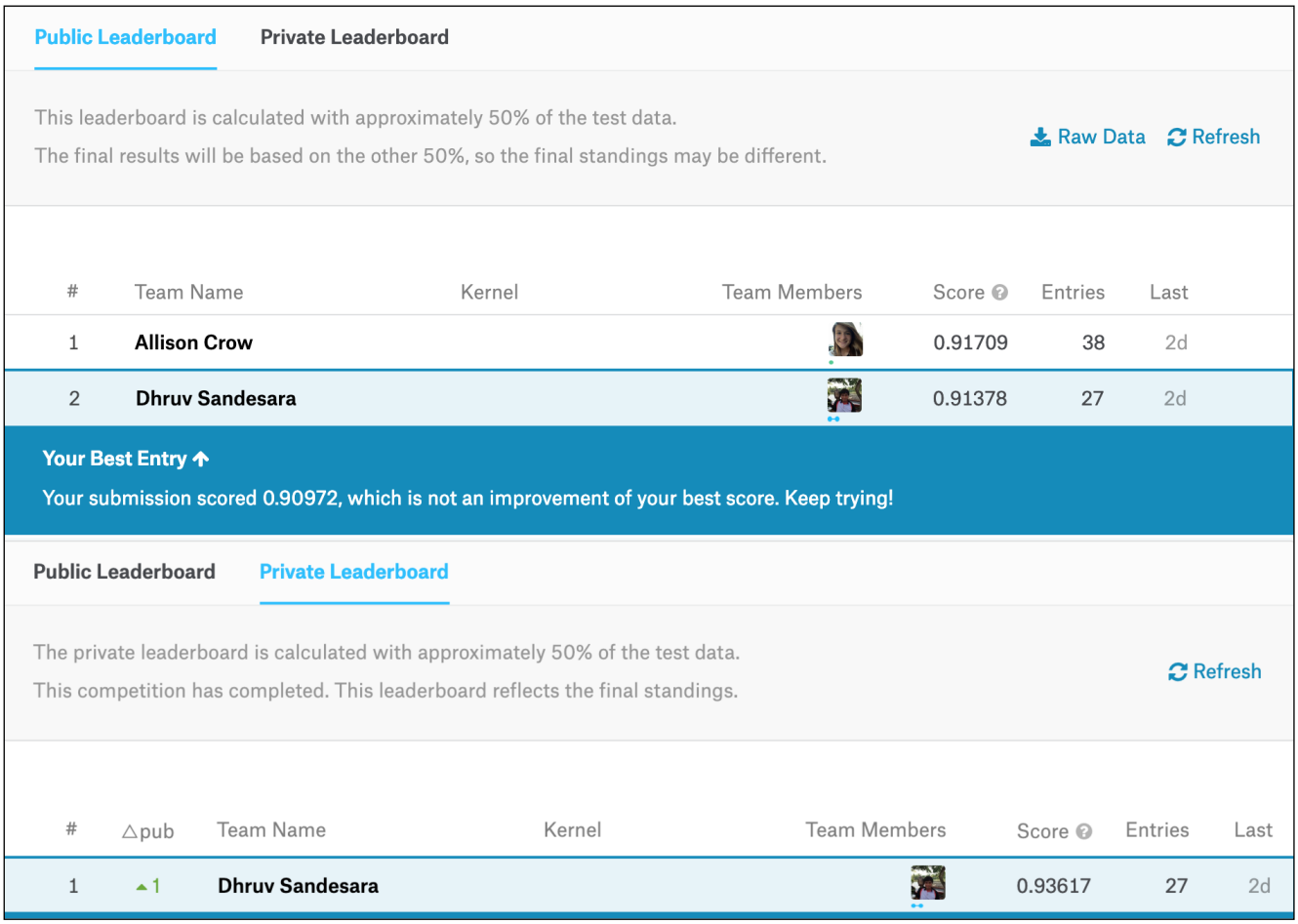
Kaggle Report

Spring 2019

Dhruv Sandesara (djs3967) - March 7, 2019



Public Leaderboard | Private Leaderboard

This leaderboard is calculated with approximately 50% of the test data.
The final results will be based on the other 50%, so the final standings may be different.

Raw Data | Refresh

#	Team Name	Kernel	Team Members	Score	Entries	Last
1	**Allison Crow**			0.91709	38	2d
2	**Dhruv Sandesara**			0.91378	27	2d

Your Best Entry ↑
Your submission scored 0.90972, which is not an improvement of your best score. Keep trying!

Public Leaderboard | Private Leaderboard

The private leaderboard is calculated with approximately 50% of the test data.
This competition has completed. This leaderboard reflects the final standings.

Refresh

#	△pub	Team Name	Kernel	Team Members	Score	Entries	Last
1	▲1	**Dhruv Sandesara**			0.93617	27	2d

Overall rank: **1st**
Public Leaderboard score: **0.91378**
Private Leaderboard score: **0.93617**

Model Used

The very first thing that I did to prepare for this competition was to be in class when the professors were dropping strong hints on what to use in the class and what the main objective was for this kaggle competition. Professor Dimakis said that logistic regression or linear regression will be basically useless compared to random forests or Xgboost. This will be as they just don't have enough nuance to accurately predict the output. He also said that Xgboost will give a better result compared to random forest. So I entirely focused my attention on using only the Xgboost model. Also by what the professor was saying, I gathered that the objective of this assignment was to get familiar with hyper parameter tuning and thus decided ton entirely focusing my efforts on that

Initial Sources

The first thing I did was to read the documentation associated with xgboost from this website: https://xgboost.readthedocs.io/en/latest/python/python_api.html. From this I got to know the main params I will be tuning in regard with xgboost and what exactly they change in a model. Namely these were: learning rate, n estimators, max depth, min child weight, max_delta_step, subsample, colsample_bytree, colsample_bylevel, reg_alpha, reg_lambda, scale_pos_weight, base_score and gamma. From this I separated which ones were regularization params so that I can tune them in the very end after figuring out which features I would like to keep and which ones to discard. These turned out to be gamma, reg_alpha and reg_lambda. To figure out how to do the tuning I mainly followed the first article that popped up on google from Analytic Vidya: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/. Which I discuss more in depth in the next section

Tuning

The way I did most of the tuning was by GridSearcCV. I would basically pass in a big range of values combinations that I though would be appropriate for our dataset and then see which values the grid search cv would find to be the best pair. Then I would basically narrow the range around best combination found and then retry GridSearchCV on the next decimal point for more accuracy. This was the high overview of the param tuning.

Tuning Missteps

The weekend that I didn't have any kaggle submissions was when I tried to run a very large combinations of hyper parameters for each param I was trying to tune. This was in conjunction with a cross validation of 7 which I later realized meant it would try close to 40,000 combinations on my lowly 2015 MacBook Pro. This in hind side was a very bad decision as I basically couldn't try to submit any submissions for 2 days after which I realized that it would roughly take ~12 days to finish running all combinations. At this point I had to begrudgingly stop the kernel and go back to fine tuning closely related params like depth and min child weight in unison and then moving on to tune other hyper parameters for the previous optimal pair. This method was slower but ultimately got the job done.

Ingenious package

One moment that I was really proud of was when I got really bored of constantly checking if my cell had finished running that I basically did a simple google search to find if there is some way that slack could notify me when the cell was done. Lo and behold I stumbled upon jupyter-slack which does exactly that. It took me 30 mins to setup and this was by far the most helpful thing I brought upon myself. Now I could leave my laptop running at home and have it notify me on my apple watch via slack when it was done running so I could head back home and redo the process all over again just with better params.

The two great Ahah moments

I also had 2 major breakthroughs which caused my scores to jump by a large amount. One was when I was tuning my max depth. I was trying all values between 0 and 20. And it gave me the best value was 8-12 ish with a min weight of 1. And I was sad as I had tried this combination for a long time and I was not having a score increase no matter what I tried. I went back to scouring the documentation page to find some hidden param that I could tune to make it all better but alas couldn't find any. THEN I saw, oh wait the min child weight is an int and HOLD ON!, ints don't start at 1 they start at 0. Let me try that And lo and behold now the optimal min weight was 0 and then I started tuning the depth from 0-12 and it hit 12. So I said ok lets try bigger values till 20 and it hit 20. I was like oh my god my data is overfitting, Thats not good. So I took a break and tried to reason through this and thats when it it hit me. I have 24 features which I have deemed important of course if it thinks that it is getting good value out of them it is going to keep rising. And thus I went back and tried with

values till 30. Thats when it leveled off at 25. I then ran K fold with theses new values and had a great jump in accuracy. I submitted and lo and behold I had jumped to a very high number on the leaderboard and I was satisfied.

Different Hardware things tried

I knew one of the big limitations I had was my hardware as I just had 2 cores and I wasn't even using that to its full capacity till I found n_jobs=-1. So tried to run my code on PaperSpace with 16 cores to cut down on my time usage. I tried a lot but couldn't figure out how to get these packages onto that machine and thus had to abandon the idea in the end. I did try running on the kaggle kernels but the UI just didn't suit me after I was so accustomed to using anaconda.

Different Software things tried

I didn't try to use any other models apart from Xgboost as it was working pretty well for me. What I did try to do was trying to eliminate useless features. I tried this multiple times by determining their importance. Even though they game me a relatively low importance discarding them always made the model worse maybe because they were deciding factors in very rare circumstances or that the regularization that affected them through gamma, reg_alpha, and reg_lambda just found an optimal combination with them to give the best results.

Finally Selecting the submission

The final thing that I had to grapple with was to decide submissions I wanted to count for the private leaderboard. As my model had reached a higher accuracy I needed to be more sure of the hyper parameters. Thus I had decreased my learning rate and increased n_estimators for my GridSearchCV which made it take longer and longer time. Also After I would get my good params, I would try to find what was the best lowest n_estimators which didn't give an marked improvement on accuracy by the use of early_stopping_rounds. In the end I also bumped up this number to be more sure that this accuracy was the best we could get. This all meant that in the final days validation took a long time. Thus some times when I would have submissions left over at the end of the day I would just try out random hyper

params or new random seeds. Most ties this would fail miserably but some times it would just magically increase the score. This kept me vulnerable to overfitting to the public leaderboard. Thus I had to keep a track of these submission. Thus when the final 5 submissions were available I used 3 of them in this same manner and lo and behold one of them caused a marked increase in the score. Thus when It was time to select the final entires I made sure to include 2 high scores of the public leaderboard and one where I verifiably had a high accuracy rate in my K fold accuracy. But for some reason kaggle only took 2 of my submissions and thankfully one of them was that verified high accuracy score which got me first overall.