Machine Learning: Final Project

– Prediction of User Churn for a Level-breaking Mobile Game

Abstract

The loss of new users becomes a serious problem for level-breaking mobile games. Our task is to predict players' churn or stay based on their game log within a certain time range.

The AUC scores mostly stabled at around 0.7911 on the test set with different models, such as Bagging(estimator amount: 50) with Decision Tree(Gini, Depth:3, Minimum Split:50) as the base classifier and MLP(Relu activation, iterate: 1000, hidden layer: (20,20)). A paired T-test is run on these final models with similar AUC scores, among which Bagging(estimator n=50) + LogisticRegression(solver: lbfgs, C: 1.0) has the most significant difference statistically.

Data used for learning are constructed from the game's raw record log. Two data sets are built for parameter tuning. One of which is weighted by the game level complexity, generated from the average pass rate of each game level. The best performances so far are all using the weighted data. Shapley values, cross validations and ablation study are used to analyze the feature explainability and importance. Under this task and with our weighted data model, ensemble algorithm and MLP works better than single classifiers. Most of the best performances from Ensemble and MLP converge to the same AUC. Therefore, an improvement could potentially be made if we polish our data set by introducing bias or adding in features that have more significant impact over the classification decision.

For extended experiments, we also tried re-construct data sets with less manual interference and other learning models. It is as expected that the training result drops with the data set that is closer to the raw data. The reason being in the original data construct, calculations such as weighted sum and arithmetic mean have smoothed the outliers and averaged out noises. These lower scores are a more true reflection of the raw data.

1 Introduction

Mobile games have a dominant position in today's daily entertainment. Various types of mobile games, especially level-breaking genres, have gained a very wide market as they allow users to utilize their fragmented time.

However, the loss of new users becomes a serious problem for this type of game. A significant proportion of users choose to drop the game after a short try. By interfering with users (such as with reward props) before they uninstall the game, it is possible to restore users and increase the activity of the game and the company's potential revenue. Therefore, predicting user churn has become an important and challenging task.

In this experiment, we will construct a reasonable model from unstructured log data to make such a prediction.

2 Programming Modules

Numpy and Pandas are used for data processing. Matplotlib and Seaborn are used for analyzing extracted features and training result visualization. We also invoked the Shapley module to analyze the impact of constructed features to the learning model output.

3 Data Overview

The data is from a casual level-breaking mobile game. The basic task of each level is to reach a certain goal within a limited number of steps. Users only enter the next level after completing the one before. They can use props or prompts to help during the level.

For most mobile games, users decide to quit often at the early stages of games, so retention in the second week after game release is a focus of the company. Our data selects the interaction log from 2.1-2.4 of all users registered on February 1, 2020. The data is filtered to ensure that these users have logged in on at least two days within the time range. And churn is when such a user does not log in in the next week(2.7-2.13).

Data here is in a form of raw data record, that is close to the form of log.

3.1 Train Set, Validation Set and Test Set

There are in total five data files.

These three sets come from files with similar formats and information. User IDs and associated labels are contained in the train set and the validation set. Labels are used for classification, 1 suggests player churn, and 0 means stay.

The train set contains 8158 users, 1/3 among which are identified as churn. It should be noted that in order to keep anonymity, data in this experiment have undergone certain non-uniform sampling processing. The churn rate here does not fully reflect the actual game situation. Users' IDs and levels' IDs have both been re-numbered, but should have no impact regarding our main task.

There are 2658 samples in the validation set. Its scope provides convenience for parameter sensitivity testing, ablation study and model choosing.

3.2 Level Sequence and User Log

This is the core data. It contains the records of each game level on a per-player/user base. One record is associated with one attempt at a game level.

The meaning of each column is as follows:

`user_id` : User's ID, so that data can match with the train/validation/test sets from above;
`level_id` : ID number of a level in game;
`f_success` : Whether a user has completed the level. 1 is success, and 0 is fail;
`f_duration` : Time spent(in second) on a level;
`f_reststep` : Ratio of remaining steps to limited steps amount; If fail to complete a level, the ration is 0;
`f_help` : Whether a user has used help props in attempt. 1 is used, and 0 is not used;
`time` : Time stamp; We could use this to identify whether a user has logged into the game.

3.3 Metadata for Game Levels

This contains statistical features and data for a game level. One thing that needs to be noted is that: meta information is not per-user oriented.

`f_avg_duration` : Average time spent to complete a level(in second, including all attemps);
`f_avg_passrate` : Average pass rate;
`f_avg_win_duration` : Average time spent to complete a level(in second, only on the attemp that passes the level);
`f_avg_retrytimes` : Average re-try/re-play times;
`level_id` : Level ID that can match with the sequence data in Section 1.4.

4 Data Pre-processing

We will use the information from the above section to construct the data set that could represent the users more. One difficulty is, as analyzed above, two data sets(Section 3.2 and 3.3) that documented core information are aiming at different targets, that is users and game itself, therefore we need to construct our final data model by building a connection between these sets. In short, we need to convert the game metadata to per-user information.

All data features that we could think of to represent a user will be calculated and constructed first. Then we will analyze all these features and decide whether to keep the feature or not.

4.1 Timestamp Related Data

We will focus on the timestamp data from the sequence collection. This portrayed users from a personal perspective, such as their addiction to the game and login information.

- $days$: Log in days amount; If a user keeps logging in for most of the days between 2.1-2.4, then it is possible that they will stay for daily registration or some other daily activities in the next week.
- $logins$: We will use a threshold between timestamps to determine whether a new login happens. This could be used to evaluate users' addiction to the game. A shortcoming is that this feature may not be accurate enough to represent players' addiction since they could also keep playing for a long time without re-logging in.
- $dailytime$: Portray users' addiction from another dimension, make up the deficiency described in "logins", that is long hours with less re-logins.

- $enddate$: The last day a user has logged into the game. If a user stops logging into the game at an early stage, then it is unlikely for him or her to come back for the game in the next week.

4.2 User Experience Related Data

When it comes to in-game and external features that can chase the users away, game complexity can have a huge impact. If a player keeps failing at a level, his or her impatience can build up, which can easily lead to a churn. The main task in this section is to build data that can interpret how difficult a user finds the game is.

Complexity can be analyzed from different dimensions:

1. Whether the game gives a user a good first-time experience, in general, that is if the first level(usually known as beginner's tutorial) is clear enough for him or her to understand[1]. We could measure it by the ratio of a user's time spent for passing the first level to the level's average duration.
2. Whether the game is overall complex enough:
    A. $maxlevel$: The max level a user has accomplished. If most of the churn users can only accomplish lower level tasks, then it is fair to say that the game may be too difficult to keep users interested.
    B. $maxtry$: Max retry times a user has tried for any level. This feature portrays how patient a user is, or how long they can keep interest when they keep failing on attempts to a level.
    C. $propuse$: Amount of times a user has used help props within the logged time range. The more props they used, the more unclear the game guidance can be or the more difficult the game is. The use of props could also be affected by the game recharge and reward mechanism. However, since we do not have any related data, we will identify this feature as a complexity-measurement for now.
    D. $remainstep$: We already have a ratio of remaining steps to limited steps in the sequence data. To connect it with the game metadata, we will do a ratio between the remaining step ratio and average remaining step ratio for each level. Data here will be logged on a per level base.
    E. $retry$: Ratio of retry times to average retry times; If retry times is zero, the ratio is 0; Potential risk is that when average retry times is too small and the ratio becomes too large; Therefore we might use the difference between two numbers instead. Every user has a retry ratio for each game level.
    F. $timespent$: Ratio of time a user spent on a level to the level's average duration (this is also a per level feature);
    G. $lastwinrate$: If a user keeps failing on one level or few levels, there is high potential that they will lose patience and quit the game; Therefore, we will take the last 20 records for one user and calculate the rate they accomplish a level within these attempts.
    H. $winrate$: Ratio of wins to the total number of attempts for a user. Same as G, some players can get more interested or addicted to the game as the challenge gets harder, whilst others can get impatient and quit the game.

There are in total 1509 levels logged in the metadata. Thus, there is a huge chance for us to encounter "the curse of dimensionality"[2] if we have three features: $remainstep$, $timespent$, $retry$, all at a per-level base.

An optimization is to get the arithmetic mean on the ratios. The mean value can be calculated with uniform weights, or with some other weights. To have the data more related to the game meta, we will derive the weight matrix from the average pass rates for game levels.

In fact, we will use the optimization method first, and switch back if the learning result is not ideal.

The image left is the code we use to generate weights from the average pass rates. The lower the pass rate is, the more difficult the level is for general users, and the heavier the weight will be when calculating the mean value. For example, if a user retries for 3 times on an early stage level(should be easier), and also 3 times for a more difficult level(lower average pass rate), the first 3 retries should be weighted more to emphasize that for this user, his game capability goes under the average.

```
def normalize(data):
    return (data - data.min()) / (data.max() - data.min())
```

```
def get_weights_from_passrate(data):
    with np.errstate(divide='ignore', invalid='ignore'):
        weights = np.true_divide(1,data)
        weights[weights == np.inf] = 0
        weights = np.nan_to_num(weights)
    return normalize(weights)
```

We will construct two data sets, one with these three features calculated unweighted, and one weighted. Simple classifiers will be run on them to analyze which one is more representative.

Another part to note is when we construct the max level a user has accomplished. Due to non-uniform sampling for anonymous, there can be missing information. That is, in level-breaking games, players can only play the next level after passing the level before it. When we extract accomplished levels, we could notice that the passed levels for a user can be not continuous. Under this case, we will generate a warning or error in the terminal. However, we will not exclude any of the levels, we will only identify as a missing sampling. Finally, for constructing user experience related data, we will set missing level features to an extreme condition, for example, set remaining steps to 0.

```
Missing info for user 2; Logged passed levels are not continuous!
Missing info for user 3; Logged passed levels are not continuous!
Missing info for user 4; Logged passed levels are not continuous!
Missing info for user 5; Logged passed levels are not continuous!
Missing info for user 6; Logged passed levels are not continuous!
Missing info for user 8; Logged passed levels are not continuous!
Missing info for user 9; Logged passed levels are not continuous!
Missing info for user 10; Logged passed levels are not continuous!
Missing info for user 11; Logged passed levels are not continuous!
Missing info for user 13; Logged passed levels are not continuous!
Missing info for user 14; Logged passed levels are not continuous!
Missing info for user 16; Logged passed levels are not continuous!
Missing info for user 17; Logged passed levels are not continuous!
Missing info for user 18; Logged passed levels are not continuous!
```

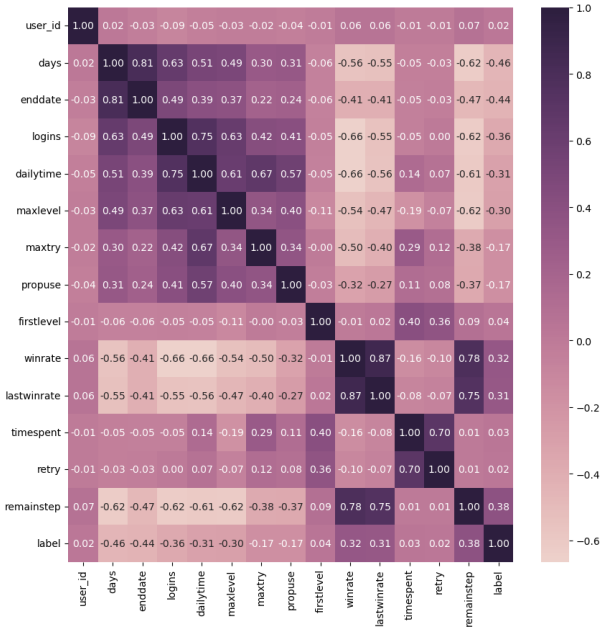
```
# if passed levels are continuous
if continuous_set(pass_levels):
    # if the last pass level is out of the 1509 levels
    if pass_levels[-1]>1509:
        max_levels.append(1509)
    else:
        max_levels.append(pass_levels[-1])
# if passed levels are not continuous
# still append the max levels but warn forehead
else:
    sys.stderr.write("Missing info for user {}; Logged passed levels are not continuous! \n"
    if pass_levels[-1]>1509:
        max_levels.append(1509)
    else:
        max_levels.append(pass_levels[-1])
```

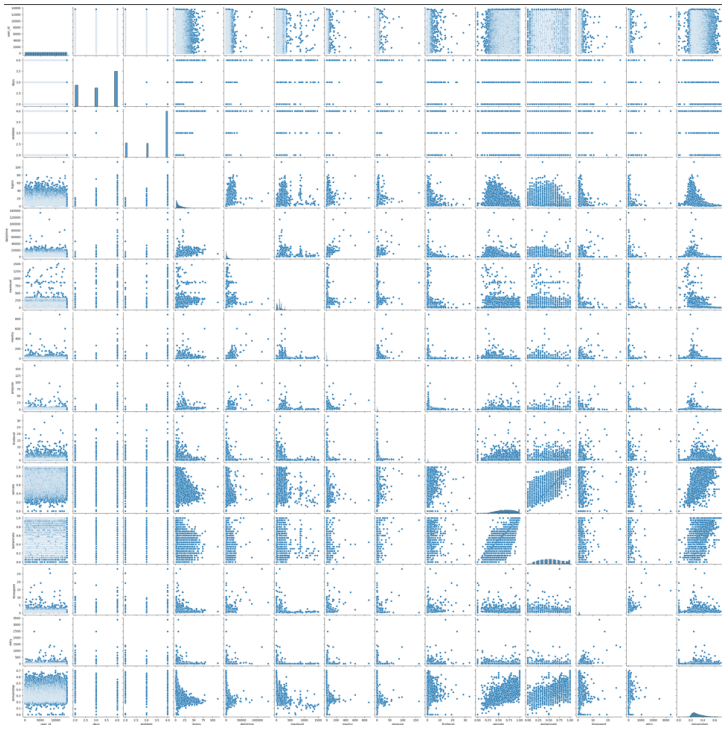
4.3 Construct Final Data

After extracting and calculating all feature values, we construct two final data collections for learning. These two sets are written out to .csv files so we could downcut the running time in the future by reading in rather than re-running the feature construction from Section 4.1-4.2 from time to time.

```
uniform_final_data.to_csv("./data/uniform_user_info.csv", encoding='utf-8', index=False)
weighted_final_data.to_csv("./data/weighted_user_info.csv", encoding='utf-8', index=False)
```

4.4 Correlation Analysis





From the right image, we could see:

For both uniform weighted data and pass rate weighted data, high correlation pairs are the same. They are: winrate - lastwinrate; dailytime -retry; login - retry; login - timespent. The correlation score matrix is next used(left image) for further evaluation.

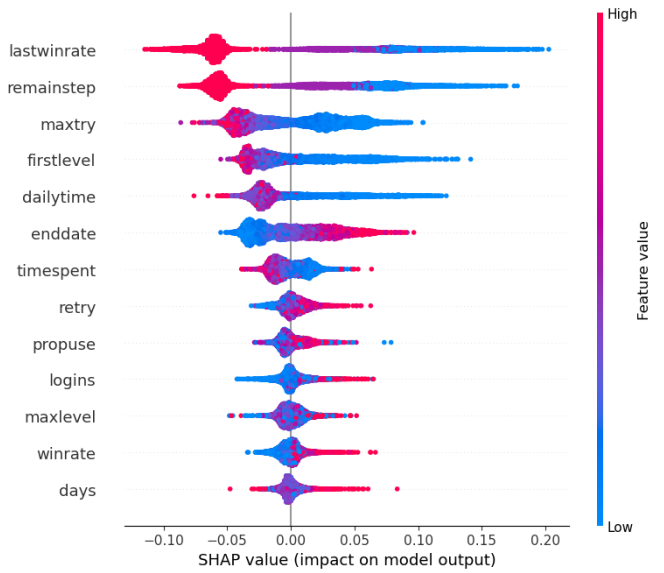
By observing the correlation heatmap, we have below conclusions:

1. Both the uniform weight data and non-uniform weight data have similar correlation matrices.
2. Feature "first level", "time spent", and "retry" do not have high correlation with the classification labels. We will test these three features later.
3. Feature "winrate" and "lastwinrate" are closely connected to each other. "Days" and "enddate" are also a high-correlated pair. However, the correlation score is not too high(under 0.9), so we will not consider to rule these features out.

For further analysis, we will use random forest classifiers and Shapley values to evaluate the explainability and interpretability.

4.5 Explainability and Interpretability

For further analysis, we will use random forest classifiers and Shapley values to evaluate the explainability and interpretability of our constructed features.



Predictive models are not easy to interpret. A recent technique to interpret models has stood out among others: SHAP (SHapley Additive exPlanations). SHAP measures the impact of variables taking into account the interaction with other variables[3].

The image has a y-axis indicating the variable name, in order of importance from top to bottom. The x-axis is the SHAP value. Indicates how much is the change in log-odds.

Feature `lastwinrate` and `remainsteps` are associated with high positive values on the target. When `lastwinrate` is high (or true) then the SHAP value is high. Players tend to stay for the game more when they win more easily during the last attempt.

Points in feature days have a comparatively large dispersion on high feature values. This suggests that when this feature is close to 0, it does not affect much on the classification.

An observation that is contrary to the previous analysis(in Section 4.4) is that feature `firstlevel` has a greater impact on the classification result. Feature `retry` and `timespent` have low Shapley values, that is, have low correlation regarding how the classification decision is made.

4.6 Portrait of Players

Based on the analysis above, we could do a persona over our players on how their game experiences can affect churn or stay.

1. Players' end dates have a positive proportion to their total login days. However, these two do not contribute much to decision making.
2. Time users spend on the game are closely related to their game experiences and persona(patience).
3. Players' stay or lose is closely connected to how easy they find the final few attempts(game levels)are. The game overall being too difficult or too easy can both result in user churn, due to the lack of instant gratification or challenge.
4. First-time experiences can have a comparatively large impact on users' feeling to the game.
5. A patient player is less likely to churn.
6. A player who spends less time playing the game daily is more likely to churn.

5 Test with Simple Classifiers

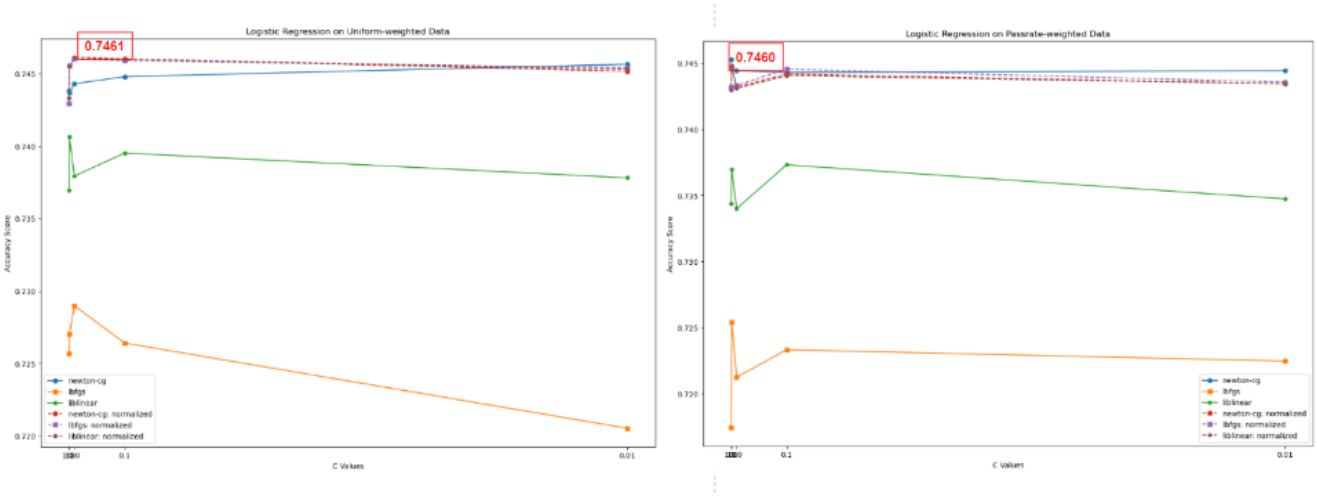
We will choose certain simple classifiers to run on two data sets. The main task is to:

1. Evaluate which data set carries out better learning results;
2. Identify unstable classifiers and check if any ensemble algorithms or deep learning could be introduced in improvement.

Following classifiers are tested: DecisionTreeClassifier, LogisticRegression, Naive Bayes, and Knn. Mean accuracy scores from cross validation are used to measure the performance of classification.

5.1 Logistic Regression

Hyperparameter tuning in this section is inspired by an online article “Logistic Regression Model Tuning with scikit-learn — Part 1”[4].



The best accuracy score has reached 0.746(normalized uniform-weighted dataset with lbfgs solver and C set to 1.0).

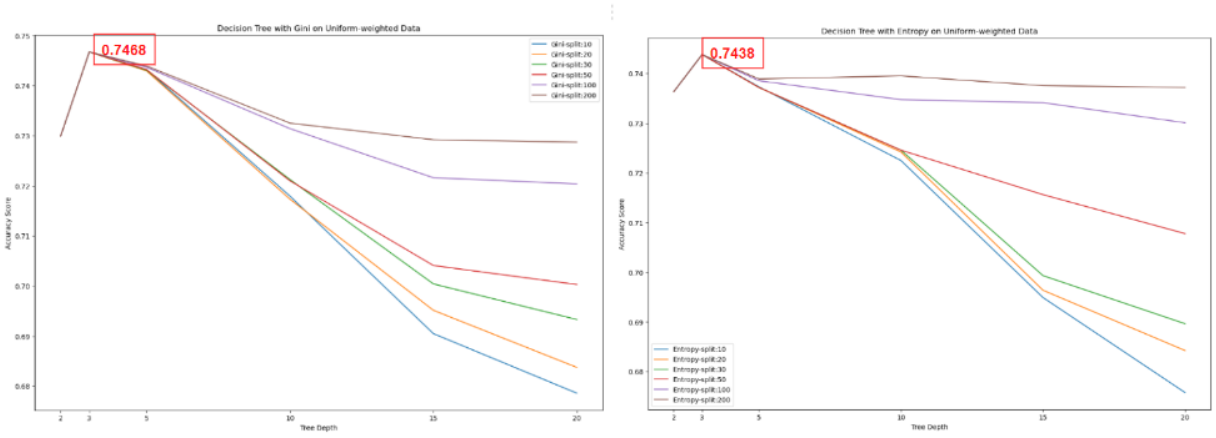
With logistic regression, weighted data carries out worse learning results than uniform-weighted data, especially when the regressor solver is set to 'lbfgs' and 'liblinear'. Normalized data set has better learning results, more accurate and stable predictions. This is due to the vanishing gradient problem during the training phase without normalization[6].

Among three solvers we tested, newton-cg and lgbfs are better than liblinear. This matches conclusions that "In general, if the dataset is small and the number of features is not too large, then the lbfgs or newton-cg solvers are good choices. If the dataset is large, then the sag or saga solvers are better choices"[7].

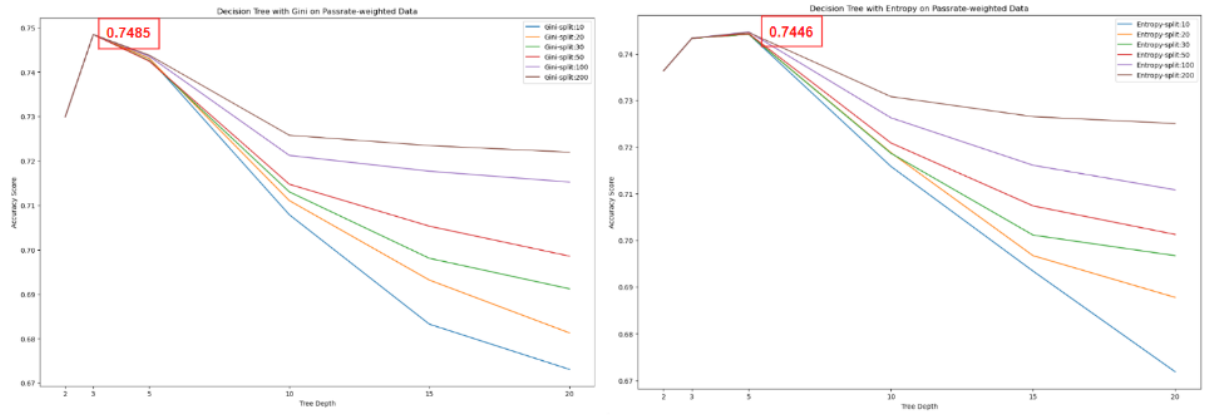
From now on, we will use normalized data sets for further experiment.

5.2 Decision Tree

Decision Tree on Uniform-Weighted Data



Decision Tree on Passrate-Weighted Data



Left two diagrams are with Gini and the right are with Entropy. The two on the top are with non-weighted data and the bottom two are with weighted data. Both two data sets are normalized.

In our data model, Decision Tree with Gini algorithm performs generally better than Entropy. And, Pass Rate-weighted data improves the classifier behavior at a small improvement on accuracy scores.

Accuracy drops as the tree depth grows, and climbs up as the minimum split sample amount increases. This refers to overfitting.

The best accuracy score is at 0.7485 on weighted data, with the tree's hyperparameter set to (gini, depth=3), and it is slightly better than logistic regression.

5.3 Naive Bayes

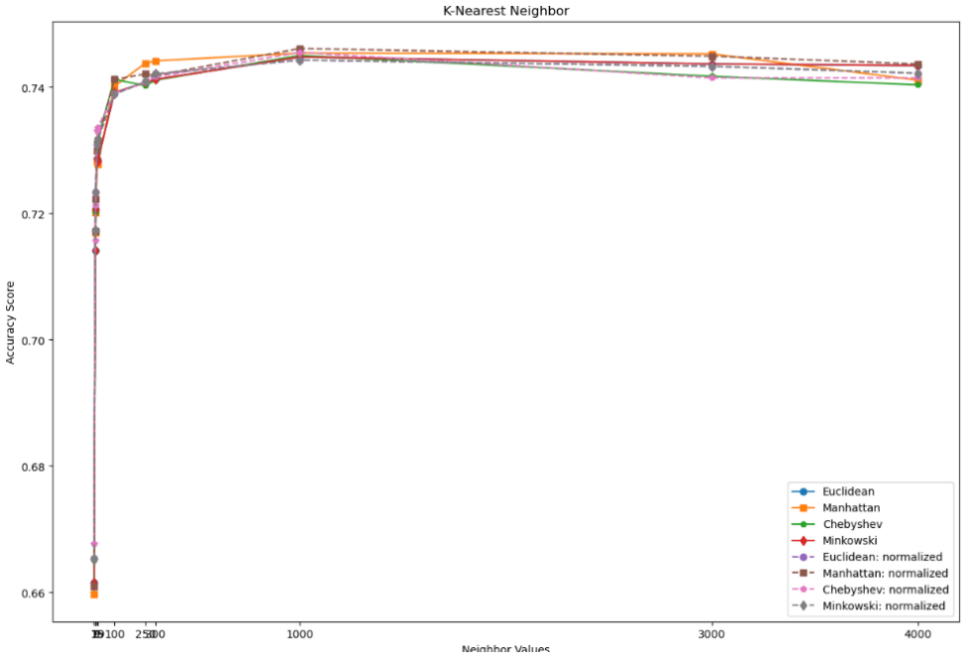
We test Multinomial, Complement and Gaussian Naive Bayes in this section. Naive Bayes has the weakest performances so far.

Gaussian Naive Bayes is used when we assume all the continuous variables associated with each feature to be distributed according to Gaussian Distribution[8]. Gaussian Naive Bayes here performs the worst, especially on the weighted data set. This implies that our data model does not follow Gaussian Distribution. Complement Naive Bayes have the highest accuracy scores, as they work well especially when the dataset is imbalanced. For Naive Bayes, uniform-weighted data works better.

```
Data: uniform; Model: MultinomialNB(); Accuracy: 0.7257948028997007
Data: uniform; Model: ComplementNB(); Accuracy: 0.7307005956889194
Data: uniform; Model: GaussianNB(); Accuracy: 0.6864455224149746
Data: weighted; Model: MultinomialNB(); Accuracy: 0.7153753561510442
Data: weighted; Model: ComplementNB(); Accuracy: 0.7267773740397446
Data: weighted; Model: GaussianNB(); Accuracy: 0.6423166798908404
```

We will test GaussianNB for the ensemble, since it has the worst performance. The goal is to figure out if ensemble learners can improve learning on non-Gaussian distributed data by Gaussian method. The guess for now is no.

5.4 K-Nearest Neighbors(Knn)



Performances with different distance metrics are similar; Among them Manhattan distance carries out the best results. The combination of Manhattan distance metric on weighted data has the highest accuracy at 0.7461. Manhattan distance is usually preferred over Euclidean distance when we have a case of high dimensionality.

Accuracy scores rise first as the neighbor amount grows, then slightly drop when the neighbor amount reaches almost 1/3 of the data set amount(overfitting). When the neighbor amount is set to 1, the whole model is underfitting.

5.4 Section Conclusion

Data sets will be normalized in later sections. Since some algorithms(classifiers) we will use later have involved distance calculation. In that case, we need to normalize our data as some models are sensitive to the order of magnitude of the features[6].

Uniform-weighted data performs better for Logistic Regression and Naive Bayes. Weighted data works better with Decision Tree. And both data sets work similarly for K-Nearest Neighbors. Therefore both data sets will be used for ensemble and deep learning. We could see this as a form of ablation study, to determine whether level pass rates have a positive impact over our model output.

Since most of the accuracy scores in the previous experiments are above 0.7, we need to pay extra care to avoid overfitting in ensemble learning.

6 Ensemble Learning

In this section, learning results will be evaluated on the validation set.

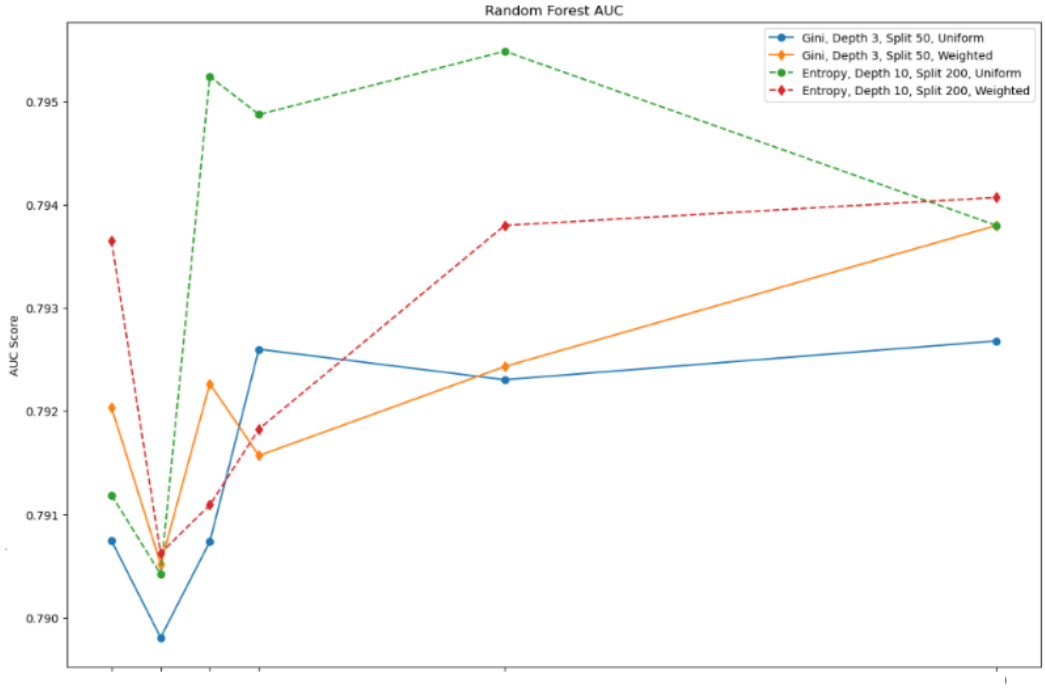
We know that error can be composed from bias and variance. A too complex model has low bias but large variance, while a too simple model has low variance but large bias, both leading to a high error but two different reasons. As a result, two different ways to solve the problem come into people's mind, variance reduction for a complex model, or bias reduction for a simple model, which refers to random forest and boosting[9].

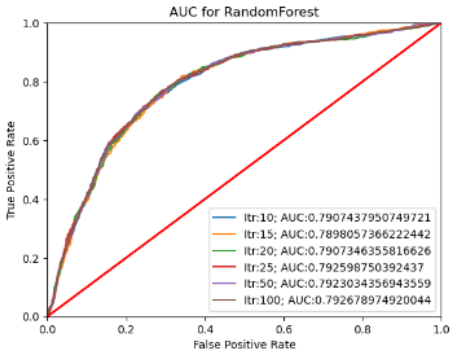
We will use two ensemble learners in this section: RandomForest and Bagging. Below listed the combinations of base classifiers, hyper parameters and datasets we are using:

Classifiers with same highlight color are control groups!

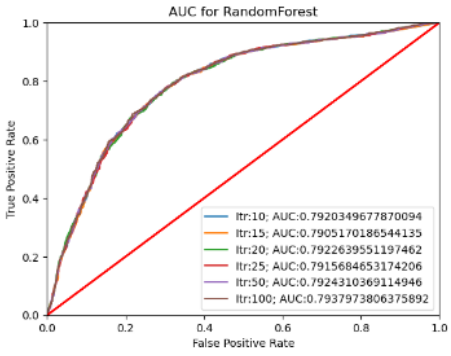
	Base Classifier	Classifier Parameter	Data (Normalized)	Accuracy Score	Reason
1	Logistic Regression	Solver: lbfgs C:1.0	Uniform-weight	0.7460	Best performance
2	Logistic Regression	Solver: lbfgs C:1.0	Passrate-weight	0.7332	1. Worse performance; 2. Check impact from game level passrates[1]
3	Decision Tree	Gini Depth:3 Minimum Split:50	Passrate-weight	0.7485	Best performance
4	Decision Tree	Gini Depth:3 Minimum Split:50	Uniform-weight	0.7433	[1]
5	Decision Tree	Entropy Depth:10 Minimum Split:200	Uniform-weight	0.7395	Second Peak for this combination of parameters
6	Gaussian NB		Passrate-weight	0.6423	Worst performance
7	Knn	N_neighbors: 1000 Metric: Manhattan	Passrate-weight	0.7461	Best performance
8	Knn	N_neighbors: 1000 Metric: Manhattan	Uniform-weight	0.7454	[1]
9	Knn	N_neighbors: 19 Metric: Chebyshev	Uniform-weight	0.7319	One of the worst performances due to underfitting

6.1 Random Forest

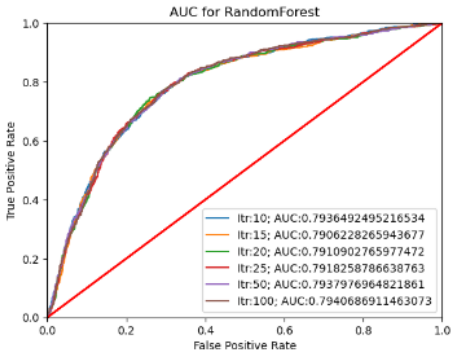




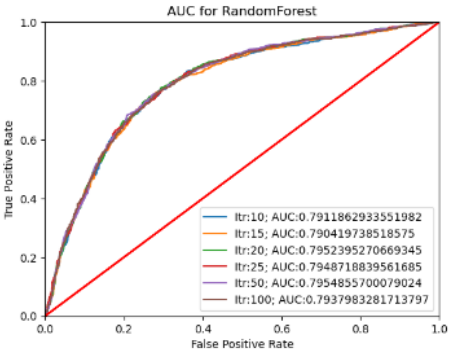
Gini Max Depth:3 Minimum Split: 50
Uniform-Weighted Data



Gini Max Depth:3 Minimum Split: 50
Weighted Data



Entropy Max Depth:10 Minimum Split: 200
Uniform-Weighted Data



Entropy Max Depth:10 Minimum Split: 200
Weighted Data

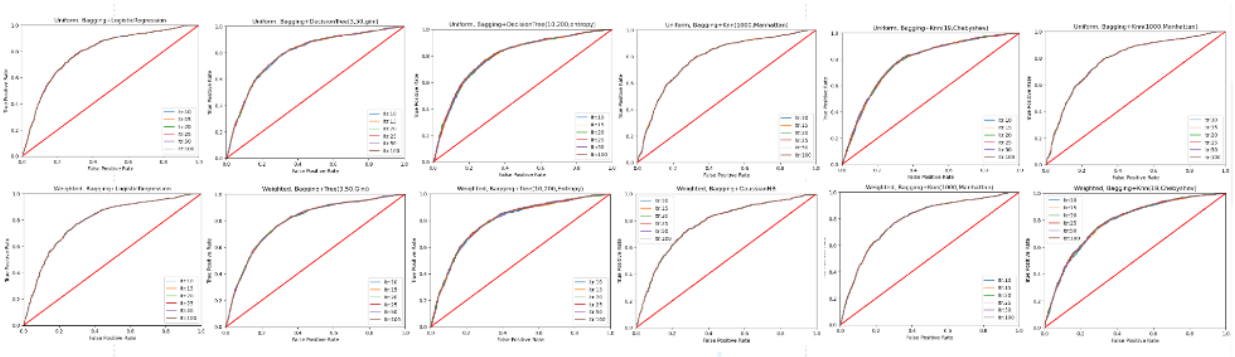
Compared to simple decision tree classifiers, both hyperparameter combinations have improved. The weaker classifier improves more than the stronger one in RandomForest. The best performance(high accuracy and stability) is still with the stronger decision tree classifier with accuracy score at 0.7550. Weighted data set provides better learning results than uniform-weighted data.

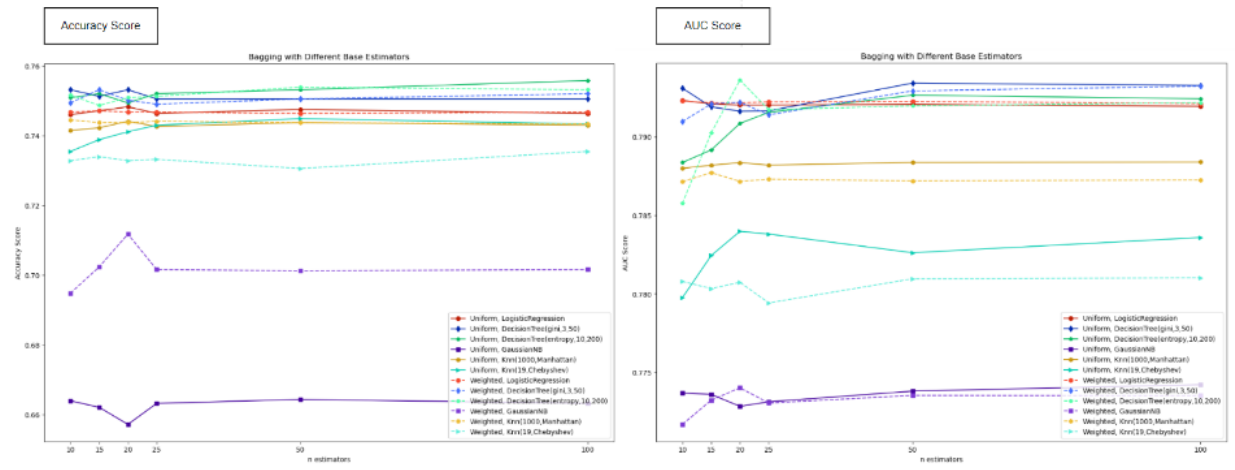
Roc AUC curves are similar with different n_estimator values. AUC scores all drop down when the estimator amount is set to 15 then climb up. Among what we tested, an entropy tree with depth at 10 and minimum split at 200 on uniform-weighted data, the AUC score reached the highest at 0.7955 (n_estimator: 50).

It's fair to have the conclusion that accuracy and performance of a random forest model improve as the tree amount increases. We will test the best accuracy and best AUC condition on the test set later, observing if any overfitting can happen.

6.2 Bagging Ensemble

Since we only have 13 features in total, we will not consider 'max_features' in parameter tuning. We will use 0.7 as max features. Estimator amount to be tested: [10, 15, 20, 25, 50, 100].





Compared to single classifiers, the bagging ensemble has brought improvements on prediction accuracy for all classifiers.

Gaussian Naive Bayes have better accuracy on the weighted data model than uniform-weighted one. It still has the lowest prediction accuracy and AUC scores. As we analyzed previously, our data does not follow a Gaussian distribution. Even though ensemble brings in improvement, Gaussian Naive Bayes does not work with our data logically.

Knn with 1000 neighbors voting out the final classification has comparatively low AUC and accuracy, this is most likely caused by overfitting. Knn with Chebyshev distance and neighbor amount sets to 19 also does not have ideal performances. We could assume that Chebyshev does not work with our data and our task.

For decision tree classifiers, decision tree(depth:3, minimum split:50, Gini) and decision tree(depth:10, minimum split:200, entropy), both accuracy scores and AUC scores follow a gentle upward trend as the estimator amount increases.

For logistic regression and Knn, the non-weighted data set gives better learning results. Decision trees and Gaussian Naive Bayes are the opposite.

The best accuracy score 0.7506 is with:

- Uniform-weight data, decision tree(depth:3, minimum split:50, Gini) and estimator amount set to 100.

The best AUC scores are:

- 0.7936: Weighted data, decision tree(depth:10, minimum split:200, Entropy) and estimator amount set to 20
- 0.7929:Weighted data, decision tree(depth:3, minimum split:50, Gini) and estimator amount set to 50

6.3 Section Conclusion

For all base classifiers, their AUC curves with different estimator amounts do not differ too much. This is potentially because each classifier itself already gives a promising learning result. Ensemble learning generally enhances the prediction accuracy as the estimated amount grows.

The best ensemble performances are all when the base estimator is set to the decision tree. For decision tree based ensemble classifiers, weighted data contributes more positive impact than non-weighted data.

Based on the analysis above, on the test set, we will run following:

- Uniform-weight Data+RandomForest(depth:10,split:200,entropy) -- AUC:0.7955 Accuracy:0.7509
- Weighted data + Bagging(estimator:50)+DecisionTree(depth:3,split:50,gini) -- AUC:0.7929 Accuracy: 0.7505
- Weighted Data + Bagging(estimator:50)+LogisticRegression -- AUC:0.7923 Accuracy: 0.7464

7 MLP and RandomizedSearchCV for Hidden Layer Optimization

The Multi-Layer Perceptron (MLP) Classifier in Sklearn is a powerful tool that can be used for a wide range of classification tasks. However, optimizing the hidden layers of an MLP is often a challenging task. Inspired by this article[10], we will explore how to optimize the hidden layers of an MLP Classifier using RandomizedSearchCV in Sklearn.

An MLP is a type of artificial neural network that consists of multiple layers of nodes. Each node in an MLP is a mathematical function that takes as input the weighted sum of the outputs of the nodes in the previous layer. One of the most important aspects of building an MLP Classifier is choosing the number and size of the hidden layers, which have a significant impact on the performance of the model.

A method that can be used to optimize the hidden layers of an MLP Classifier is RandomizedSearchCV. This method randomly samples a subset of hyperparameters and evaluates their performance. This approach is more computationally efficient than GridSearchCV and is ideal for large datasets.

```
"""To use RandomizedSearchCV for hidden layers optimization, we first need to define a parameter grid.
The parameter grid specifies the hyperparameters we want to optimize and their possible values."""

hidden_layer_sizes = [(10,), (15,), (20,), (50), (100),
                      (10, 10), (15, 15), (20, 20), (50, 50), (100, 100)]
param_grid = {'hidden_layer_sizes': hidden_layer_sizes}
```

```
mlp = MLPClassifier(max_iter=1000)
randomized_cv = RandomizedSearchCV(mlp, param_grid, n_iter=10, scoring='roc_auc')
```

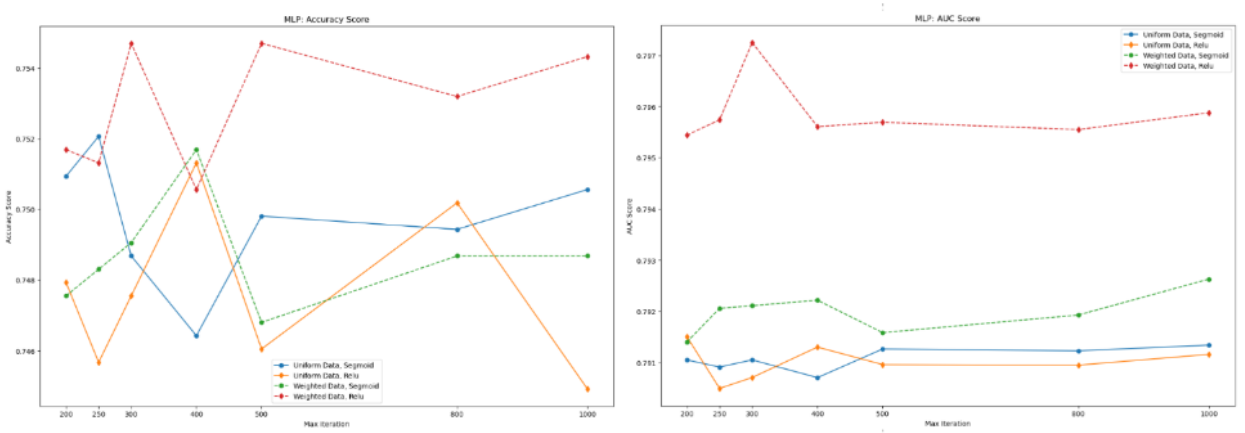
```
randomized_cv.fit(norm_weighted_x_train, weighted_y_train)
print(randomized_cv.best_params_)
```

```
{'hidden_layer_sizes': 50}
```

```
randomized_cv.fit(norm_uniform_x_train, uniform_y_train)
print(randomized_cv.best_params_)
```

```
{'hidden_layer_sizes': (20, 20)}
```

We will use the optimizing results above to train our model. Below code is the same for both uniform data and weighted data, just the hyperparameter changes for hidden layers.



MLP with Relu activation on Weighted Data has the best performance, with the highest AUC at 0.7957 when max iteration is 250. This is also the highest AUC score among the experiments we run.

The accuracy scores fluctuate a lot for all hyperparameter combinations. The AUC score generally follows an upward trend, drops down and then slowly climbs up again as the max iteration increases. Intuitively, this basically means that some portion of examples is classified randomly, which produces fluctuations, as the number of correct random guesses always fluctuate (imagine accuracy when coin should always return "heads"). Basically sensitivity to noise (when classification produces random results) is a common definition of overfitting[11].

From AUC perspective, weighted data contributes to better learning results compared to unweighted data. Also, Relu is better than Segmoid on our data model. Relu is known for faster calculation and faster convergence.

The weakest performance is MLP with Relu on unweighted data with 800 iterations, and the AUC is 0.7909.

We will run the best performance and the worst performance on the test set.

8 Final Models On Test Set

We will run the best performance and the worst performance on the test set.

	Data Set	AUC on Validation Set	AUC on Test Set
Random Forest(estimator:50) Depth:10 Split:200 Entropy	Uniform	0.7955	0.7341
Bagging(estimator:50) + DecisionTree(depth:3,split:50,Gini)	Weighted	0.7929	0.7911
Bagging(estimator:50) + LogisticRegression(lbfgs,C:1.0)	Weighted	0.7923	0.7911
MLP(Relu) Itr:250 Layer:(50,)	Weighted	0.7957	0.7911
MLP(Relu) Itr:800 Layer:(50,)	Uniform	0.7909	0.7341
MLP(Segmoid) Itr:1000 Layer:(20,20)	Weighted	0.7926	0.7911

Seen from the diagram above, the best AUC score throughout the whole experiment becomes stable at 0.7911.

AUC for two highlighted classifiers drop down massively from the validation set to the test set. This is where overfitting happens. If we want to further improve our data model, we need to consider re-construct our data set, since most of the experimented learning models converge to the same AUC.

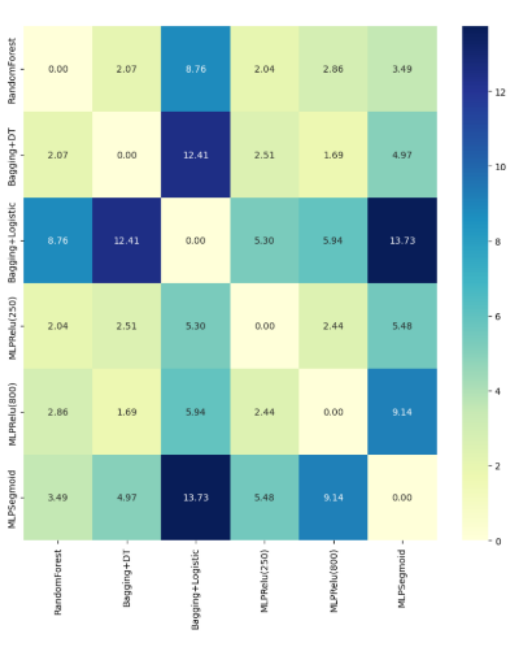
8.1 Paired T-Test

Comparing machine learning methods and selecting a model is a common operation in applied machine learning.

Models are commonly evaluated using resampling methods like k-fold cross-validation from which mean skill scores are calculated and compared directly. Although simple, this approach can be misleading as it is hard to know whether the difference between mean skill scores is real or the result of a statistical fluke[13].

Statistical significance tests are designed to address this problem and quantify the likelihood of the samples of skill scores being observed given the assumption that they were drawn from the same distribution. If this assumption, or null hypothesis, is rejected, it suggests that the difference in skill scores is statistically significant.

Paired T-Test is a common statistical significance test method. Here, we run each model on the test set for 5 times. The more precise way is to use 20 to 30 test sets (k-fold), or a 5x2 cross validation[14]. We only iterate 5 times here due to the time budget.



t Table

cum. prob	$t_{.50}$	$t_{.75}$	$t_{.80}$	$t_{.85}$	$t_{.90}$	$t_{.95}$	$t_{.975}$	$t_{.99}$	$t_{.995}$	$t_{.999}$	$t_{.9995}$
one-tail	0.50	0.25	0.20	0.15	0.10	0.05	0.025	0.01	0.005	0.001	0.0005
two-tails	1.00	0.50	0.40	0.30	0.20	0.10	0.05	0.02	0.01	0.002	0.001
df											
1	0.000	1.000	1.376	1.963	3.078	6.314	12.71	31.82	63.66	318.31	636.62
2	0.000	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	22.327	31.599
3	0.000	0.765	0.978	1.250	1.638	2.353	3.182	4.541	5.841	10.215	12.924
4	0.000	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	7.173	8.610
5	0.000	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	5.893	6.869
6	0.000	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	5.208	5.959
7	0.000	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	4.785	5.408
8	0.000	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	4.501	5.041
9	0.000	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	4.297	4.781
10	0.000	0.700	0.879	1.093	1.372	1.812	2.228	2.764	3.169	4.144	4.587
11	0.000	0.697	0.876	1.088	1.363	1.796	2.201	2.718	3.106	4.025	4.437
12	0.000	0.695	0.873	1.083	1.356	1.782	2.179	2.681	3.055	3.930	4.318
13	0.000	0.694	0.870	1.079	1.350	1.771	2.160	2.650	3.012	3.852	4.221
14	0.000	0.692	0.868	1.076	1.345	1.761	2.145	2.624	2.977	3.787	4.140
15	0.000	0.691	0.866	1.074	1.341	1.753	2.131	2.602	2.947	3.733	4.073
16	0.000	0.690	0.865	1.071	1.337	1.746	2.120	2.583	2.921	3.686	4.015
17	0.000	0.689	0.863	1.069	1.333	1.740	2.110	2.567	2.898	3.646	3.965
18	0.000	0.688	0.862	1.067	1.330	1.734	2.101	2.552	2.878	3.610	3.922
19	0.000	0.688	0.861	1.066	1.328	1.729	2.093	2.539	2.861	3.579	3.883
20	0.000	0.687	0.860	1.064	1.325	1.725	2.086	2.528	2.845	3.552	3.850
21	0.000	0.686	0.859	1.063	1.323	1.721	2.080	2.518	2.831	3.527	3.819
22	0.000	0.686	0.858	1.061	1.321	1.717	2.074	2.508	2.819	3.505	3.792
23	0.000	0.685	0.858	1.060	1.319	1.714	2.069	2.500	2.807	3.485	3.768
24	0.000	0.685	0.857	1.059	1.318	1.711	2.064	2.492	2.797	3.467	3.745
25	0.000	0.684	0.856	1.058	1.316	1.708	2.060	2.485	2.787	3.450	3.725
26	0.000	0.684	0.856	1.058	1.315	1.706	2.056	2.479	2.779	3.435	3.707
27	0.000	0.684	0.855	1.057	1.314	1.703	2.052	2.473	2.771	3.421	3.690
28	0.000	0.683	0.855	1.056	1.313	1.701	2.048	2.467	2.763	3.408	3.674
29	0.000	0.683	0.854	1.055	1.311	1.699	2.045	2.462	2.756	3.396	3.659
30	0.000	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.385	3.646
40	0.000	0.681	0.851	1.050	1.303	1.684	2.021	2.423	2.704	3.307	3.551

From the t-static values between the 6 models mentioned in Section 8 and compare with the t-table, there are some interesting observations:

1. We assume a p-value = 0.05, and since we re-run each model for five times, the degree of freedom is set to 5-1=4. According to the table, if t between two pairs is within [-4.604, 4.604], we cannot reject the null hypothesis and there is no significant difference between our 2 classifiers.
2. RandomForest model has AUC in Section 6.1 at 0.7431. MLP(Relu, iter:250) and MLP(Segmoid, iter:1000) at the time had AUC both at 0.7911. Although there exists a large difference between the AUC scores, the t values between RandomForest and these two models are within the score range, and there is no significant difference between them from a statistical point of view.

MLP(Relu, iter: 800) has encountered similar situations. Again, this proves again that purely relying on the score value comparison can be misleading.

3. There are four models having the same AUC scores on the test set in Section 8 diagram. They are:
    a. Model b has rejected the null hypothesis and has significant differences with all the other models. The difference is most significant when paired with Model d.
    b. Model a has small differences with other models(except b), so does model c.

- Bagging(estimator n=50) + DecisionTree(Gini, depth:3, minimum split:50)
- Bagging(estimator n=50) + LogisticRegression(solver: lbfgs, C: 1.0)
- MLP(Relu, iter=250, layer:(50,))
- MLP(Segmoid, iter=1000, layer:(20,20))

8.2 Ablation Study on Features

We will drop only one feature at a time from the pool and use model b in Section 2 with T-test. This will run on the validation set. The t-test questions whether the difference between the groups represents a true difference in the study or merely a random difference. Higher values of the t-score indicate that a large difference exists between the two sample sets. The smaller the t-value, the more similarity exists between the two sample sets[15].

	Mean AUC	Difference with all features	t static	Reject Null Hypothesis
All features	0.791164	/	/	/
days	0.790953	-0.038%	5.710184580874714	Y
enddate	0.791165	+0.0001%	3.3659627663726877	N
logins	0.791101	-0.008%	5.177956322292882	Y
dailytime	0.791276	+0.014%	3.4751906066841345	N
maxlevel	0.790781	-0.049%	4.675745287745806	Y
maxtry	0.791471	+0.03%	2.0961925205275955	N
propuse	0.790848	-0.050%	2.023291182817654	N
firstlevel	0.791187	+0.0029%	3.1884632391068535	N
winrate	0.791173	+0.0011%	2.498909829857978	N
lastwinrate	0.791186	+0.0028%	4.3559072599807935	N
timespent	0.791083	-0.020%	5.593531270151375	Y
retry	0.791394	+0.029%	3.0790685895575662	N
remainstep	0.791148	-0.0021%	5.352140721897239	Y

9 Conclusion

Under this task and with our constructed data model, ensemble algorithm and MLP works better than single classifiers. Most of the best performances from Ensemble and MLP converge to the same AUC.

The best AUC score is 0.7911 on the test set. Bagging(estimator amount: 50) with Decision Tree(Gini, Depth:3, Minimum Split:50) as the base classifier and MLP(Relu activation, iterate: 1000, hidden layer: (20,20)) are more stable under this occasion.

Features in the constructed data do not have too close correlation with each other. Seen from the Shapley diagram, they also do not have too large an impact over the model output. The highest Shapley is around 0.15 with the last win rate. Therefore, if we want to significantly improve learning results, it is mandatory to reconstruct the data set. As we mentioned earlier, since the last win rate has more obvious influence over the classification strategy, the re-construction could start from there. For example, in random forest one could bias distribution from which you sample features to analyze towards the ones that you are interested in. Another potential method is to weighted features with significant influences by some per-level information generated from last win rate. A risk of this method is that we add in more correlation between last win rate and modified features. This bias may shadow the true rule of classification. There is an article "8 Ways to Improve Accuracy of Machine Learning Models"[12] updated in 2023, we could also test if techniques mentioned can be used for our task.

Due to the limitation of time, we are not able to complete all planned experiments in this assignment. In the future, we'd like to try to reconstruct the data model for LSTM learning. It would also be interesting to do ablation study on a voting ensemble to see if learning results can be enhanced by a combination of multiple classifiers.

Also, loads of manual intervention happens in the feature construction. The best way would probably be to avoid manual intervention and directly feed the raw data to the learning model. Under this circumstance, a way to deal with the core files with two dimensions(users and game) would be to append the game metadata based on the user attempted level. There is a greater possibility in this that the AUC gets lower, since there are more noises in raw data. However, the prediction with this method is more in line with real rules. We will come back to this task again when we have better understanding over deep learning in the future.

10 Extended Experiments

Limitation for this section is that all experiments are one-off. There is no statistical analysis regarding the change of scores.

10.1 Features to Higher Space

Inspired by the article "8 Ways to Improve Accuracy of Machine Learning Models"[12] mentioned above, we tested outlier removal and dimension promotion through polynomial features.

With polynomial expansion degree set to 4, the combination of Tree--depth:3, split:50 and Bagging--n:70, sample:0.7, feature:0.6 has produced the highest AUC 0.7935.

10.2 Voting Ensemble

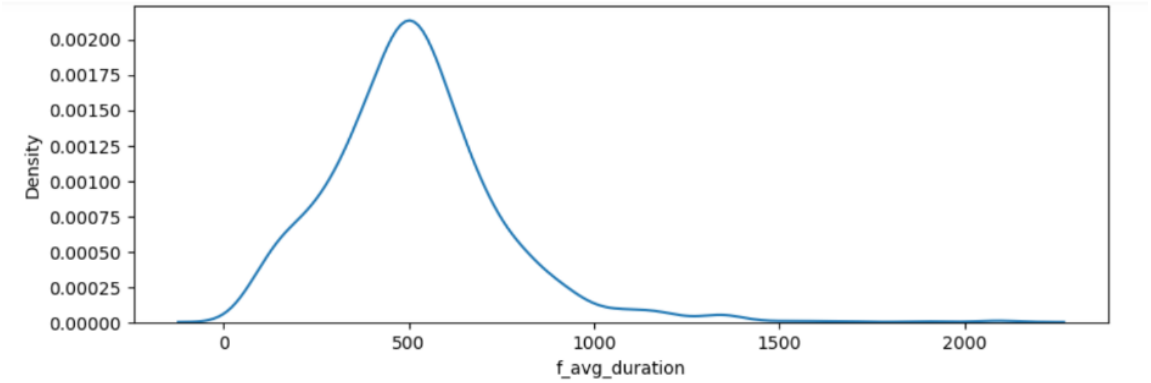
We use a voting ensemble with: Logistic Regression, Decision Tree, Knn and SVM. An ablation study is also run on this ensemble model, but the experiment result is not documented and due to the imprecision of experimental procedures(not compared statistical significance), we will not use this experiment for any conclusion.

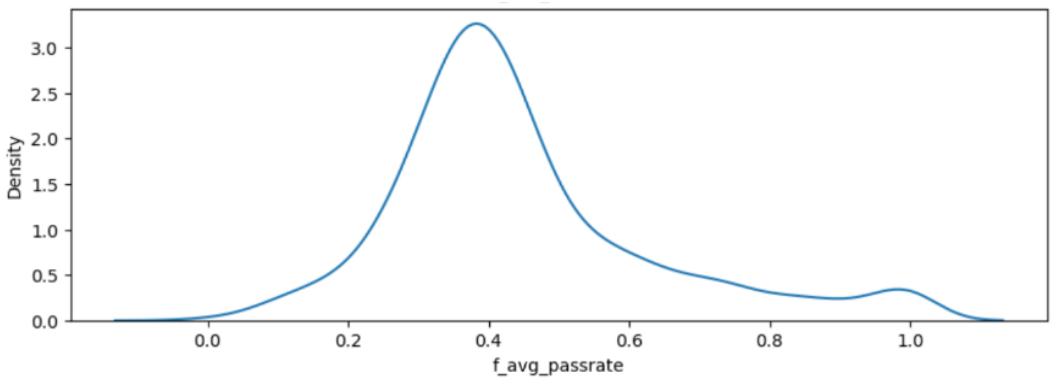
10.3 Data Construct

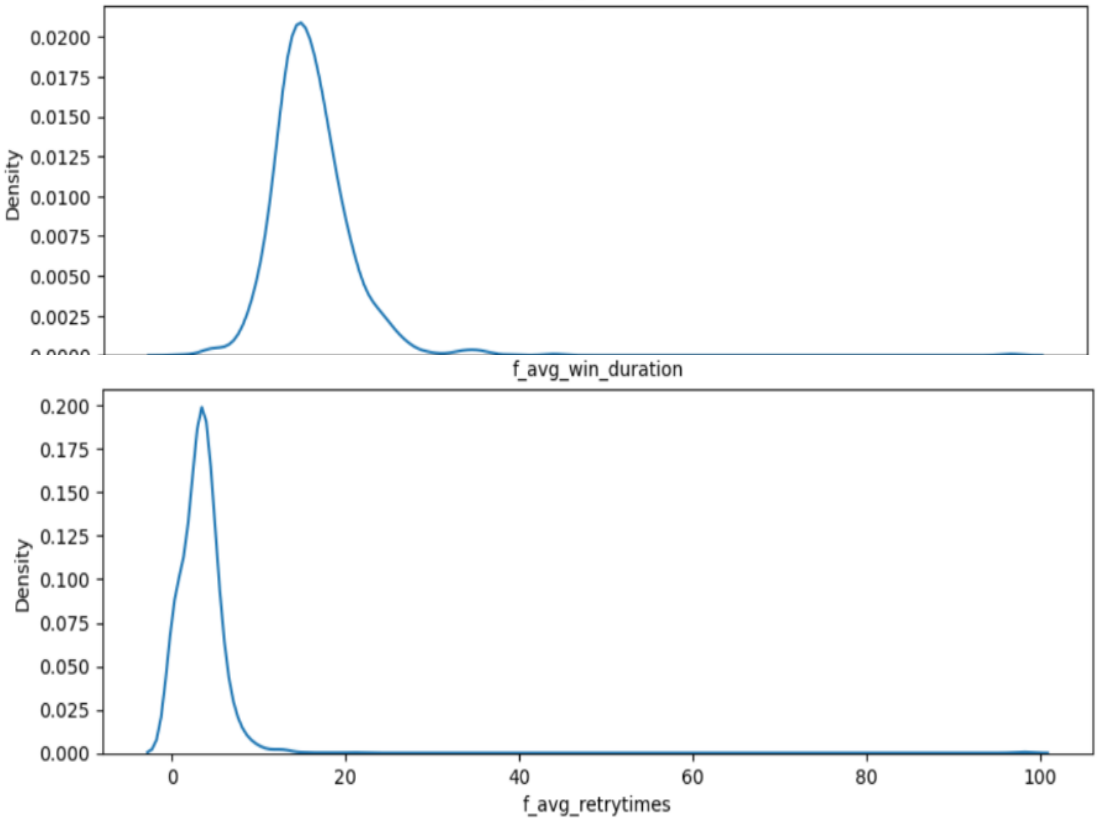
In the original experiment, lots of manual interference happens in the data set construction, and we interpolate many features with a weight matrix derived from game average metadata. However, average information can lead to risks while training: outlier reduction, not considering users groups and so on.

Since our data does not include any user group related data, such as gender, age and so on, here, we only observe the data distribution.

As can seen from the image, all 4 average data's distributions are quite different. Therefore, we will limit the usage of metadata.







We use two methods for this experiment to construct dataset:

a. Append metadata information after each sequence attempts

b. Fill missing sequence data with metadata average values

Both two methods have carried out lower AUC scores with the models in the above sections. But we are not able to analyze further due to the time budget.

11 References

[1] 10 Ways to Lower Your Mobile Game's Churn Rate in 2023
https://www.blog.udonis.co/mobile-marketing/mobile-games/churn-rate

[2] Curse of dimensionality
https://en.wikipedia.org/wiki/Curse_of_dimensionality#:~:text=The%20curse%20of%20dimensionality%20refers,physical%20space%20of%20everyday%20experience

[3] A gentle introduction to SHAP values in R
https://blog.datascienceheroes.com/how-to-interpret-shap-values-in-r/

[4] Logistic Regression Model Tuning with scikit-learn — Part 1
https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5

[5] Do I need to tune logistic regression hyperparameters?
https://medium.com/codex/do-i-need-to-tune-logistic-regression-hyperparameters-1cb2b81fca69

[6] Which models require normalized data?
https://towardsdatascience.com/which-models-require-normalized-data-d85ca3c85388#:~:text=Logistic%20regression%20requires%20normalization%20as,problem%20during%20the%20training%20phase.&text=If%20you%20train%20a%20linear,as%20indicators%20of%20feature%20importance.

[7] The Choice of the Optimal Solver for Logistic Regression Fitted to MNIST
https://saturncloud.io/blog/the-choice-of-the-optimal-solver-for-logistic-regression-fitted-to-mnist/

[8] Naive Bayes Algorithm: A Complete guide for Data Science Enthusiasts
https://www.analyticsvidhya.com/blog/2021/09/naive-bayes-algorithm-a-complete-guide-for-data-science-enthusiasts/#:~:text=Gaussian%20Na%C3%AFve%20Bayes%20is%20used,is%20also%20called%20Normal%20distribution.

[9] Is random forest a boosting algorithm?
https://stats.stackexchange.com/questions/77018/is-random-forest-a-boosting-algorithm#:~:text=Random%20Forest%20is%20a%20bagging,to%20achieve%20a%20low%20error

[10] Sklearn MLP Classifier Hidden Layers Optimization (RandomizedSearchCV)
https://saturncloud.io/blog/sklearn-mlp-classifier-hidden-layers-optimization-randomizedsearchcv/

[11] Why is the validation accuracy fluctuating?
https://stats.stackexchange.com/questions/255105/why-is-the-validation-accuracy-fluctuating

[12] 8 Ways to Improve Accuracy of Machine Learning Models
https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/

[13] Statistical Significance Tests for Comparing Machine Learning Algorithms
https://machinelearningmastery.com/statistical-significance-tests-for-comparing-machine-learning-algorithms/

[14] Paired t-test to evaluate Machine Learning classifiers using Python
https://towardsdatascience.com/paired-t-test-to-evaluate-machine-learning-classifiers-1f395a6c93fa

[15] T-Test: What It Is With Multiple Formulas and When To Use Them
https://www.investopedia.com/terms/t/t-test.asp#:~:text=This%20calculated%20t%2Dvalue%20is,between%20the%20two%20sample%20sets