

COSI 148B – Term Project

Predict Student Performance from Game Play

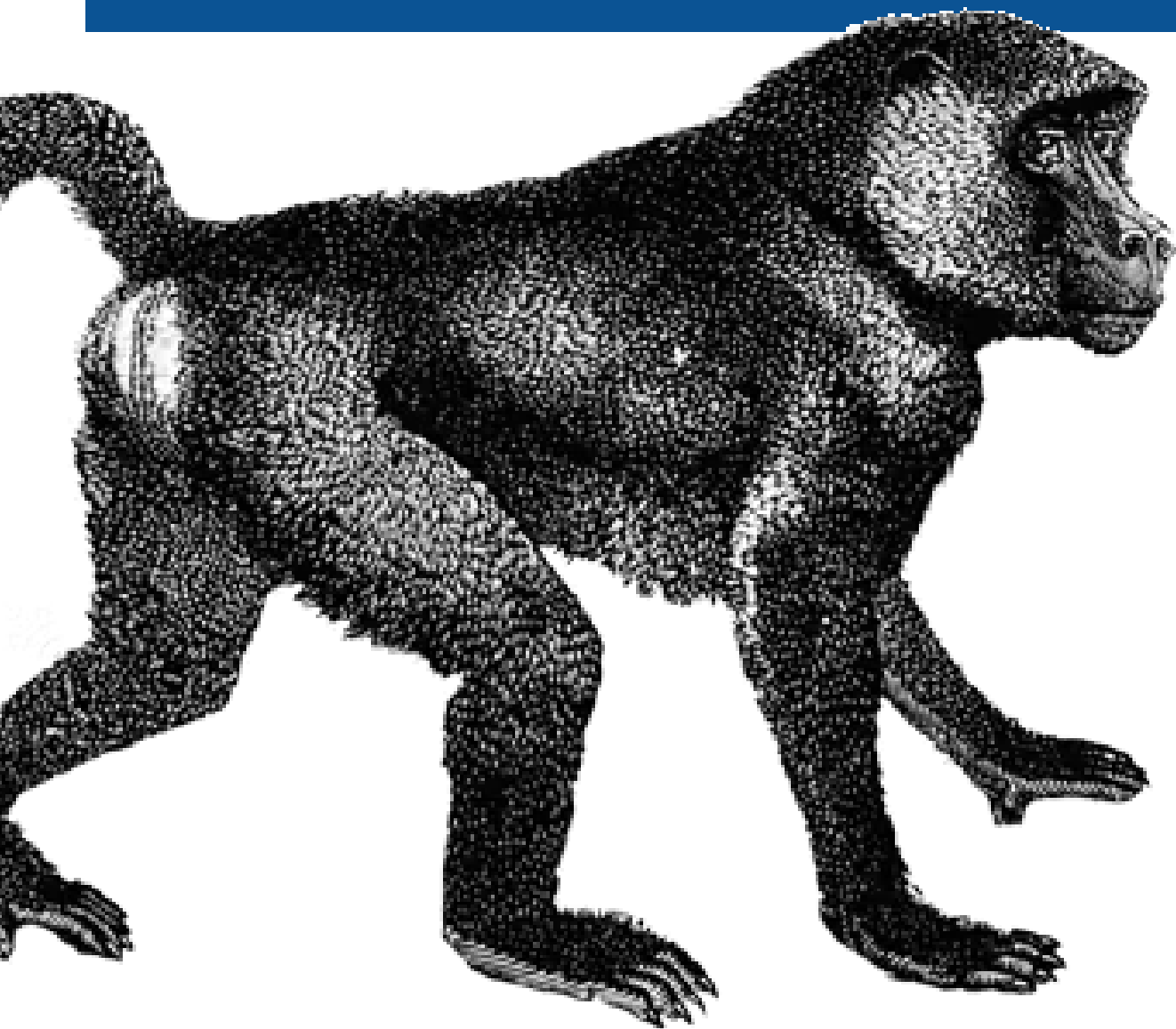


Table Of Contents

<u>Introduction</u>	3
<u>Basic Idea</u>	4
<u>Data Processing</u>	4
<u>Models</u>	7
<u>Final Model</u>	9
<u>Conclusions</u>	11
<u>Takeaways</u>	11
<u>Next Steps</u>	12
<u>Contributions</u>	12
 <u>References</u>	 13

Introduction

This report serves as the final project report for COSI 148B Introduction to Machine Learning with Economic Applications course over the time period of Spring 2023. The team members who worked on this project were Tal Kronrod, Nicole Meng, and Facundo Roitman.

The goal of this project was to design a model that predicts student performance during game-based learning in real time as explained in the guidelines of the *Predict Student Performance from Game Play* Kaggle competition. The users, alluded to in the data sets with their corresponding session number, will go through 18 questions which can be answered correctly or incorrectly.

The input data consists of 26,296,946 log events including elapsed time events (the time in milliseconds from session began until time of the event), screen coordinates information (coordinates of the click reference of the event), and hover duration information (in milliseconds). Additionally, for the data corresponding to the training sample, we have another frame determining whether the results for the user in question are correct or incorrect (they are matched to each event based on session id number and question number). When importing our data set, we divided the frame into 10 subsets in order to avoid memory errors, and randomly selected a subset to avoid any bias in our predictions.

After dividing and choosing a slice of the data set provided, we carried out the necessary data processing to successfully test different prediction models. This data processing, which will be further explained in the *Data Processing* section of this report, included normalization of continuous variables, one-hot-encoding for categorical variables, binarization and grouping among others. Our objective in doing this was to feed our models the cleanest version of the data sets in order to judge their accuracy in the most efficient conditions.

We continued to test this data with five models we learned in class: Linear Classifier, Logistic Classifier, Tree Classifier, and Random Forest. After running each model, we concluded that the Random Forests model performed the best out of all of them, which we will further explain in the Model section of this report.

Finally, we used our trained model to predict the testing data and submit it to the Kaggle competition, receiving a macro F1 score of 0.656.

Basic Idea

The basic idea of this project is to build a model to predict student performance during game-based learning in real-time. The dataset given is one of the largest open datasets of game logs. We were given four datasets: train.csv, which contains all the training set of the data, test.csv, which is the testing set of data, train_labels.csv, which contains the correct labels for the training data, and the sample_submission.csv, which demonstrates the sample submission format. In our data processing and model selection process, we will only look at the training data and its corresponding labels. The basic idea in data processing is to first examine all the columns and understand what they mean and correspond to. Then we will drop the columns that does not intuitively contribute to our prediction. After that, we plan to do some feature engineering on both the categorical and the continuous data, converting some of the column values into binary numbers, and grouping rows together based on certain attributes. Note that at the same time, we need to worry about memory issues since our training dataset is very large and loading all of the training data at once will crash the program. So we decided to train our model on a subset of the training data and perform pruning on the data while keeping the a decent final score.

After loading and processing the training data, we decided to try some of the models we talked in class including linear classification, decision trees, tree classifier, logistic classification, and random forests. After obtaining the F1 scores on these models, we will choose the best performing model as our final model and produce the final submission file.

Data Processing

Data Description

We originally received a training data set consisting of 26,296,946 log events with over 23,000 independent user sessions. Each independent user session was also separately given a list of 18 boolean values, corresponding to whether the user answered the questions in levels 1-18 correctly. This data was separated into two files, the first named 'train.csv' which contained the session_ids and the corresponding feature columns, the second file named 'train_labels.csv' contained the session_ids, question numbers, and whether the corresponding question was answered correctly by the users.

We analyzed each column's description and data type and created the following table for conceptual visualization and understanding of each column. With each column, we specified the description, data type, and universe (range of all possible values).

Columns Names and Characteristics:

Column Name	Description	Data Type	Universe
session_id	the ID of the session the event took place in	integer	>23,000 unique ids
index	the index (order) of the event for the session	integer	[0 - infinity)
elapsed_time	how much time has passed (in milliseconds) between the start of the session and when the event was recorded	integer	[0 - infinity)
event_name	the name of the event type	categorical	11 unique events
name	the event name (e.g. identifies whether a notebook_click is is opening or closing the notebook)	categorical	6 unique sub-events
level	what level of the game the event occurred in	integer	[0 - 22]
page	the page number of the event (only for notebook related events)	float	[0 - 6]
room_coor_x	the coordinates of the click in reference to the in-game room (only for click events)	float	[-1,990 - 1,260]
room_coor_y	the coordinates of the click in reference to the in-game room (only for click events)	float	[-918 - 544]
screen_coor_x	the coordinates of the click in reference to the player's screen (only for click events)	float	[0 - infinity) Note: right skewed, most cluster between 1 - 1,000
screen_coor_y	the coordinates of the click in reference to the player's screen (only for click events)	float	[0 - infinity) Note: left skewed between 0 - 650
hover_duration	how long (in milliseconds) the hover happened for (only for hover events)	float	[0 - infinity)
text	The text the player sees during this event	categorical	597 unique events
fqid	the fully qualified ID of the event	categorical	128 unique ids
room_fqid	the fully qualified ID of the room the event took place in	categorical	19 unique ids
text_fqid	the fully qualified ID of the text the player sees	categorical	126 unique ids
fullscreen	whether the player is in fullscreen mode	binary	0 or 1
hq	whether the game is in high quality	binary	0 or 1
music	wether the music is on or off	binary	0 or 1
level_group	which group of levels - and group of questions - this row belongs to	categorical	['0-4', '5-12', '13-22']

Figure 1

Data Processing

We needed to pre-process our data for many reasons. First, the size of the training dataset was too large to hold in the 8GB memory space that was allocated by Kaggle. To address this issue, we split the dataset into 10 roughly equal partitions and randomly selected one of the sections to train on. In this way, we will save computational resources and power including memory space and running time, as well as keeping the subset of data representative of the whole dataset through randomization. Using this method, the dataset only took up 2.5GB of memory before any feature engineering steps were taken, and freed up the necessary memory space to complete the rest of the code execution. The partition we used had 2,684,191 rows in its log and 2357 unique session ids.

Second, the log of events and the question answers were not associated directly. This made it difficult to work with the data and load it into the model. It also made it difficult to create a simple true label for each row in the dataset. To address this issue, we extract the session ids and question numbers of the label dataset and assign the true label during training time, where a model is trained for each question. Essentially, we trim the log up to the point of the question being asked for each question (1 through 18), and the true label set is assigned at that time by extracting the relevant question number for each session id.

Third, we chose to use feature engineering to further improve our data before feeding it into our models. We chose to normalize the following continuous features: `elapsed_time`, `room_coor_x`, `room_coor_y`, `screen_coor_x`, `screen_coor_y`, and `hover_duration`. We chose to normalize the continuous features in line with the material learned in the course. More specifically, since the continuous variables universe space in this training dataset is of different scales, with different levels of skewness, normalization can help prevent bias towards feature columns. Moreover, since our final submission file will be generated on an unseen set of test data, normalizing the features can improve the generalization performance of the model on the testing data. This is because the model will be less sensitive to small variations in the feature values due to scaling differences. Additionally, we chose to convert the following columns to binary: `name`, `text`, `fqid`, and `page`. While we were inspecting the types of data present in those columns we noticed that a majority of rows were empty for those features, so we chose to convert these columns to binary to represent true if there was data present in the cell, and false otherwise. Lastly, we chose to drop the `level` and `text_fqid` columns since they were colinear with the `level_group` and `text` columns, respectively. We also dropped the `event_name` and `room_fqid` columns instead of converting them to dummy columns since it significantly reduced the model runtime due to the reduced number of columns.

We also tried to use principal component analysis on the continuous columns in our dataset. We chose to use 3 principal components for the 6 columns, which contained ~85% of the variance of

these columns. However, once these principal components were substituted for the continuous columns, the F-score reduced from 0.662 to 0.591 pre-submission.

Models

As a group, we decided to test four different models: Linear Classifier, Logistic Classifier, Tree Classifier, and Random Forest. In this section, we will visit and describe the first three of those models. It is important to highlight that, for stylistic purposes, we chose not to include the failing models in our source code since the actual commands were not too different from those reviewed in class.

Model 1: Linear Classifier

Our initial approach was encouraged by our work in Homework 2. The subject material and objective of said assignment was to introduce us to the Linear Regression model given a data set. Because of the binary nature of our current project, we decided to use a Linear Classifier (as opposed to a Linear Regressor).

A Linear Classification model predicts the classification of the input data based on a hyperplane in the feature space. Simply put, this model separates the data with a straight line and concludes that every observation on one side of the line will be labeled correct, and the observations on the other side will be labeled incorrect. This is achieved by assigning weights to each feature of the model such that the weighted sum of the features for a given data point determines its position in the feature space. Linear classification models are most commonly used in the contexts of fraud detection, image recognition, and medical diagnosis.

In the context of this project, the training of our Linear Classification model gave each one of our 20 features a weight, which were then used when making a prediction for new observations. However, as we have witnessed in homeworks and class discussions, Linear Classification is limited when it comes to complex data sets like the one we were given. The reason for this limitation is simple: Linear Classification is best suited for data that has copious amounts of columns and whose decision boundary (the “straight line” dividing the data into correct or incorrect predictions) is linear or close to linear. After going through the *Data Processing* section of this report, anyone would agree that this data is by no means linear (or linear resemblant), hence why this model failed to make a good prediction of our data.

Model 2: Logistic Classification

Since our final prediction should be a binary number, we also wanted to try the logistic classification model in hope of a better f1 score. Furthermore, since linear regression did not perform as well, we were also hoping to use a logistic classification model to catch nonlinear relationships between the predictor variables and the outcome.

Logistic classification, also known as logistic regression, is a type of supervised learning algorithm used in machine learning and statistics to classify data into two or more categories. The goal of logistic classification is to build a model that can predict the probability of an input belonging to a particular class, given a set of input features. The algorithm works by fitting a logistic function to the data, which maps the input features to a probability value between 0 and 1. This function is then used to make predictions on new data by setting a threshold value, typically 0.5, above which the input is classified into one of the categories.

After trying Logistic Classification, even though our f1 score improved, it was still not ideal. We think that there might be a more complicated relationship between the features and the predictions other than logistic function can capture, creating an overfitting problem. Therefore, we decided to try tree classifier next.

Model 3: Decision Tree Classification

Even though the Logistic Classification model also underperformed, we did notice a small improvement from our initial attempt, which further established our view of the nonlinearity of this data set. Therefore, our next step was inspired by Homework 3: Decision Tree classifiers.

Decision Tree classifiers make decisions by following a series of yes-or-no questions based on the features of the data they are trained with. The model is trained by choosing the most informative feature of the model and splitting the data into two subgroups based on that feature. It then recursively continues this process in the subgroups with the new most informative feature and creates new splits until it reaches a stopping criterion. Once created, this tree can be used to make predictions for new, unseen observations.

For the scope of this project, there is one major drawback for the Decision Tree Classification model which is its proneness to overfitting when trees are too complex. Because of this major weakness, the model yielded a lower F1 score than expected. Nevertheless, it was better than those of the past models.

Final Model: Random Forest Classifier

The final model we tried was a random forest classifier. We chose to try it as it was one of the final models we were introduced to in course and the guidance we received while working on this project.

Theoretically, this model is a good choice for this dataset for multiple reasons. First, the model is able to handle both continuous and categorical variables, which in our case is useful because our training dataset is a mix of both. Second, RFCs are generally good at capturing the non-linear relationship between the features and the outcome by building a set of decision trees, each of which partitions the column space into regions based on the training data. Each region has its own predicted outcome, and the model uses these predictions to make a final prediction by combining the predictions of all the decision trees. Therefore, random forest classifiers can capture interactions and non-linear effects. Furthermore, RFCs are also less prone to over-fitting than decision trees since they use an ensemble of trees. We care about the overfitting problem because it could result in poor generalization to new, unseen data. Essentially, the overfitting problem happens when a model is capturing and making decisions based on noise and random fluctuations of the training data instead of focusing on the underlying patterns and relations that will generalize well on unseen testing data. Finally, RFCs usually handle high-dimensional datasets well due to the fact that it randomly selects a subset of predictor variables at each split point.

To train our random forest model we used group k-fold with 5 folds by splitting the log by session_ids so that each fold would have an independent set of session_ids of any other fold. For each fold, we would iterate through each of the 18 questions that had to be answered, and train a model for that individual level. Each model would be fed a trimmed version of the training log, such that it would not see events past the question that the model is being asked to predict. After training was concluded, we would pick the best model for each level out of the 5 folds, giving us 5 possible models to choose from.

Our random forest classifier had the best performance relative to the other models, with a macro F-score of 0.662. We achieved this F-score using a random forest classifier with the default sci-kit learn training parameters, with the only exception being a maximum tree depth of 6. We limited the depth of the trees due to the Kaggle 12-hour runtime constraint. When we tried greater depth trees, we would breach this Kaggle constraint. We further tuned the classifier post-fit by picking the best threshold to convert model probabilities into binary classifications.

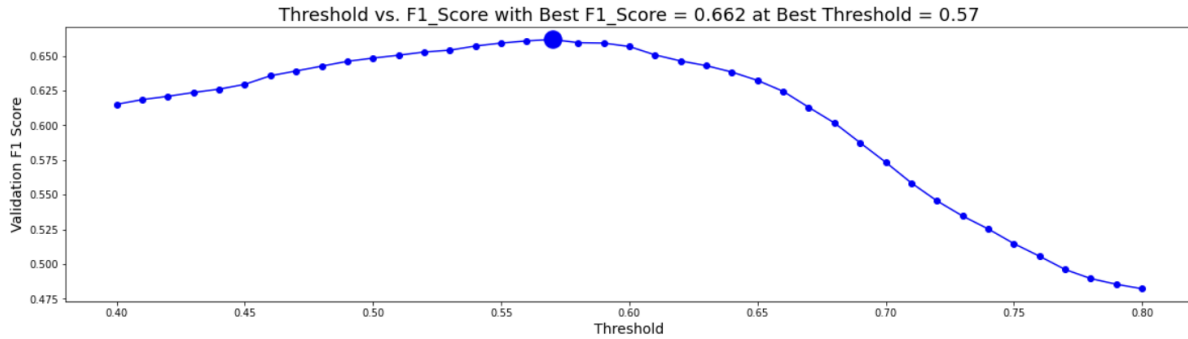


Figure 2

As seen in *Figure 2*, the best threshold to convert these probabilities was 0.57. This means that if the model was less than 57% confident that the user would answer the question correctly, it would predict that the question would be answered incorrectly. If the model was more than 57% confident, then the model would predict that the question would be answered correctly. This yielded the highest macro F-1 score of 0.662. The individual F-1 score per question can be seen in *Figure 3*.

```
When using optimal threshold...
Q0: F1 = 0.5622858984590828
Q1: F1 = 0.4962598845907245
Q2: F1 = 0.48695189493796404
Q3: F1 = 0.5466915895928776
Q4: F1 = 0.3187744170562127
Q5: F1 = 0.5797551828188873
Q6: F1 = 0.5420276969683049
Q7: F1 = 0.5273496640113665
Q8: F1 = 0.5957372217943573
Q9: F1 = 0.3396085893178367
Q10: F1 = 0.5809513816404515
Q11: F1 = 0.48150320806599456
Q12: F1 = 0.4262414800389484
Q13: F1 = 0.5813620357688156
Q14: F1 = 0.36229997451428125
Q15: F1 = 0.46991702472686814
Q16: F1 = 0.46584696361411526
Q17: F1 = 0.48604448320976884
==> Overall F1 = 0.6618563739245985
```

Figure 3

Conclusions

In conclusion, we were able to build a reliable model based on sklearn's framework's Random Tree model to predict a user's performance during game-based learning in real time. Moreover, we were able to establish a successful data processing mechanism for a data set of over 26M observations while maintaining the resource requirement established by Kaggle using only the techniques that were introduced in this course. We were able to achieve a final macro F-1 score of 0.656 on the hidden Kaggle test set. As seen in the historic submission scores in *Figure 4*, we were able to increase the accuracy of our model by eliminating the event_name and room_fqid columns as mentioned in our *Data Processing* section. It appears that dropping the additional columns created a more accurate model.





Submission and Description		Public Score ⓘ	Select
	Random Forest WORKING - Version 7 Succeeded · Tal Kronrod · 5d ago · Notebook Random Forest Version 7	0.656	
	Random Forest WORKING - Version 5 Succeeded · Tal Kronrod · 6d ago · NiTaCu Random Forest Version 5	0.653	

Figure 4

Takeaways

One of the major takeaways from this project is that sophisticated feature selection and engineering techniques are essential to create well performing machine learning models. This is especially important on high-dimensional and highly varied datasets. By carefully analyzing each columns in the dataset to select which columns should be dropped or modified, we can create a more useful dataset. Feature engineering can also prevent overfitting, which is how the score increase in *Figure 4* can be explained.

Additionally, another takeaway from this project lies on the constraints that come from working with extremely large data sets. Up to this point in the class, we handled relatively manageable data sets, which meant that runtime and memory constraints have not been a problem before. For the first time in the course, we realized that we did not have unlimited CPU power, which caused us to run into immediately after starting the project (since loading a 26M data set obviously

yielded an error). This, however, gave a very valuable lesson about resource management of a computer and adapting to the available capacity of a machine.

Building on the last takeaway, we also learned that we don't need to use the whole training dataset to create a model with similar results to models trained on the whole dataset. By using sophisticated feature selection and engineering techniques we are able to create a higher quality, more compact dataset. This dataset becomes more useful for training models.

From a more social point of view, this project showed that skills can be generalizable, meaning that they can be applied in various contexts. We often don't associate gameplay with academic performance, and some may even view gamers as lacking in important academic and working skills. However, we have found and proved that there is a link between gaming and certain academic performances. This discovery in the project is noteworthy and a good takeaway because it suggests that students are able to develop and apply skills in non-academic contexts, rather than relying solely on traditional methods of learning.

Next Steps

Even though our final model gives us a relatively high F1 score with respect to alternative models we went over in this class, we know that there is still room for improvement: using advanced boosting models such as TensorFlow Decision Trees, XGBoost model, or LightGBM would yield a significantly greater prediction. The reason for that lies in the higher complexity of gradient boosted models as well as the added feature of GPU implementation which would allow us to execute the models both faster and with smaller memory space. Due to that, we could be able to train the more complex models with the whole data set (as opposed to one tenth of it).

We also attempted to use grid search on our random forest model, but we were unable search all possible combinations we wanted to because the training time required exceeded the 12 hour runtime limitation of Kaggle notebooks. So in our model, we kept the hyperparameters that performed the best in our search ability. In the future, given we would have more computing resources and power, we would execute this method to its completion.

One major limitation we had is the problems with Kaggle's session interface. We tried to use the python pickle package to save models between sessions. However, every time a Kaggle session is ended, the working directory gets reset. This caused the saved models to be deleted after every run, causing the process to be redundant. This hindered our progress significantly since having to re-train the models every new session significantly increased the amount of time we had to wait to make new changes. In the future, if we had an improved or different interface, google colab

for example, we would be able to save the trained models and making it easier for us to try different models and parameters.

Contributions

Each team member contributed equally to the project. Everyone contributed code segments, additions, and edits to the Notebook. Everyone also contributed to writing $\sim\frac{1}{3}$ of the report.

References

Figures

Figure 1 -- A table showing the column names, and their corresponding descriptions, data types, and universes.

Figure 2 -- A diagram showing the relationship between the model decision threshold and the F-1 score.

Figure 3 -- An image showing the individual F-1 scores for each of the 18 questions and the overall F-1 score.

Figure 4 -- An image showing the historic final F-1 scores for each submission.