# Ludum Dare Classification (w/ LightGBM + Image & Text Processing)

Jason Amri (*~~123456789~~*)

School of CS, University of Waterloo

*~~email@email.com~~*

Prepared for CS480, Fall 2022

## Abstract

 Ludum Dare is an online game development competition, where submissions receive a rating from 1-5 stars. This project aims to predict the rating of a project using a set of 32 diverse parameters. To achieve this, a model was developed centred around [LightGBM](https://lightgbm.readthedocs.io/) as a multiclass classifier. In addition, images were assessed for their number of colours, and the project name and description were given a classification separately using [PyTorch](https://pytorch.org/) text classification. With tuning, this approach achieved a 94.15% accuracy, placing 1st on the class leaderboard (at the time of writing). It also demonstrated the value of practical machine learning.

## Introduction

 Competitions usually entail a level of suspense or delayed result – you don’t know who’s going to win until they do. However, as access to data and the abilities of machine learning (ML) have grown, this is often no longer the case. Take for example Google’s Prediction APIs, which are sometimes presented alongside live scores from sports games to predict, with a high degree of accuracy, the outcome of the game (UJ et al., 2018). As ML technology becomes more accessible, and with the background provided by CS480, individuals can use similar prediction tools at home. This report explores an application of such prediction to data from the Ludum Dare competition.

## Problem Formulation

### Step 1: Understanding the data (step-1.py)

 To begin, it was important to understand the fields at hand and identify which would be useful for the classification problem. To do this, the data was grouped by type (Text, Date, Number (Average, Rank, Binary, Other), URL). Next, the data was cleaned. Date fields were converted to UNIX-timestamps, links and tags were counted, and ranks were normalized by category and competition type. Similar normalization was attempted on averages and interaction data but had no positive impact. Finally, the correlation between each field and its actual label was assessed (see [Table 1](#_Table_1:_Fields) for a summary). Fields with the highest correlation were num\_comments, ratings\_received, the rank fields, and the average fields. Fields with the lowest correlation were the date fields (which weren’t used in the final model), and the competition and category fields (which were identified as categorical variables). The fields with high correlation became prime candidates for inclusion in the final model. However, text fields and images remained unused, motivating Steps 2 and 3.

### Step 2: Using Images (step-2b.py)

 The first attempted approach was to use PyTorch Image Classification to classify the images separately. However, after working through many errors and completing the code, this proved computationally infeasible given the time constraints of the competition. This code is included in step-2a.py. Instead, a simpler hypothesis and approach were taken. It was theorized that “nicer” images correlated with a better-quality game. As a proxy for the “niceness” of an image, the number of colours in the image was computed using the Python Imaging Library (via pillow). This measure yielded a 0.200 correlation with the labels.

### Step 3: Using Text (step-3.py)

 Again, PyTorch Text Classification was attempted. However, in this case, runtimes were quite reasonable. Minimal adaptions were made from the example tutorial. Key parameters, including embedding size, learning rate, batch size, and epoch were tuned, which mostly resulted in unchanged values. The outcome was a reasonable accuracy (around 45%, given randomness) on a 5% validation set.

### Step 4: Selecting a model and putting it all together

With the data well understood, images processed, and text classified, everything was ready to be included in a final model. This problem is a multiclass classification problem across 6 classes: 0 (for projects with too few votes), and 1-5 Stars. Notably, it is also a competition-style problem, with a limited and uncommon dataset, a fixed time frame, and a limited number of submissions. A review of articles on the best tools for this situation (namely Brionne, 2021 and Ramavtar, 2022) showed that XGBoost or LightGBM (LGBM) were popular choices. These are both gradient boosting algorithms, which work similarly to the boosting algorithms seen in CS480 (eg. AdaBoost) but optimize based on a differentiable loss function (Masui, 2022). Sentiment on a head-to-head comparison showed that LGBM would be the best choice for a dataset of this size: >10000 rows, high dimension (Kasturi, 2021).

To make predictions, the classifiers in steps 2 and 3 are run to generate new columns in the testing and training datasets. Then the data is loaded and normalized, as explored in step 1. Then LGBM Cross Validation is performed to determine to the optimal number of rounds to minimize the training error. The log loss and error rate, with optimal rounds for both, are also reported. Then a final model is trained and used to predict labels. Results are stored in a .csv for upload. Parameters like the learning rate (~0.004) were tuned with a manual binary search over training errors.

## Main Results

 This model was able to achieve a training error of just 3.58% and classified elements in the public portion (30%) of the test set with 94.15% accuracy, topping the leaderboard (at the time of writing).

## Conclusions

This project demonstrated the power of LGBM for classifying mid-sized datasets. LGBM was remarkable not only for its accuracy but also for its speed and ease of use. While there are few high-quality examples due to its relative newness and niche, the documentation itself is well done, and the barrier to achieving good results is very low.

As more work is put into packaging bleeding-edge Machine Learning concepts into user-friendly libraries, the accessibility of ML, and the opportunities to realize its value will grow. This project demonstrated a frivolous application, which at best makes for a strong final assignment submission, and at worst ruined the suspense of the Ludum Dare competition. However, one can imagine similar principles being applied to accelerate the classification of malware threats, genome sequences, fraudulent financial transactions, and more. However, the real power in making this technology accessible isn’t in large enterprises, but in classifying small, unprofitable, yet meaningful problems, even if that meaning is only for a few people.

## Execution Guidance

Files are labelled to reflect steps in [Problem Formation](#_Problem_Formulation).

Dependences (version in use):

pandas (1.5.2), numpy (1.23.5), pillow (9.3.0), torch (1.13.0), torchtext (0.14.0), lightgbm (3.3.3)

To replicate results:

1. Copy Python Code, test.csv, train.csv, and thumbnails folder into a directory
2. Run code in the following order:
	1. step-2b.py
	2. step-3.py (run twice with PARAM=”name” and PARAM=”description”)
	3. step-4.py
3. Assess predictions.csv for predicted results

## Table 1: Fields and Correlations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Type | Use | Cleaning | Correlation |
| id | Number | Images | Note: Unused in model |
| name | Text | NLP |  |  |
| slug | URL | Unused |  |  |
| path | URL | Unused |  |  |
| description | Text | NLP |  |  |
| version | Number | Unused | Unknown purpose, ignored |
| theme-rank | Rank |  | Scale to 0..1 | 0.889030226 |
| innovation-rank | Rank |  | Scale to 0..1 | 0.880433165 |
| fun-rank | Rank |  | Scale to 0..1 | 0.86856225 |
| graphics-rank | Rank |  | Scale to 0..1 | 0.823907886 |
| mood-rank | Rank |  | Scale to 0..1 | 0.808277056 |
| humor-rank | Rank |  | Scale to 0..1 | 0.703132337 |
| num-comments | Number |  |  | 0.636340664 |
| ratings-received | Number |  |  | 0.635233393 |
| audio-rank | Rank |  | Scale to 0..1 | 0.616527158 |
| fun-average | Average |  |  | 0.613710593 |
| innovation-average | Average |  |  | 0.593464699 |
| theme-average | Average |  |  | 0.572742859 |
| ratings-given | Number | Hail Mary |  | 0.562814136 |
| mood-average | Average |  |  | 0.562106753 |
| graphics-average | Average |  |  | 0.539841386 |
| feedback-karma | Number | Hail Mary |  | 0.494193026 |
| humor-average | Average |  |  | 0.428440386 |
| audio-average | Average |  |  | 0.39932131 |
| link-tags | Text |  | Count elements | 0.265618923 |
| links | URL |  | Count elements | 0.26299434 |
| num-colors | Number | From Step 3 |  | 0.199794147 |
| prev-games | Number | Hail Mary |  | 0.179824073 |
| num-authors | Number | Hail Mary |  | 0.124851779 |
| modified | Date | Unused |  | 0.087488727 |
| competition-num | Number | Categorical | 0.082361376 |
| published | Date | Unused |  | 0.078622758 |
| category | Binary | Categorical | Text to Binary | -0.046943153 |

## References

Brionne, A. L. D. (2021, February 5). Kaggle Competition: Multi class classification on image and Data. Retrieved December 16, 2022, from https://www.linkedin.com/pulse/kaggle-competition-multi-class-classification-image-alexandra/

Kasturi, S. N. (2021, September 26). Lightgbm vs XGBOOST: Which algorithm win the race! Medium. Retrieved December 16, 2022, from https://towardsdatascience.com/lightgbm-vs-xgboost-which-algorithm-win-the-race-1ff7dd4917d

Masui, T. (2022, February 12). All you need to know about gradient boosting algorithm − Part 1. regression. Medium. Retrieved December 16, 2022, from https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502

Ramavtar. (2022, March 17). Microsoft Malware Kaggle Challenge: Multiclass Classification and feature enginnering. Medium. Retrieved December 16, 2022, from https://medium.com/@smartrambit/microsoft-malware-kaggle-challenge-multiclass-classification-and-feature-enginnering-a719d6d8661

UJ, U., PJ, A., &amp; DN, S. (2018). Predictive Analysis of Sports Data using Google Prediction API. Research India Publications. Retrieved December 16, 2022, from https://www.ripublication.com/ijaer18/ijaerv13n5\_96.pdf