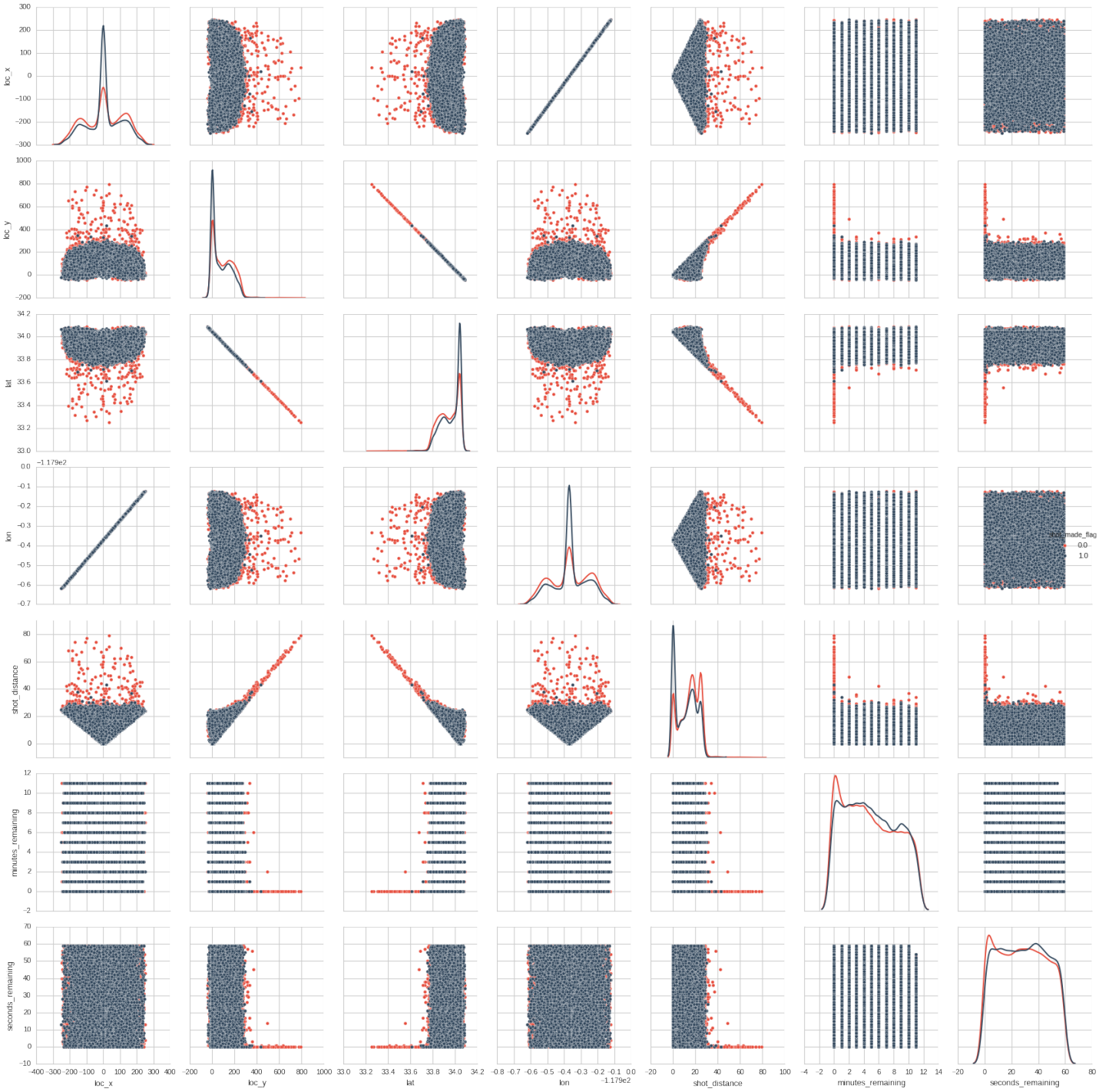
Kobe Bryant Shot Selection Solution.

Priyankar Talukdar : priyankartalukdar@gmail.com

Data Exploration :

The data-set provided need to be understood. In order for feature engineering, exploratory feature analysis is required. The following observations are made for the image hereunder:

- loc_y vs min_rem/seconds_rem : Kobe is more likely to take riskier shots at last minutes from deep which he almost missed all the time. The frequency of such attemps are exclusive to last minutes and second towards end of the game. This is corroborated observing shot_distance vs loc_x/loc_y and to an entext lat and lon plots.
- The shot distance vs success of shots has clean classification.
-



Categorical variables are crucial to modeling the training set. Here the probale categorical valriables are evaluated.

- game_event and game_id shows promise to categorize
- action_type : The frequncy of Jump Shot, though high, also suffers from high failure mark. The same can be said for layup Shot.
- combined_shot_type: jump shot (high failure), dunk and Layup seems to dominate
- periods, playoff and season shows promise with modified categorical modeling

The categorical variable post DE analysis can be seen to contribute quite substantialy to our categorical modeling framework. It is difficult to lay out all the modeling decisions here. The code section under data processing module is verbose.

The fatures arising out of categorizing **game_date, matchup, game_id and game_event_id** needs additional thought as they will bloat the dimension of the problem . A tradeoff mechanism is exploited in the code. Random Forest Technique of retaining relevant features, therey a tradeoff between dimensionalitu reduction and modeling accuracy. PCA was not used here.

A great tool t-SNE for visualizing high dimensional data might be evaluated in future to analyze better. The data exploration task can be less messy with pandas and sns /matplotlib. Note: t-SNE technique is notch up than PCA.

Data Preprocessing:

After design exploration step, the categorical and non categorical variables are separated and processed.
The following assumptions are made to model and pre-process data prior loading into predictive modeler.

- The box plot gives some numbers to start with. The **time remaining** is best split in quartiles for better granularity as categorical scope. Further analysis from prior (Fig1) showed the crucial factor of last seconds (appox 5-6 seconds) into the game, where Kobe is highly likely to take riskier shots that he will eventually miss most. The setup is made verbose in the code.

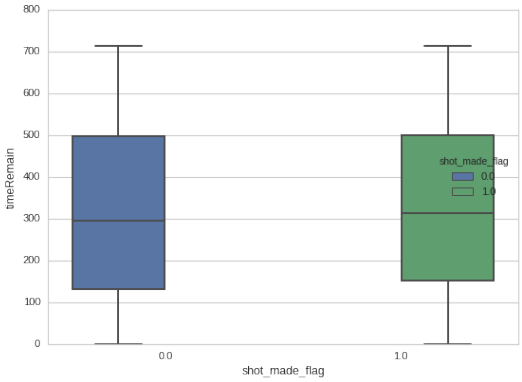


Figure 1: Analyzing the effect of minory contribution and unabalanced data-set. In this case, its farely comparable, therefore no need to for *scale_pos_weight*

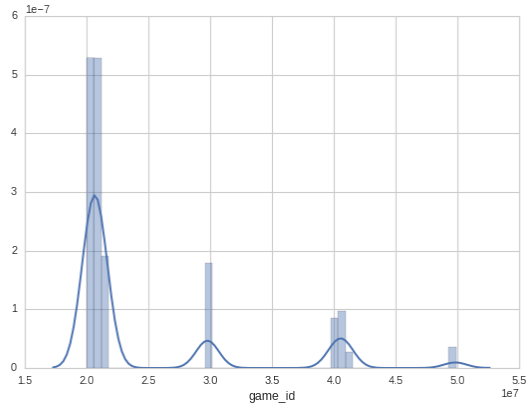
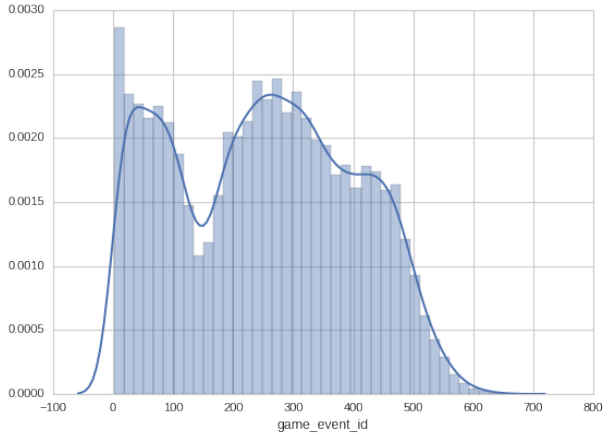


Figure 2: Analysis towards GMM models.



game_event_id.

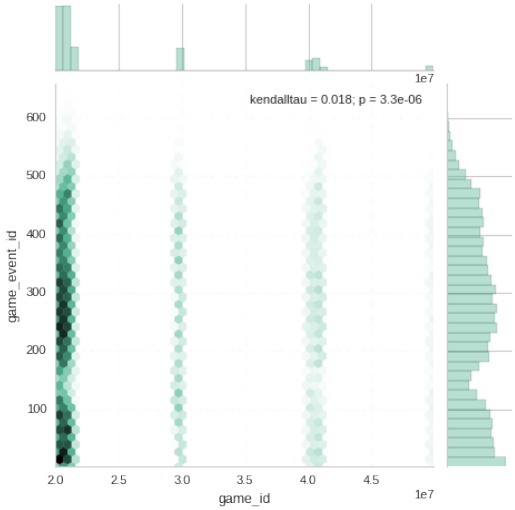


Fig 3,4 : GNM model conceptualization and correlation factors between game_id and

- Basketball games are played in four quarters of 10 or 12 minutes (NBA). Considering
- Upon observing, the frequency count of **game_id** , there seems to be four localized distinct localized areas. This makes it easier to do “Feature Selection” to reduce the categorical feature dimension conributed.
- However, the same cannot be said for **game_event_id**.

```
#  Categorising Estimate :
#  Note : Clear strategy is needed to model for categorical feature reduction >100
#             action_type :  57
#             combined_shot_type :  6
#             game_event_id :  620                    # expensive feature add
#             game_id :  1559                         # expensive feature add
#
#             lat :  457                              # moderate expense : GMM
#             loc_x :  489                            # moderate expense : GMM
#             loc_y :  457                            # moderate          : GMM
#             lon :  489                              # moderate          : GMM
#             minutes_remaining :  12
#             period :  7
#             playoffs :  2
#             season :  20
#             seconds_remaining :  60
#             shot_distance :  74
#             shot_made_flag :  2
#             shot_type :  2
#             shot_zone_area :  6
#             shot_zone_basic :  7
#             shot_zone_range :  5
#             team_id :  1
#             team_name :  1
#             matchup :  74
#             opponent :  33
#             action_type_num :  57
#             game_year :  21
```

data.py : Preprocessing step

Additional data exploration diagrams are attached with this file in the folder.

Feature Engineering

For feature engineering, the Gaussian Mixture model and One Hot Encoding is modeled individualy or combined to keep the feature dimension less, while at the same time cpature the important features of the data. Data exloration is strongly considered while such features have been engineered.

For example : game_id if modeled using dummy variables, might result in 1559 feature vectors(dummy var). Using a combination of data exploration insight along with Gaussian Mixture Model (GMM) and dummy variable insertion (One hot encoding), the game_id and game_event id has been reduced to 30 feature sets. **data.py**

Model Selection and Model Training

Although there are a host of binary classifier methods, gradient boosted trees, random forests and extra randomized trees stands out from Kaggle competition perspective. As furture design scpope, in order to design a strong ensemble a plethora of such methods should be used to design a stacking/binning based ensemble method.

From design simulations, it was found that xgboost would be the best design choice, if hyper parameter tuned correctly. It is extremely hyper parameter sensetive unlike randomforests. This makes it almost certian that a hyper parameter optimization doule has to assist the design if xgboost library needs to be used.

Bias and Variance consideration is encouraged in the design based on the link provied in xgboost webpage under *Control Overfitting* and *Handling Imbalanced Dataset*
http://xgboost.readthedocs.io/en/latest//how_to/param_tuning.html

Workflow Pipeline:

- **Modularized feature Transformation:**
  - Earlier discussion on GMM and one hot encoding design styles are modular in natire as they can be configured based on how granular we wish to seek mixture model clusters.
  - Future work would also include a validation of important feature vectors using randomforest
- **Automated Grid Search :**
  - HyperOpt tool is used as an optimizer to explore and optimize the hyper parameter of the design problem
  - in order to not overfit and falsely boost up expectation, a k-fold cross validator method is incoporated. This provides train/test indices to split data in train/test sets.
  - The best parameters recorded
- **Automated Ensemble Selection:**
  - Using K best models for training the ensemble as soon as we put another base model in the pool.
  - This work is still in nascent stage in the submission.

The intended final intened outcome is similar to the flowchart (from Kaggle database) . A voting along with geometric men of contributed models will be designed as future scope.

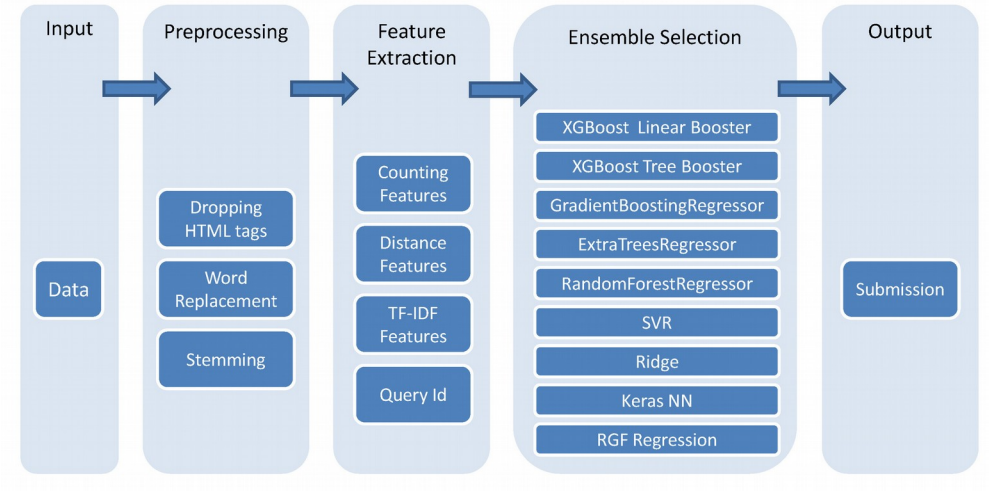


Figure 4: Ensemble Learning and workflow pipeline.

Report Summary

- The design is based around using gradient boosting exclusively.
  - This includes hyper paramter tuning and optimization
  - Boosted iterations to generate accurate outputs
  - Chosen as better bias vs variance report(kaggle) : Hyperparameter sensetive!
- Gaussian Mixture Model and One hot encoding is combined for feature modeling
  - The parameters are tunable, thereby making data engineering process partiallly automated.
  - Further work remains to automate feature engineering
- Hyperopt : tool used to estimate the hyper parameter tuning
  - Automated method , based on supervised learning method
  - The tool has issues with mixture models (mixed integer params). This messed up the max_depth param in xgboost iterations. In the code the max_depth is left to the default state which is 6.
  - Other tool like Spearmint/MOE/BayesOpt (python) may be evalauted.
- The hyper parameter optimization extraction is computationally expensive method. For my designs I have kept it fairly computable friendly by smart guessing the range of optimal paramter and keeping iteration counts low. This churns out hyper parameters under 2 hours each. Due to paucity of time and things that be done from grounup, I assume the hyper parameter generated aren't optimal and therefore needs detailed simulations and tweaking.
- Scores
  - Based on weak exploration of hyper parameter tuning, a osingle boost round. num_boost_round =1, the kaggle score barely made it beyond 0.693, which is just 50% accurate prediction.
    - Perhaps I need to round to %.2f as suggested in the submission sheet.
    - Perhaps my hyper param is not accurate (though its my hunch that the boost should result in logloss score of .6002 and should figure in <50 rank. I am not ruling out possibility of bug in submission csv file genration.
    - Another observation that validates my cocnern that the hyper parameter is not optimal – Upon increasing the num_boost_round = 10 and above, the predictiona ccuracy degrades. This hints at overfitting somehow.

- Future work remains to be done. A stonger ensemble with other methods like logistic regression and random forests might make the final geometric mean preeiction stronger..
- The code has some issues as the translation of prediction scores during runtime and final outcome is different. The iterational output during hyper parameter tuning is ample evidence that **logloss** score of .60002 is much achievable.
  - Bug fixing needed. This is my first time evaluation of xgboost and there are certian details that remins to be explored and understood.
  - The final translation towards submission might be erroneous.
- A stronger ensemble has been conceptualised to tackle the design problem. Further analysis and deepr insight to complex ensemble creation to hack into the top 50 rank. I will be tweakinga and improving the model in due course.

Futher Analysis

In order to validate the prior low rank and score, here are a few examples to drive home the point. Here the prior problems stated has been solved and will be deat step by step and explained. *GameGaussianMixtureModel : n_component = 15*

params = {'objective': 'multi:softprob', 'eval_metric': 'mlogloss', 'colsample_bytree': 0.7, 'min_child_weight': 9.0, 'subsample': 1.0, 'eta': 0.1, 'max_depth': 7, 'gamma': 1, 'num_class': 2, 'n_estimators': 580.0 }	params = {'objective': 'multi:softprob', 'eval_metric': 'mlogloss', 'colsample_bytree': 0.7, 'min_child_weight': 9.0, 'subsample': 1.0, 'eta': 0.1, 'max_depth': 7, 'gamma': 10, 'num_class': 2, 'n_estimators': 580.0 }
num_boost_round = 1500	num_boost_round = 1500
Score: **0.64732** Rank: **735**	Score: **0.6109** Rank: **540**

Table I

As it is clearly seen here that modifying the param has a dramatic effect on the outcome score. *I did fix a small bug in csv file generation*. These results were recorded post the bug fix.* A simple change of *gamma* parameter shows dramatic lowering of logloss value. Higher *gamma* values prevent over fitting by preventing the features with higher correlation to grow on same decision trees.
It would be impossible to interpolate in between the fine detals of parameter perturbation. As far as Kaggle is considered, the score centric nature of *good* solution makes it even more important to design a hyper parameter optimizer. Based on the output from the hyper parameter optimzer, the efficient param shall generate much better rank and score.

params = {'objective': 'multi:softprob', 'eval_metric': 'mlogloss', 'colsample_bytree': 0.7, 'min_child_weight': 9.0, 'subsample': 1.0, 'eta': 0.1, 'max_depth': 6, 'gamma': 10, 'num_class': 2, 'n_estimators': 720.0 }	params = {'objective': 'multi:softprob', 'eval_metric': 'mlogloss', 'colsample_bytree': 0.7, 'min_child_weight': 9.0, 'subsample': 1.0, 'eta': 0.1, 'max_depth': 6, 'gamma': 10, 'num_class': 2, 'n_estimators': 480.0 }
num_boost_round = 2000	num_boost_round = 3000
Score: **0.60993** Rank: **513**	Score: **0.60993** Rank: **513**

Table II

params = {'objective':'multi:softprob', 'eval_metric': 'mlogloss', 'colsample_bytree': 0.7, 'min_child_weight': 10.0, 'subsample': .95, 'eta': 0.08, 'max_depth': 6, 'gamma': 10.0, 'num_class': 2, 'n_estimators': 520.0 }	params = {'objective':'multi:softprob', 'eval_metric': 'mlogloss', 'colsample_bytree': 0.3, 'min_child_weight': 6.0, 'subsample': 1.0, 'eta': 0.2, 'max_depth': 6, 'gamma': 3.0, 'num_class': 2, 'n_estimators':480.0 }
num_boost_round = 1800	num_boost_round = 2000
Score: **0.60858** Rank **462**	Score: **0.60335** Rank **277**

Table III

params = {'objective':'multi:softprob', 'eval_metric': 'mlogloss', 'colsample_bytree': 0.3, 'min_child_weight': 3.5, 'subsample': 1.0, 'eta': 0.2, 'max_depth': 7, 'gamma': 3.1, 'num_class': 2, 'n_estimators':480.0 }	<reserved for param extraction based on hyper param Opt>
num_boost_round = 2000	num_boost_round = 2000
Score: **0.60170** Rank **(150-206)**	Score: Rank
GameGaussianMixtureModel : n_component = 25	GameGaussianMixtureModel : n_component = 25

Table IV

It would have been conveninet if compute server was allocated to churn data. Unfortunately there is limit to how much I can thrash my laptop. It often overheats. A large complex hyper parameter optimization code (tuner) might run for half a day on 8 cores and 1Gb ram. The boosted trees and csv file generation costs me 20 min on a 4 core machine. The delay is based on the num_boost_round value. For back of the envelope calculation - max_depth = 6 for 1950+ feature, a good estimate of num_boost_round >1500 approx is good enough to achive convergence. *However for obvious conservative reasons the value is set to 2000.*

As can be observed that num_boost_round beyond 2k does not contribute much, as has been discussed earlier. Therfore we fix the boost round count fixed at 2000. In order to imporve the rank further, hyper parameter optimization is the key.

- My prior assumption was that the predicted output was a confidence interval probability for a positive shot. Upon closer inspection, it turned out to be probablity of numclasses defined in multiclass classification. This correction resulted in starting rank of 750.
- Growing larger number of trees helped achieve convergence of the best score for a param setting as shown in Table II. Table III in comparsion with Table II and I shows how volatile the param setting can be.
- Table IV shows that expliting the automated feature engineering – gameGaussianMixtureModel : n_component = 25 , which is an increase of 10, yields better results. **data.py**
- Table IV lists down the best param found so far using ad-hoc and educated guesses. This manual method of tweaking and extraction is seriously discouraged though. The entire optimal param extraction must be automated as given in **tuner.py**

What lies Ahead?

"Most surprisingly, the winning teams report very minor improvements that ensembles bring over a single well-configured XGBoost." -KDD Cup Review [1]. Based on this observation, it makes sense to configure well the param[2]. Till now, it had been ad-hoc

- Tha majority of relevamt scores(who tried) hover around .61 to .60170. Just to drive home the point - a logloss score of .6010 has a rank of 75! This means a target improvemnt of .0007 needed on my part, to get into the top 10%.
- The improvement is improbable by guessing params. This brings the need of hyper optimization, **tuner.py**
- Once a single well configured param setting has been extracted (complete day simulation), minor feature improvement technique such as stacking will be carried out. The probable improvement from this step mightat most be .001. Therefore the final projected score is .6002 (<50 rank) as have been stated earlier.
- However all these robust analysis and simulation costs compute resource and time.
- Last stage feature analysis to improve rankings. However it will be hard to imrpove subtantially in top 5% scores.

It will be helpful if you may provide me access to some compute server to speed up my submission timeframe.
You may validate the claims in this report by executing predictor.py. The settings are made for **Rank 150-206** score.
Dump the file : **submitAdhoc.csv** to kaggle submitssion page to validate the claimed rank. The code took 25 min to execute on my 4 core i5 laptop.

$ python predictor.py

Final Thoughts and Conclusion

The design conclusions can be described as below :

- From Table IV, the constant tweaks to params, shows that ranks of 150 and lesser can be achieved. However manual tweaking is time consuming without robust exploration.
- A stacking method is implemented **main.py ,** whereby a single later stacking is expected to imporve upon the score of the previous classification predictions.
- The design exploration for roubust analysis and a good rank (top 5%) requires the entire code to operate at full capacity on a compute server for a day or two.
- Automated modeling of feature engineering is helpful as can be seen from Table IV analysis. Automating this into the code would have made the code base complex and therefore not attempted.
- As dicussed earlier ensemble methods might have slight improvement over a highly effective tuned XGBoost. Therefore our main attempt should be for hyper parameter optimization.

How to run the detailed simulation?

$ python main.py

Detailed simulation will explore all possibilities and create the best solution. Note: The simulation will take up all the cores in the system where you may run it. In case you wish to limit the cores, do modify the same in the params of tuner.py. In case there be any issue, do getr back to me with your system specs. I will duly revert back with the code configured to your specifications.

The code submitted has much more potential than being honored with 150 rank. Though there is little evidence as of now to claim the code can spew solution to be be in top 1% rank, my design intuition makes me believe it can. Feel free to ask me any details and suggest if any.

There is a Ipython notebook file added to the folder. It may help you play around with the matrices and params.

References:

The literature refernenced or studied during the design phase is attached in the included folder with this file. Larger ffiles were deleted.

[1] https://www.linkedin.com/pulse/present-future-kdd-cup-competition-outsiders-ron-bekkerman
[2] https://www.kaggle.com/c/santander-customer-satisfaction/forums/t/20662/overtuning-hyper-parameters-especially-re-xgboost?forumMessageId=118487