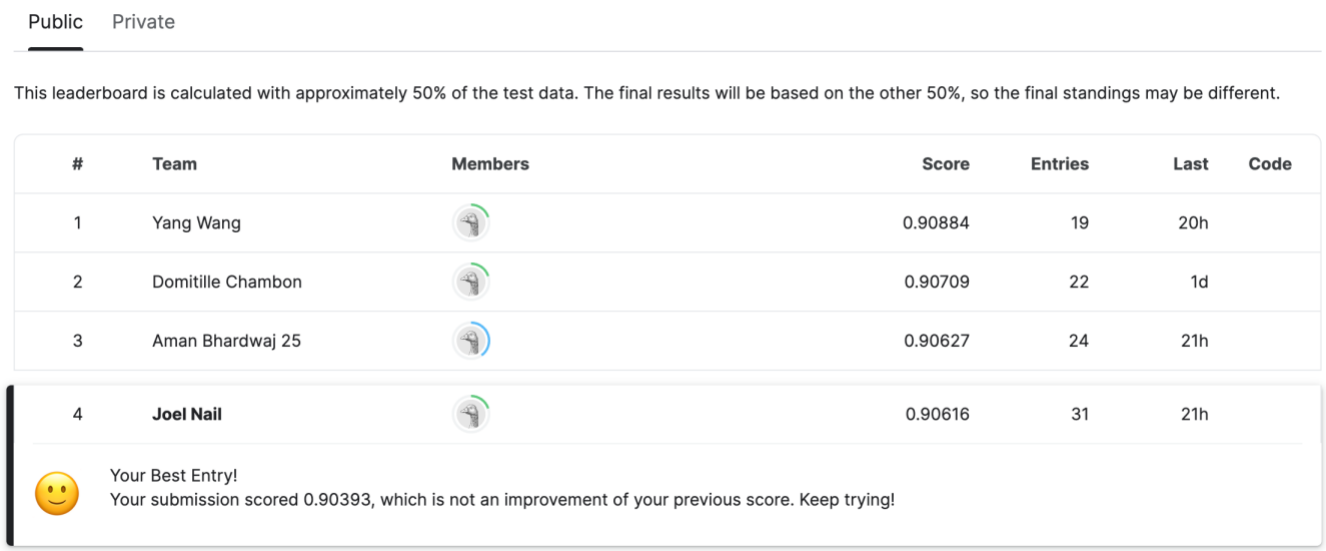
Joel Nail
BDS
Kaggle Report

Public Leaderboard: (4th)



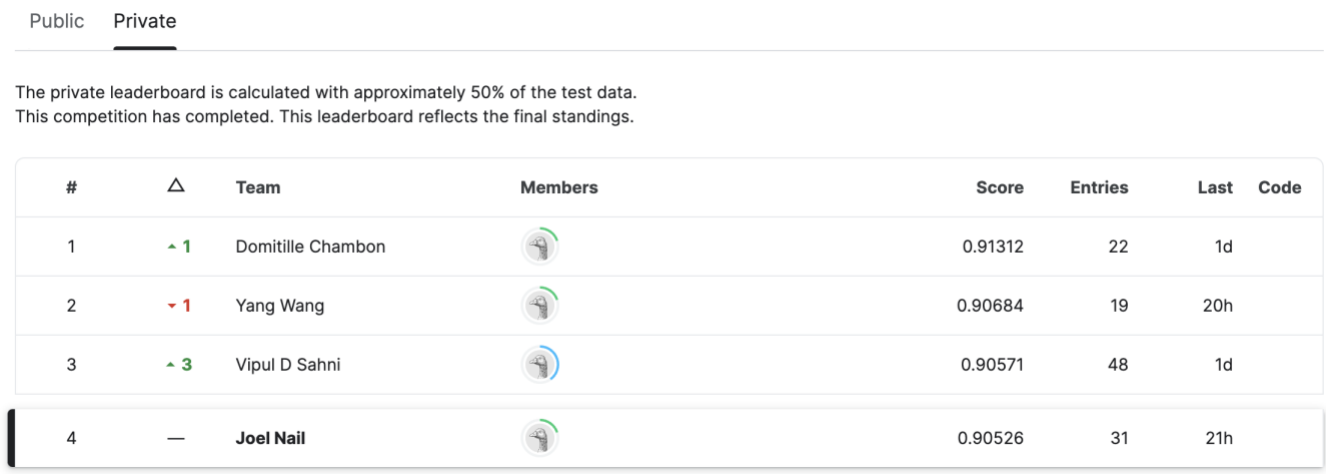
Public Private

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.

#	Team	Members	Score	Entries	Last	Code
1	Yang Wang		0.90884	19	20h	
2	Domitille Chambon		0.90709	22	1d	
3	Aman Bhardwaj 25		0.90627	24	21h	
4	**Joel Nail**		0.90616	31	21h	

Your Best Entry!
Your submission scored 0.90393, which is not an improvement of your previous score. Keep trying!

Private Leaderboard: (4th)



Public Private

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

#	△	Team	Members	Score	Entries	Last	Code
1	▲ 1	Domitille Chambon		0.91312	22	1d	
2	▼ 1	Yang Wang		0.90684	19	20h	
3	▲ 3	Vipul D Sahni		0.90571	48	1d	
4	—	**Joel Nail**		0.90526	31	21h	

What I Tried

- <u>What Worked</u>
  - **XGBoost** – The first advanced method that I tried was XGBoost since I was familiar with it from prior homework assignments. XGBoost gave me reasonably good scores (.863) right out of the box i.e., without hyperparameter tuning. I used Scikit Learn's grid search to tune XGBoost's hyperparameters, but the grid search took a long time to complete. As such, I ended up moving on to CatBoost and left XGBoost for future exploration.

- **CatBoost** – I had heard good things from my classmates about CatBoost, so I decided to try it out. I was pleasantly surprised when my non-tuned CatBoost model scored a good bit higher (.873) than my non-tuned XGB model. After my initial submission, I used Scikit Learn's randomized search to find better hyperparameters for my model; I feel as though I was lucky to discover a good set of hyperparameters early-on which allowed my model to score much higher than before (.895). I spent the next few days playing around with different CatBoost hyperparameters, but I was only able to improve my score marginally (from .895 to .898). That was, until I realized that I was training my final models on only 70% of the dataset (the X_train from my train/test split). After changing my code to train the model on the entire training set, I was able to cross the 90% mark with a score of .903. After playing around with stacking and neural networks, I went back to CatBoost and continued tuning parameters; this time, rather than using randomized search, I manually tuned my model one parameter at a time while referencing the CatBoost Common Parameters documentation. Through this manual tuning process, I was able to discover hyperparameters that allowed me to achieve a score of .906 on the public leaderboard (my highest public score). I tried many things to improve my score, but CatBoost was by far the most effective approach that I came across during the competition.

- **Stacking** – I explored stacking during the competition, but I was unable to produce a model that outperformed my solo CatBoost model. I think this is because I was not tuning my XGBoost models well enough before including them in the stack. I learned a lot about stacking from my classmates after the competition closed, so I decided to revisit stacking after the competition was over. I stacked XGB + XGBRF + CatBoost and used Logistic Regression as my final estimator. With this stacked model I was able to achieve a score of .907 on the private leaderboard (my highest private score). I wish I would have explored stacking more during the competition because it is certainly a powerful predictive tool.

- What Didn't Work

  - **Logistic Regression** – I began this competition by running a Logistic Regression model to get a baseline score to compare against my more advanced models. Even after tuning hyperparameters, Logistic Regression was giving very low scores, so I didn't even bother submitting them to Kaggle. Although logistic regression is very important and useful, more advanced algorithms make logistic regression look like child's play.

  - **Neural Network** – I used Tensorflow's Keras library to build an artificial neural network with two hidden layers. It took me a while to troubleshoot some issues that caused my model to max out its accuracy at ~75%. After doing a good amount of research into using Keras for tabular data, I was able to produce a neural network model that achieved 90%

accuracy while training very quickly. I believe this model was overfitting because after submitting to Kaggle, these predictions only scored ~87% AUC. Neural Networks did not produce good results for me, but I did enjoy playing around with them since I find them pretty neat. I think it would be possible to achieve a better score by applying deep learning, but it would require a lot of tuning on my end.

Biggest Challenges

- **Feature Engineering** – I honestly didn't know where to start with feature engineering. I tried using Scikit Learn's feature selection tools to reduce the number of features which did not provide better results. I also tried examining the variance and standard deviation of the different features, but I quickly became overwhelmed with the large number of features. As such, I decided to focus on hyperparameter tuning since that was showing me positive results. Had the competition been a bit longer, feature engineering would have been something interesting to explore.

- **Keeping Track of Everything** – I ended up with more than five different notebooks because I kept creating new ones to keep things organized (not a good idea). This led to some confusion as I tried to find code for models from days before but had no idea where they were. I would often forget to train my models on the full training set which led to much lower scores and also required me to resubmit my predictions i.e., wasting a submission. Looking back, I wish I would have set up an outline for my approach in order to keep things more orderly throughout the competition.

- **Computer Issues** – My laptop struggled to keep up whenever I was running greedy algorithms like grid search and randomized search with a large number of iterations. Several times my computer crashed while running these algorithms which wasted a lot of time. I ended up using Google Colab to run my grid and randomized searches since it seemed to handle those algorithms better.

Reflections

Sadly, I did not choose my best model for the private leaderboard as part of my final submission. Although this was unfortunate, it was a good learning experience. I submitted two very similar models that ended up with pretty good scores on the public leaderboard. I chose the one with the higher public leaderboard score as the one to include in my final submission; I didn't choose to include both because I assumed they would get similar private leaderboard results, and I also wanted to include some of my simpler models in my final submission. I did

this in case those simpler models ended up performing better on the private leaderboard, but that was not the case.

What I learned from this is that for different models, public leaderboard score is a good indicator of private leaderboard score. But for models that are the same except for slightly different hyperparameters, the public leaderboard scores being higher does not always mean the same will be true in the private leaderboard. I'm sure this is a generalization and won't hold true in every Kaggle competition, but I think it's a good rule of thumb that if a different model gets you a significantly better public score, it will most likely give a better private score as well. Small increases in public leaderboard scores are not necessarily indicative of a corresponding increase on the private leaderboard.

I did attempt to improve upon my score following the competition's close. I was able to improve my score to 0.90777 by stacking CatBoost, XGBoost, and XGBRF with Logistic Regression as my final estimator. I had to increase my number of CV folds from 5 to 10 in order to get an improved score. I also tried a CV of 20, but this produced only marginally better results. I wish I would've explored stacking more during the competition since it's definitely a powerful tool, but that's just another great lesson that I learned from this competition.