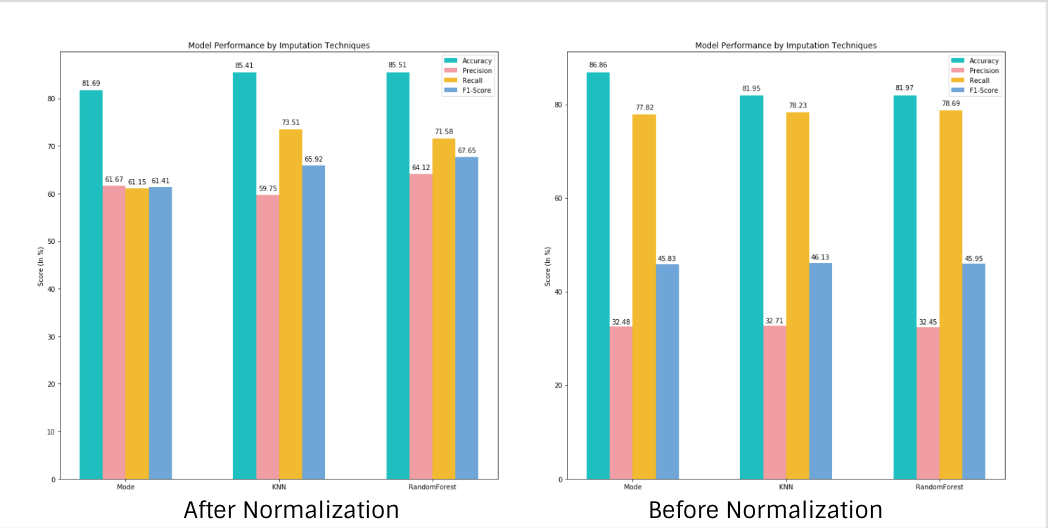


Figure : Results Before and After Normalization

* 1. 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 : Result table after Parameter Optimization

The results can be visualized in following plot

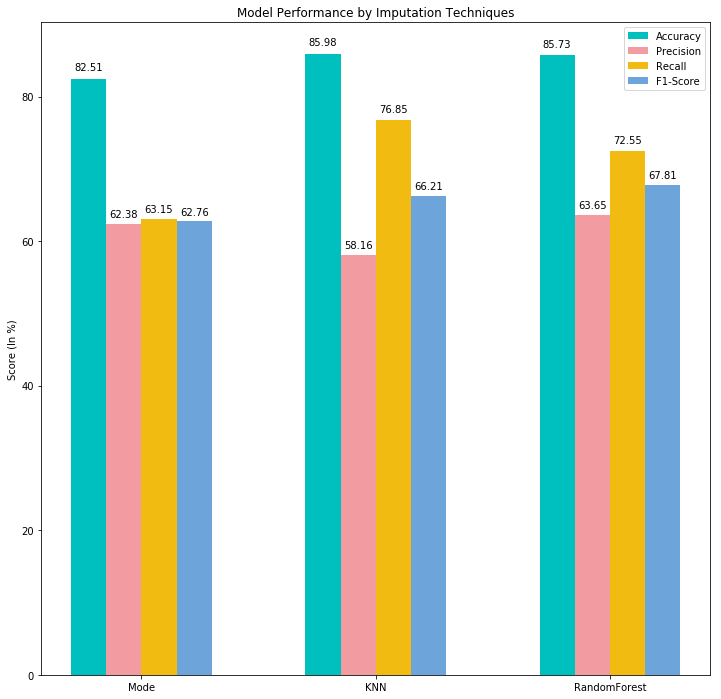


Figure : Plot of results after parameter optimization

* 1. 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 amount 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 : Result table after performing Feature Selection

The results can be visualized in following plot.



Figure : Plot of results after performing Feature Selection

1. ***Optimal Model***

We created three processed data sets with the imputation techniques mentioned above in [section 6.2](#imputation) which are Mode imputation, KNN imputation and Random Forest imputation.

We tested all three Ensemble models A, B and C (mentioned in [section 4](#Ensemble_Models)) on these three processed data sets. Table 5 shows the best pairing for of ensemble models with respect to each processed data set with imputation values.

Out of which we found out that the pairing of **Ensemble C and the Random Forest Dataset** is the Optimal Model as evaluated by the four performance measures – accuracy, precision, recall and F1-score. However, this determination comes with a caveat.

Model of the pair *Ensemble B and KNN imputation with feature selection* *of 60* came pretty close to the optimal model in terms of accuracy, actually yielding the best results for precision and F1-score. 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 F-1 score 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 were edible or poisonous, we would rather err on the side of false negatives than false positives, since the price of a wasted food is much lower than that of food poisoning and possible death.

Though Mode imputation was the simplest and least computationally expensive approach to imputation, it fell short in terms of performance compared to the other more complicated imputation methods. This claim is evidenced below by the results displayed on table 5.

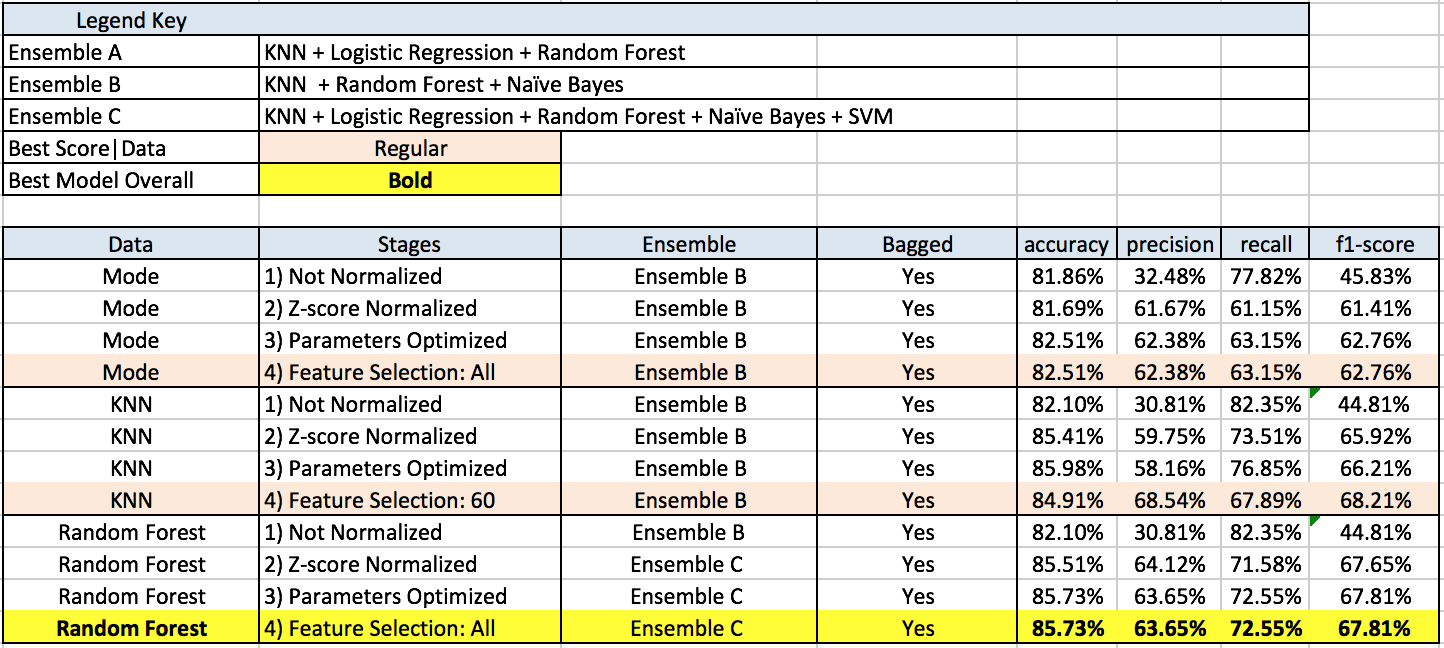


Table : The Final result table



Figure : Final Results Plot