Learning to Rank with XGBoost Ranker and Optuna

IDENTIFIERS

- Samuel Macintyre
- S3888492

OVERVIEW

For this Assignment I have opted to use XGBoost's XGBRanker as the model for training and Optuna for further parameter tuning.

MODEL DESCRIPTION

XGBoost is a supervised learning algorithm utilising gradient boosted decision trees. The model works by creating an initial decision tree, it will then calculate the residuals and feed them into another tree. The average of the residuals in the leaf nodes in this second tree are then multiplied by the learning rate and added the initial residuals. This process is then repeated as with the residuals gradually decreasing until a maximum specified limit is reached or there is no significant reduction in residuals from additional runs.

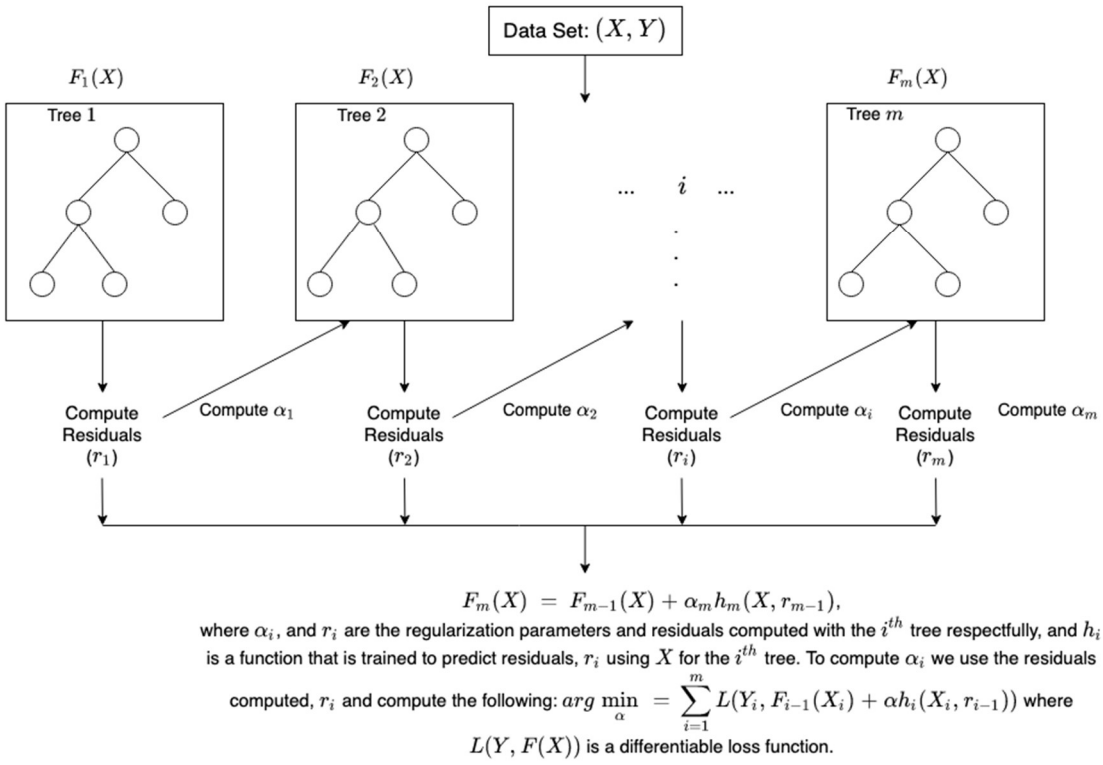


Figure 1. A Map of the XGBoost Algorithm (How XGBoost Works, 2021)

Optuna which was used for hyperparameter tuning is an automatic optimiser which creates a series of trials to identify the mix of parameters. It works by iterating through combinations of parameters to output whichever had the highest scoring metric, this is then run over multiple trials do look for a new optimum mix of parameters.

EXPERIMENTS

I utilised 3 experiments for this assignment.

Experiment 1: Model Train with 60/40 Split (Score: 0.6192)

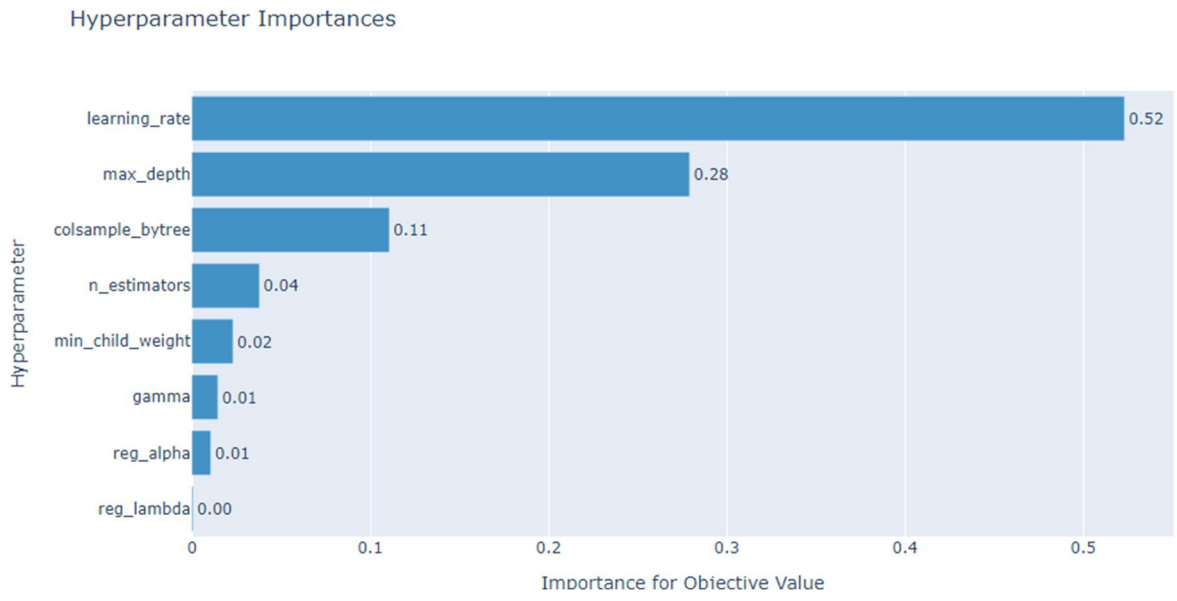
For the initial experiment I used a 60/40 training/test split. The data was split using sklearn.GroupShuffleSplit.

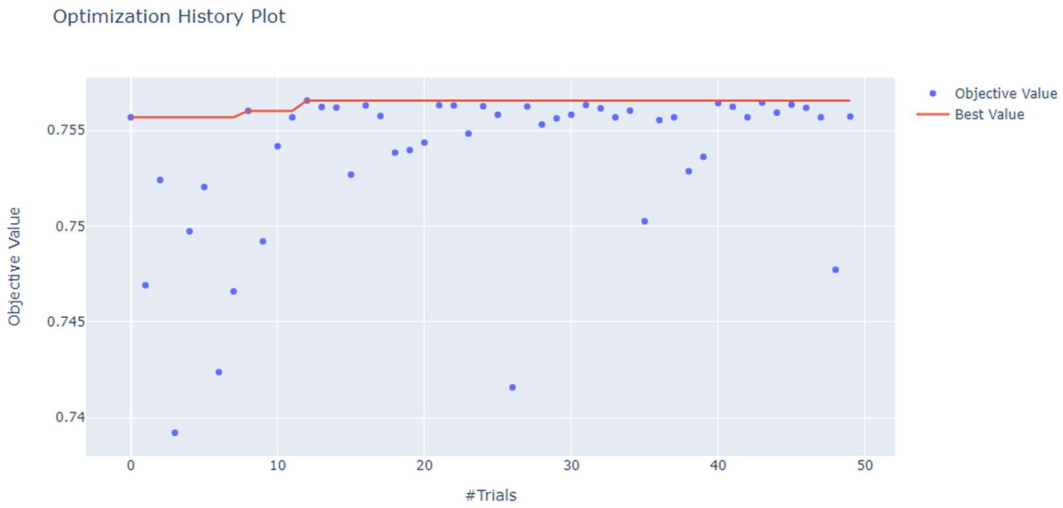
The parameters used were as follows { tree_method='hist', booster='gbtree', objective='rank:ndcg', random_state=42, learning_rate=0.1, colsample_bytree=0.9, eta=0.05, max_depth=6, n_estimators=110, subsample=0.75). As an initial run this was quite promising as it generated a score of 0.6192 when submitted.

Experiment 2: Model Train on full data with 5 Fold Cross Validation optimisation using Optuna (Score: 0.6038)

For the second model I utilised Optuna for hyperparameter tuning. Here I used 5 fold cross validation during the optimisation process, ran it over the full data set and scored it using sklearn.cross_val_score.

Additionally, in this experiment I used Optuna to identify hyperparameter importance to be used in a later run. For this experiment the XGBClassifier was used for parameter tuning as the default scoring could be used by sklearn.cross_val_score. This experiment ran 50 trials.





The parameters used were as follows {tree_method='hist', booster='gbtree', objective='rank:ndcg', random_state=42, learning_rate=0.013, colsample_bytree=0.46, max_depth=7, n_estimators=150,reg_alpha=5, reg_lambda=4, min_child_weight=3,gamma=4}. This yielded a score of 0.6038.

Experiment 2.5: Model Train with restricted parameters on full data. (Not Scored)

This experiment was the same as the above but using restricted parameters using only learning_rate, max_depth, colsample_bytree and n_estimators. The premise for this was that the model may be overfitted. However further research I opted to not score this run, but rather to continue with experiment 3.

Experiment 3: Model Train on full data with NDCG focused optimisation and XGBRanker(Score: 0.6032)

Experiment 3 utilised the full dataset for both optimisation and training, XGBRanker was used in the optimisation and the optimiser focused on maximising NDCG score using sklearn.ndcg_score. This assumed that the reduction in score may be due to the optimisation maximising the wrong scoring metric or that the optimiser for XGBClassifier was tuning the hyperparameters incorrectly. This however had a significantly longer run time with 1 trial taking 5hr 35m. As such only a singular trial was used. This yielded a score of 0.6032 even lower than before, suggesting that perhaps the issue is coming from overfitting and not from incorrect optimisation technique. Parameters used: {tree_method='auto', booster='gbtree', objective='rank:ndcg', eval_metric='ndcg', reg_alpha= 0, reg_lambda= 4, random_state=42, min_child_weight=0, gamma=1, learning_rate=0.11449, colsample_bytree=0.67, max_depth=21, n_estimators=740}.

Experiment 4: Manual Testing for Train Test Splits

For this experiment a series of the splits using the initial parameters and the optimised parameters were used to determine which splits would yield the best NDCG score and be least prone to overfitting.

```
   train/test %            NDCG

Split 99/01 (Optimised) : 0.647
Split 90/10 (Optimised) : 0.772
Split 60/40 (Optimised) : 0.797
Split 50/50 (Optimised) : 0.805
Split 30/70 (Optimised) : 0.789
Split 10/90 (Optimised) : 0.820
Split 01/99 (Optimised) : 0.826

Split 99/01 (Original)  : 0.657
Split 90/10 (Original)  : 0.772
Split 60/40 (Original)  : 0.796
Split 50/50 (Original)  : 0.806
Split 30/70 (Original)  : 0.815
Split 10/90 (Original)  : 0.824
Split 01/99 (Original)  : 0.825
```

Based on the concerns around overfitting I opted to use 50/50 (Original) as the final split/model.

LIBRARIES USED

Pandas, Numpy, xgboost, sklearn, Optuna.

REFERENCES

Amazon AWS. (2021) How XGBoost Works – Amazon SageMaker. https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html