STATISTICAL MACHINE LEARNING - FINAL PROJECT REPORT

ERAN AMAR† AND TOMER LEVY*

Weizmann Institue †*eran.amar@gmail.com* **tmrlvi@gmail.com*

Abstract

As final project for the course "Statistical Machine Learning" at HUJI, we attempted to tackle a simplified version of the 2009 Netflix Challenge. We reviewed various techniques to create the best model possible for predicting user ranks. We began with exploring the standard models individually, trying to build a non-trivial enhancement to these models. Later, we composed these models in order to create stronger predictor. The techniques used include k–Nearest–Neighbors, Random Forest and Elastic Net (with stronger emphasis on Ridge regression). The code of our project can be found in the following link: https://github.com/eranamar/netflix_challenge

Task Overview

The task is a simplified version of the Netflix Challenge from 2009. The dataset is a matrix of rating of 12,931 users for 99 movies. For each rating there is also the corresponding date in which it was given (as a count of days since some unspecified epoch). The datasets were already split into Train and Test sets (10,000 users in the Train set). In both datasets, the first 14 movies were ranked by all the users, whereas other movies might have missing ranks (i.e. not all the users ranked them). The goal is to predict the rating of users from the Test set for a specific unseen movie - *Miss Congeniality*. The rating for that move (which is the response variable) were given only for the training set, and we denote it as y.

For the rest of this paper, denote $n = 10,000$ the number of labeled examples, and $p = 99$ the number of features we have. The training data is a ranking matrix $X \in \{1, .., 5, NA\}^{n \times p}$ with the first 14 columns have no NA entries, and the response vector $\mathbf{y} \in \{1, .., 5\}^n$. The dates matrix $D \in \mathbb{N}^{n \times 100}$ where each entry $D_{i,j}$ have the date in which person i rated movie j. The dates are given as integers denoting the count of days since some epoch (unknown to us). Note that $D_{i,j} = NA$ if and only if $X_{i,j} = NA$. Note that we consider the dates corresponding for the movie of the response vector as part of the matrix D rather than an a separate vector.

The test datasets are similar to the training but for 2,931 unlabeled examples.

Our Approach

1. Before Getting Started

1.1. **Insights about the Data.** There are two main data matrices, the first corresponds to the rating of each person for each movie X, and the second matrix D contains the date in which each rate were given for each of the movies. In our models, we only consider the rating matrix because it contains most of the important information. We observed that the most common ratings are 3 and 4 (34% and 31% respectively) and the rarest ratings are 1 and 2 (with 1.6% and 5% resp.).

We tried to examine the correlation between dates and the corresponding *average rating* for any fixed movie. So first we fixed an arbitrary movie, then we divided the date range into discrete intervals (bins). We discretize the range in order to reduce the variance in the measurement. Inside each bin we calculated the average rating for the chosen movie, and estimated the confidence interval using the estimation of the standard deviation (assuming Gaussian distribution of the mean, by the *Central Limit Theorem*). We plotted for each bin its corresponding average and confidence interval to observe the change in the average rating over time (for that specific movie). After repeating for several movies, we conclude that the change in average rating over time is apparently small. The implies that the rating of a movie is relatively stable over time. The bin sizes that we considered were 7, 30 and 60 days.

See figure 1.1 and 1.2 for some examples.

All the exploration process described above ignores NA entries in the data. Since the given matrices are far from being complete, we have to be very careful in the way we treat missing entries in our data. We considered the following options (elaborate description will be given in section 2):

- Removing incomplete entries when fitting the model.
- Imputing the missing values with some fixed constant that was deduce from the data.
- Imputing the missing values gradually and iteratively (resulting with different imputed values).

1.2. **Modeling Decisions. Numerical variables**: We treat the rating values as a numerical variables rather than categorical. Note that in some models we are able to simulate one-hot encoding of the ratings by using proper

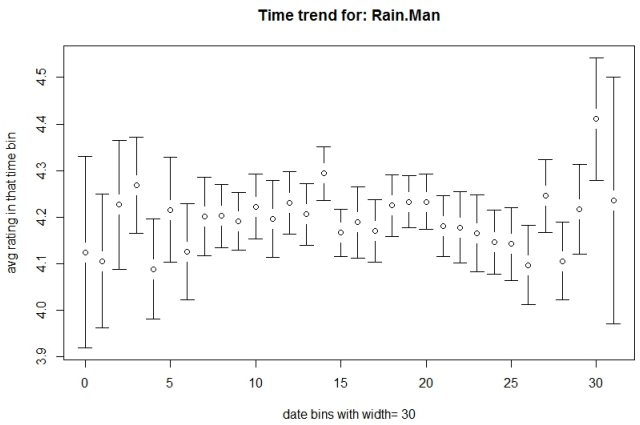

FIGURE 1.1. Average rating for the movie "Rain Man" by time slots of 30 days (with confidence intervals)

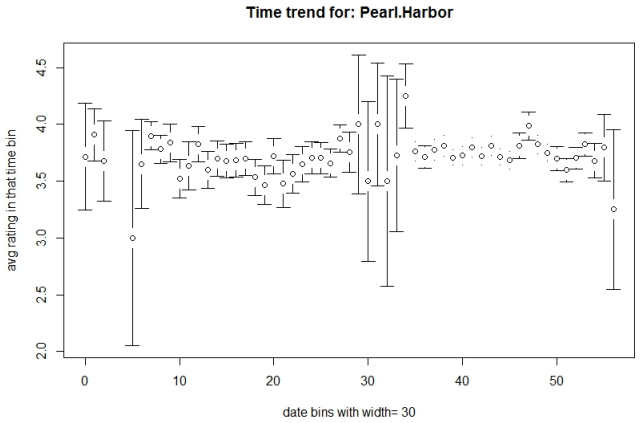

FIGURE 1.2. Average rating for the movie "Pearl Harbor" by time slots of 30 days (with confidence intervals)

kernel function. For instance, consider the KNN model, and suppose we want to use the ℓ_2 norm as the distance measure. We can design a distance function as follows: given two *scalar* rating x, y (rather than 1-hot) the distance will be

$$dist(x,y) := \sqrt{2} \cdot \mathbf{1}_{[x \neq y]}$$

and observed that if we used $\mathbf{x}, \mathbf{y} \in \{0,1\}^5$ as the 1-hot encoding of x, y, then $||\mathbf{x} - \mathbf{y}||_2 = dist(x,y)$. Although that is not a very compelling example, it demonstrates that with suitable distance function, there is no need to worry about data encoding.

Problem definition and loss function: Throughout this paper, we most of the time, consider the prediction task as a regression problem over the real numbers. In order to get valid predictions from each model in the range of $\{1,..,5\}$, we rounded each regressed value in the predicted response $\hat{\mathbf{y}}$ to the closest integer in $\{1,..,5\}$, then, we used the Root Mean Squared Error

$$RMSE(\mathbf{y}, \hat{\mathbf{y}}) := \sqrt{\sum_i (round(\hat{y}_i) - y_i)^2}$$

that gives us a score we used later to evaluate the model and compare it to other models.

2. Methods We Explored

2.1. ***k*–Nearest–Neighbors (KNN).** *KNN* regression is a simple model, in which each variable is predicted to have the mean value of the k "closest" (under some distance function) samples from the training set. That model assumes nothing regarding the data and its underline distribution, therefore has low bias (with the cost of high variance). When we come to try that model, we faced two decisions:

(1) Which distance function to use? In essence, even though the ranking is a number between 1 and 5, it could be that different ranking contribute differently to the distance of the samples (i.e., user rating 3 instead of 2 is more significant than 5 instead of 4).
(2) Imputations: how to compute distance above between incomplete vectors?

2.1.1. *Distance Functions.* As for the first question, the most standard choice is the ℓ_2 norm. However, from examining the data, most of the ranking are around $3, 4$, and rankings of 1 are relatively rare. Even though it is not mandatory, a proper distance function would be a neat way to represent this difference. We want to have non-linear measure which is proportional to the relative abundance of the relevant rankings.

A method to build such distance function could start with defined distances between consecutive rankings. Then, the distance between non-consecutive rankings will be the sum of the consecutive rankings. That is, assuming we have n rankings, if the distance between consecutive rankings is $w = (w_1, \ldots, w_{n-1})$, denoting the distance $d(i+1, i) = d(i, i+1) = w_i$, then the distance between i, j, assuming w.l.o.g that $i \leq j$, is

$$d(i,j) = \sum_{k=i}^{j-1} d(k, k+1)$$

By definition it is symmetric and $d(x,x) = 0$ iff $x = x$. As well, one can show that it also satisfies the triangle inequality.

Ultimately, we test the KNN model with two possible distance functions: the first is the one described above and it will be denoted as the "rarity distance", the second function is the ℓ_2 norm. We moved now to discuss the second decision.

2.1.2. *Imputations.* **Removing NAs**: The simplest technique to handle rows and columns with missing entries is by simply removing them from the dataset. This technique has two main advantages. The first is that the prediction of the model will rely only on real complete data, rather than a dataset with artificial values that were introduced by some (possible bad) imputation technique. The second is faster running time (since we shrink both dimensions of X).

The disadvantages are: increased variance in the model by relying on smaller dataset. Moreover, imputing may act as some kind of regularizer. For instance, imputing all the NA's with the most common rank in the matrix (the mode), can have stabilization effect on the prediction of the KNN, thus lowering the variance.

We implemented the KNN model that "ignores" missing observations as follow: Let $\mathbf{x} \in \{1, .., 5, NA\}^{99}$ be a vector of ranking of some person for which we want to predict his\her rank y for the unseen movie *Miss Congeniality*. First, we removed from $\mathbf{x}$ all the entries containing NA, denote the resulting vector as $\mathbf{x}'$. Then we removed from the training matrix X the same columns corresponds to the entries removed from $\mathbf{x}$. Next, we looked only on rows of the resulting matrix which don't have NA in it. That is, we consider some sub-matrix X' of the original X such that any person left in X' rated all the movies that $\mathbf{x}$ has rated (maybe more movies, but we wont look on those values). Then, we find the k neighbors (rows) in X' that are closest to $\mathbf{x}'$ under a fixed distance function (we tried both functions from section 2.1.1). Each neighbor supply a "vote" for the value of y, we calculated the mean of the votes, and rounded if necessary, to get a prediction $\hat{y}$.

The implementation describe above, was applied to both: the original rating data, and to its centered version. The idea is that different persons might have different perception of what should be the rank for "average movie" (i.e. movie that is not too good nor too bad). The result appears in table 1

Using the mode of X: Another option to handle missing values, which is very common, is to impute the missing entries with some statistics of the rest of the data. Specifically, imputing a given column by the most common value across the entire matrix (mode). However, after imputing the missing entries in X we faced a great computational challenge, because for each prediction the algorithm will have to consider $10,000$ possible neighbors. In order to circumvent that, we reduce the *row* number of X by considering only sampled subset of the rows.

2.1.3. *Glass Ceiling?* In order to quantify the quality of each predictor, we divided the data into train and validation sets. We fit each model to the training set, and predicted on the validation set. The *RMSE* scores for each predictor (with rounded predictions) appear in the table 1.

Method	Centered By	RMSE
Bagging + C.V.	**None**	**0.860**
Bagging + C.V.	User	0.915
Bagging + C.V.	Movies	0.884
Ignoring NA's	None	0.892
Ignoring NA's	User	0.937
Ignoring NA's	Movie	0.894

TABLE 1. RMSE for different KNN methods coupled with centering techniques

Because we treated the problem as regression, the results we got were real numbers. We were interested to see the distribution of those values for each true rank class. We plotted the distributions jointly on one chart, and we realized that the separation between classes was worse than we expected. In figure 2.1, each curve represents the density of the real-valued predictions for a given rank (predicted using the ℓ_2 norm as distance). Note that our rounding function split between rank a and $a+1$ by a threshold cut on the value $a+0.5$. As you can see, there is a lot of overlaps between the different classes, especially among the most common classes 3 and 4. Figure 2.2 shows the density of predictions when we used the *rarity distance* instead of the ℓ_2 norm. Note that using the rarity distance cause to each curve to be more concentrate around its center, meaning that the standard deviation of the predictions per rank was smaller.

In both cases, it seems that we reach the glass ceiling for this algorithm, as separating between the predicted ranks via threshold-based rounding is apparently a technique that is destined to fail, and that we should either consider other distance functions or more sophisticated models.

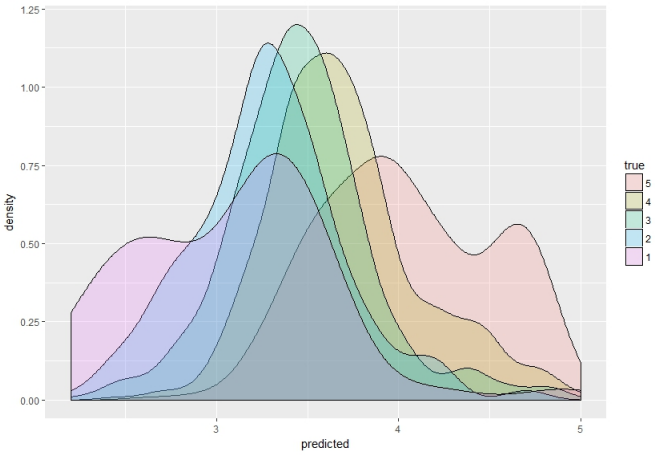

FIGURE 2.1. Prediction value density per rank class, over the ℓ_2 norm as the distance function.

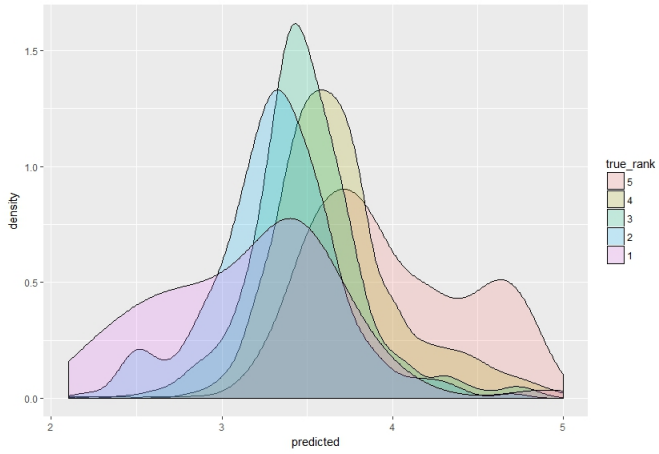

FIGURE 2.2. Prediction value density per rank class, over the *rarity distance*.

2.2. **Random Forest (RF).**

2.2.1. *Imputations.* In random forest, we are interested in the complete data. This is because, as oppose to *KNN*, the prediction in Random Forest (RF) depends each time on all the data. Therefore, we considered in this section more imputation techniques:

(1) Replacing NA's for each User by their average rating
(2) Replacing NA's for each Movie by its average rating
(3) Imputing using the mode of the matrix: 3
(4) *Iterative Imputations:* using additional RF instances to impute iteratively based solely on previous movies.

The implementation of the first three options was standard. However, we move to describe the rationalization and implementation of the last technique.

2.2.2. *Iterative Imputations.* Define the following notations: For any feature $i \in [p]$, denote by c_i the i-th column of X and let $X_{1:i}$ be the matrix composed only by the columns $c_1, .., c_i$. Recall that in our case, $c_1, .., c_{14}$ are all complete (no NA values).

The *iterative imputation* technique works as follow: for each $j = 15, ..., p$, fit a RF regressor on $X_{1:(j-1)}$, denote the regressor as $f_j : \{1, .., 5\}^{j-1} \mapsto \mathbb{R}$. If there is a missing entry k in c_j, impute it (in place) by $c_j^{(k)} = round\left(f_j\left(c_j^{(k)}\right)\right)$. Store f_j in memory and continue to next iteration.

Note that on each step j, the matrix $X_{1:(j-1)}$ has no NA values due to previous imputations.

After the *iterative imputation* process is done, we fit a final RF classifier on the "complete" data X, and wrapped it with a rounding function to make sure that prediction values will always be valid ranking. Denote the final classifier by $g : \mathbb{R}^p \mapsto \{1, .., 5\}$.

How to predict with iterative imputations? Given an unseen test sample with possibly missing values $(x_1, .., x_p) = \mathbf{x} \in \{1, .., 5, NA\}^p$, impute (if needed) the coordinate x_j with the value $round\left(f_j\left(x_1, .., x_{j-1}\right)\right)$. Let $\mathbf{x}'$ be the "complete" version of $\mathbf{x}$, predict the value $g\left(\mathbf{x}'\right)$.

2.2.3. *Classification Trees vs. Regression Trees.* We split the data matrix X into train and validation sets, and compute the *RMSE* on the validation set. First we examined whether *iterative imputation* works better with classification trees or with regression trees. That is, either $\{f_j\}_{j=15}^p$ are functions into $\mathbb{R}$ or functions into $\{1, .., 5\}$.

Note that when using discrete range $\{1, .., 5\}$, we can supply the classification algorithm with *class weight* parameter (vector in $\mathbb{R}^5$). Table 2 compares the best parameters for *iterative imputing* together with which prediction range was used. The *class weight* parameter was computes as follow, given the response vector of the training set $\mathbf{y}$ for each $i = 1, .., 5$

$$frequency_i = \frac{|\{j : y_j = i\}|}{length\left(\mathbf{y}\right)}$$

In table 3 we compare the standard imputation techniques (for #Tree=150, Leaf size=5) for fixed prediction domain $\mathbb{R}$. As one can see, imputing with the mode is the best technique among all the options.

Prediction Type	#Trees	Leaf size	*RMSE* on validation
Regression	10	20	0.833
Classification	10	20	0.903
Regression	**10/100**	**10/5**	**0.823**
Classification	10/100	10/5	0.858

TABLE 2. Prediction type for iterative imputation, class weight only for the predictor $= frequency$

Impute	*RMSE* on validation
Mode	**0.817**
Movie Avg	0.822
User Avg	0.827

TABLE 3. Comparison of standard imputations for Regression

2.2.4. *Bootstrapping with Equal Proportions (Upsampled).* We wanted to check an idea inspired by the idea behind Boosting: we know that a small fraction of the data has rank of 1 (around 2% of the entire matrix) whereas there are plenty of examples for movies with rank 3 (third of the matrix). In particular, the algorithm will see only small number of examples with corresponding label equals 1. Therefore it might be that the algorithm

will generalize better if it will see the same number of examples for the rare labels as for the common labels. Formally, we construct 5 bootstrap samples $B_1, .., B_5$ of the same length t, where each B_i is a bootstrap generated from examples which ranked *Miss Congeniality* with the rank i. Then we construct a training matrix X from all the examples in $B_1 \cup .. \cup B_5$. We called that process *upsampling* but note that for some classes we actually *downsampled*.

Note that X have equal proportions of examples for each response value $i \in \{1, .., 5\}$ and that X might need to be imputed. So we using the mode then fit a *RF* regressor (wrapped with rounding function). The *RMSE* performance of this algorithm appear in table 4.

In the table we also consider the effect of different class weights for classification with and without *upsampling*.

Regarding regression, *upsampling* isn't a natural idea as it is with classification so we chose to exclude it.

It turned out that Bootstrapping with equal proportions *did not improved the result neither when we fit the model as Classification nor as Regression, regardless of the imputation technique we used.*

Class weight	Prediction Type	*RMSE*
Irrelevant	**Regression**	**0.817**
$frequency$	Upsampled Classification	0.875
	Classification	0.850
$1 - frequency$	Upsampled Classification	0.906
	Classification	0.880
$1/(1 - frequency)$	Upsampled Classification	0.877
	Classification	0.852
$1/frequency$	Upsampled Classification	0.915
	Classification	0.892

TABLE 4. Comparison of Class weights: #Trees=150, Leaf size=5, impute with the mode

2.3. **Linear Regression.** Since each value in the response vector $\mathbf{y}$ corresponds with a row in X, it seems only natural to assume linear relation between the columns of X and $\mathbf{y}$. However, linear regression does not know how to handle missing values on its own, so as before, we needed to handle the imputation of the missing values.

As done in subsection 2.2, we tried same imputation techniques, and additionally, tried to impute by movie average based on user clustering which will be described in the upcoming section.

The average *RMSE* scores over 100 trials (with fresh random train\test split for each) are presented in Table 5.

When we tried to apply regularization (either with ℓ_1 or ℓ_2 norm), the results were generally worse. This seems to imply that the linear assumption (i.e. $X\beta = \mathbf{y}$) does not apply.

Imputation Type	*RMSE* on validation
Mode	0.819
Movie Avg	0.822
User Avg	0.824
Iterative	**0.816**
User Clustering (w. Mode)	0.820

TABLE 5. Linear regression RMSE scores for different imputation methods

2.3.1. *Imputation By Clustering.* Seeing that the method of imputation greatly impacts the score of the predictor, we were motivated to improve the imputations further. One such attempt was trying to impute iteratively. Another approach we tried was to group users into clusters, and impute by the mode or mean of each movie in the given cluster.

Formally, we calculated the cosine similarity for each pair of users (defined by the following formula when ignoring NA entries)

$$s(i_1, i_2) = \frac{\langle \mathbf{x}_{i_1}, \mathbf{x}_{i_2} \rangle}{\|\mathbf{x}_{i_1}\| \cdot \|\mathbf{x}_{i_2}\|}$$

Then we apply *agglomerative clustering* algorithm to group the users using the distance measure $dist(i_1, i_2) = 1/s(i_1, i_2)$. Denote the clustering function $U : [n] \to [11]$ (we used 11 clusters). Given an entry $x_{i,j} \in c_j$ which have NA in it, impute it by

$$\hat{x}_{i,j} = mode(\{x_{k,j} \mid U(k) = U(i)\})$$

See the results of that technique in table 5.

3. The Final Predictor

We try compositions of few of the models above. On contrary to what we did in section 2.3, whenever we say in this section *linear model* we mean Ridge regression with lambda that was chosen to minimize the loss by cross validation (on validation set). We apply the rounding function on any predicted value to make sure it is a valid ranking. Below is the description of the most two promising models.

3.1. Models Description.

3.1.1. *First Model.* The first model is a sequential composition of linear model and RF as follows: given X we impute it iteratively with the linear model, then fit RF regressor on the resulting matrix X'. The RF uses 100 trees with node size of 5. We then compute the *RMSE* on a test set. The average *RMSE* over 10 trials was 0.821. We examined how switching the imputation and prediction algorithms affects the scores: We imputed with RF (20 trees, node size of 15) then predict with linear model and the average score was 0.813. We refer to last combination as our *first proposed model*.

3.1.2. *Second Model.* The *second proposed model*: we first impute the matrix iteratively using RF (20 trees, node size of 15) to get X', then learned a vector of coefficients β using linear model. In order to make a prediction on new vector $\mathbf{x}$, we apply *KNN* on subset of the rows of X', then take a weighted majority vote of the k nearest neighbors of $\mathbf{x}$, using $w(\beta)$ as the weights for some function w. Note that the optimal k parameter was chosen via cross validation (on different validation set). Note that the equivalent to apply bootstrapping and "weighted" feature selection.

We tried several ideas for the weight function w. In all cases we normalized the weights before using them with the *KNN*. The candidates appears below:

- First we try to use $w(\beta) = abs(\beta)$ because we wanted to attribute more significance to some movies such that, movies which were important in the linear model also were important in computing the distance for the *KNN*.
- Then we try to $w(\beta) = \beta^2$ in order to emphasize even more the importance of some movies and lowering the importance of intermediate movies.
- Last trial was with $w(\beta)_i = abs(\beta_i) \cdot \mathbf{1}[\beta_i > 0.03]$ where this threshold was chosen empirically to take about 20% of the coordinates. This was to imitate straightforward feature selection.

  We tried each function with different α parameter for the Elastic net, meaning that we check each function with either Ridge\Lasso Regression and a combination of both ($\alpha = 0.5$). Result appear in table

α	$abs(\beta)$	β^2	$abs(\beta_i) \cdot \mathbf{1}[\beta_i > 0.03]$
0	0.847	0.864	1.039
0.5	0.836	0.824	1.139
1	0.821	0.827	0.934

TABLE 6. Comparing weights function with types of Elastic net regularization

To our surprise simple Linear regression works very good for this problem without the need of special tweaking and tricks. Adding iterative imputation improves slightly the result but less than we expected.

Among the proposed models, the first one achieved the best *RMSE* with value of 0.813 which is the best score achieved in our project.

It left to further study

- Explore different weight functions for the second proposed model
- Find a way to utilize the dates matrix D in the predictors. One suggestion to extract meaningful information from the dates is to look at their "centered" values with relation to the date in which the movie was released. That is, for a each entry $x_{i,j} \in X$, have $d_{i,j}$ which is the count of days since the release of movie j until user i ranked it. However, in the current settings we only have the year of release for each movie, and also the dates are anatomized and not anchored to any known epoch.

The code of our project can be found in the following link: https://github.com/eranamar/netflix_challenge

Conclusion

In this project we explored various models and ways to combine them. The problem in hand is characterized by a lot of missing entries, therefore we invested a prominent amount of the project time in order to improve the naive imputation techniques. We came up with 2 special ideas, the first is the *iterative imputation* technique and the other is *cluster-based imputation*.

Among all the imputations methods we tried, there is no one methods with is better across all models. That is, in the Regression Trees the best method was the global mode, and in Linear Regression the *iterative imputation* worked best.