Machine Learning Project Interim Report: Mechanisms of Action (MoA) Prediction

Malte Meng, 2020280441 (Leader)

Khang Hui Chua, 2020280442

Kai Wen Yoke, 2020280598

Abstract

This interim report documents the methods we have tried so far for the MoA Kaggle competition (`https://www.kaggle.com/c/lish-moa/`). This is a multi-label classification task, where we are to predict the probabilities of each of 206 mechanisms of action (MoAs) of a drug sample, given tabular data on gene expression and cell viability levels of cell samples treated with that drug sample. We tried various feature engineering methods, model architectures, training strategies and evaluation methods, and have currently achieved 109th position out of 4081 teams. This was achieved by blending three different models together, with the best individual model achieving 500th position. With ten days left for the competition (ending 30 Nov), there are still several architectures we would like to try, and if they perform well individually they will be added into our model blend.

Preprint. Under review.

Contents

1 Overview of the Kaggle competition: MoA Prediction

For our Machine Learning project, we chose a currently active Kaggle competition "Mechanisms of Action (MoA) Prediction" that ends on 30th November: `https://www.kaggle.com/c/lish-moa/`.

1.1 Problem Statement

Given data on gene expression and cell viability levels of cell samples in response to being treated by a drug sample at a particular dosage and for a particular time duration, the task is to predict the MoA of the drug. This is a multi-label classification task, as each drug sample can be associated with more than one MoA. The accuracy of solutions will be evaluated on the average value of the logarithmic loss function applied to each drug-MoA pair.

1.2 Dataset overview

We have 23814 data samples in the training set. The entire dataset includes roughly 5000 unique drugs, with each drug profiled multiple times at different dosages and treatment times.

1.2.1 Raw features

The raw data features are shown in Figure 1.

	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4	g-5	...	c-90
0	id_000644bb2	trt_cp	24	D1	1.0620	0.5577	-0.2479	-0.6208	-0.1944	-1.0120	...	0.2862
1	id_000779bfc	trt_cp	72	D1	0.0743	0.4087	0.2991	0.0604	1.0190	0.5207	...	-0.4265
2	id_000a6266a	trt_cp	48	D1	0.6280	0.5817	1.5540	-0.0764	-0.0323	1.2390	...	-0.7250
3	id_0015fd391	trt_cp	48	D1	-0.5138	-0.2491	-0.2656	0.5288	4.0620	-0.8095	...	-2.0990
4	id_001626bd3	trt_cp	72	D2	-0.3254	-0.4009	0.9700	0.6919	1.4180	-0.8244	...	0.0042

	...	c-90	c-91	c-92	c-93	c-94	c-95	c-96	c-97	c-98	c-99
)	...	0.2862	0.2584	0.8076	0.5523	-0.1912	0.6584	-0.3981	0.2139	0.3801	0.4176
	...	-0.4265	0.7543	0.4708	0.0230	0.2957	0.4899	0.1522	0.1241	0.6077	0.7371
	...	-0.7250	-0.6297	0.6103	0.0223	-1.3240	-0.3174	-0.6417	-0.2187	-1.4080	0.6931
;	...	-2.0990	-0.6441	-5.6300	-1.3780	-0.8632	-1.2880	-1.6210	-0.8784	-0.3876	-0.8154
ł	...	0.0042	0.0048	0.6670	1.0690	0.5523	-0.3031	0.1094	0.2885	-0.3786	0.7125

Figure 1: Raw data features

There are 772 **gene expression features**, with each gene feature representing the expression level of a particular gene, and 100 **cell viability features**, with each cell viability feature representing the viability of one cell line, meaning that there are 100 cell lines in this dataset. The actual names of the gene features and cell types are hidden, so a hand-crafted approach would not be possible. The **cp_type** attribute indicates whether a data sample is treated with a drug (`trt_cp`) or a control (`ctl_vehicle`), and a control sample would have no associated MoAs, but note that there are data samples which are not controls and also do not belong to any of the 206 MoAs that are our target labels tested during scoring. The **cp_dose** is a binary feature and comprises two levels of dosage D1 or D2, and **cp_time** is a categorical feature comprising three treatment durations (24, 48 or 72 hours).

1.2.2 Target labels

The frequency of each of the 206 MoA target labels across the train dataset are shown in Figure 2.

```
atp-sensitive_potassium_channel_antagonist      1
erbb2_inhibitor                                 1
diuretic                                        6
autotaxin_inhibitor                             6
protein_phosphatase_inhibitor                   6
                                              ...
serotonin_receptor_antagonist                 404
dopamine_receptor_antagonist                  424
cyclooxygenase_inhibitor                      435
proteasome_inhibitor                          726
nfkb_inhibitor                                832
Length: 206, dtype: object
```

Figure 2: Frequency of each of the 206 MoA target labels in training data

The classes in the dataset are highly imbalanced as many of the target MoAs have only one entry each while some have hundreds. The train dataset also contains an additional non-scored 402 target MoAs for each data sample, meaning that we do not have to predict these auxiliary MoAs for the test dataset.

2 Methods and experimentation

Much useful information has been discussed on the Kaggle competition discussion boards. Many public notebooks have also been shared. We leverage on these resources for our experimentation process. We found feature engineering, model architectures, training strategies, evaluation methods and model blending having the greatest influence over our performance on the leaderboard, and below we will discuss further the different methods we tried.

2.1 Feature Engineering

Preprocessing and feature engineering is essential in this competition where tabular data is involved, unlike unstructured data such as images, audio and text, where deep learning can directly be applied to automatically extract features. This is because the given tabular data has heterogeneous features, some categorical, some numerical and also with different units and scaling. Furthermore, the features are also sparse with sometimes little variation in a column of the table. These lead to features in a high-dimensional space that is not dense and continuous, which makes it difficult for us to exploit NNs which have proven to be one of the most successful ML models. Below, we discuss feature preprocessing methods to make the data more amenable to NNs.

2.1.1 Raw features

The raw features of the dataset include categorical and numerical ones. Categorical features are **cp_type** (drug or control sample), **cp_dose** (low or high dosage) and **cp_time** (24, 48 or 72 hours). Numerical ones are the 772 **gene expression features** and the 1000 **cell viability features**.

2.1.2 One hot encoding of categorical features

The categorical features **cp_dose** and **cp_time** are one-hot encoded, while only data samples with **cp_dose** == `trt_cp` are retained. The control group is removed because when **cp_dose** == `ctl_vehicle`, the cell samples are not treated with any drugs so there are no associated MoA target labels.

2.1.3 Standardisation of numerical features

The raw gene expression and cell viability data given by the organisers already went through quantile normalisation to standardize the shape of the distributions on each plate. However, the data is also clipped at -10 and 10, resulting in spikes at these two points (see Figure 3). There are also outliers that occur far from the mean of the normal distributions. Therefore, we still apply `QuantileTransformer` from `scikit-learn` with 100 quantiles to transform the given raw data into normal distributions with range from -1 to 1. Furthermore, it has been empirically shown that machine learning models work better with standardized normally distributed data.

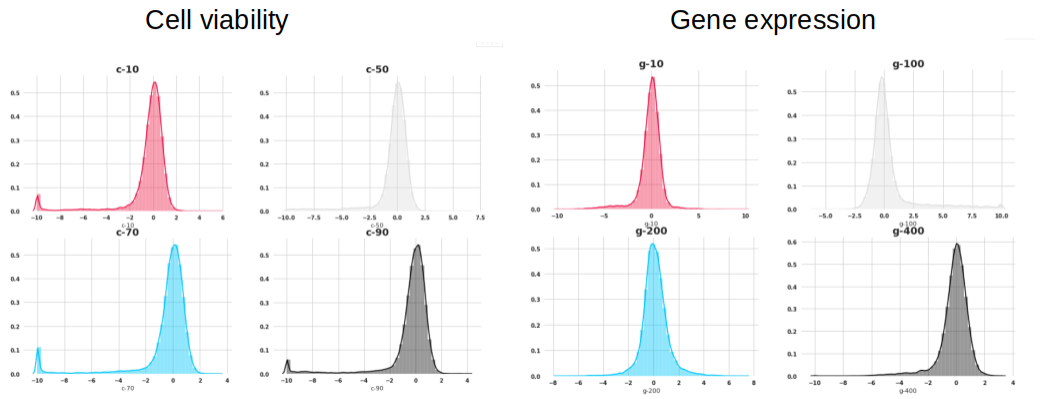

Figure 3: Distribution of a few cell viability and gene expression data features clipped at -10 to 10

2.1.4 Dimensionality reduction of numerical features

Data exploration reveals many correlations between different gene expression features and cell viability features (see Figure 4).

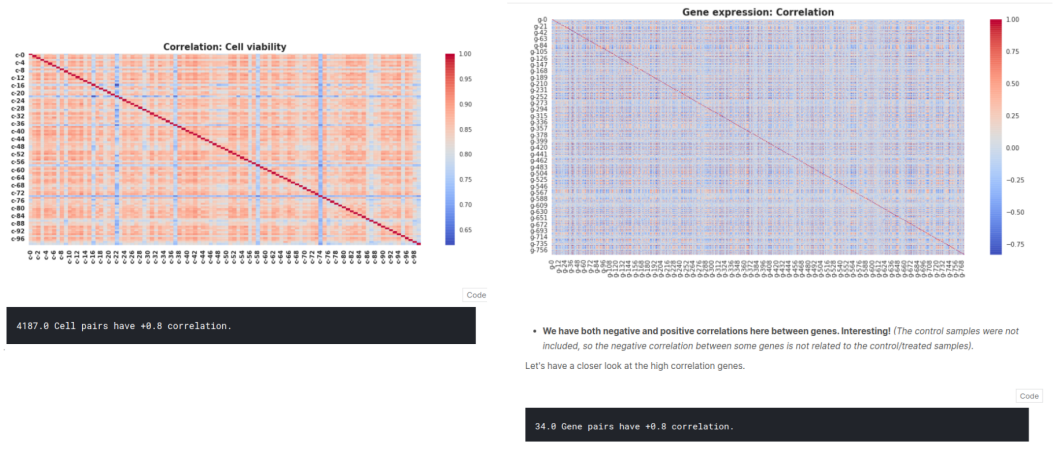

Figure 4: Correlations among cell viability and gene expression features

This suggests that many features are redundant and thus the dimensionality of features can be reduced. We achieve this by using PCA, variance thresholding, Kmeans clustering, and statistics-based aggregation.

1. **PCA**: We applied `PCA` from `scikit-learn` with 600 components for gene expression features and 50 components for cell viability features. These are the number of components required to explain 95% of the cumulative variance of the gene expression and cell viability features respectively. PCA works by summarising the correlations between different features into fewer principal components.

2. **Variance thresholding**: We remove numerical features that have variance less than 0.87 (arbitrary) before applying Kmeans clustering and statistics-based aggregation. The rationale is that if many data samples share similar values for a certain feature, then the feature is not going to be very good at distinguishing between the different data samples.

3. **Kmeans clustering**: We applied `KMeans` from `scikit-learn` to cluster the data samples into 22 clusters based on their gene expression features, and 5 clusters based on their cell viability features. The gene and cell cluster IDs of each data sample is then one-hot encoded. The number of clusters were arbitrarily chosen by visualising the clusters using `TSNE` (t-distributed Stochastic Neighbor Embedding) from `scikit-learn` as seen in Figure 5.

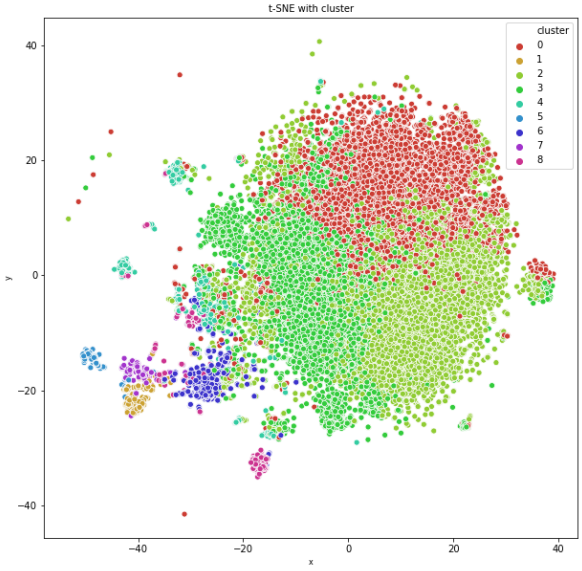

Figure 5: t-SNE visualisation plot of gene expression features with 9 clusters

4. **Statistics-based aggregation**: We also created more features from the raw gene and cell features based on statistics, such as sum, mean, standard deviation, kurtosis, skew and square across all gene features and across all cell features respectively, as a way of combining all gene features into a few representative statistics and all cell features into a few representative statistics.

2.2 Model experimentation

Most of the more successful models use neural architectures. Nevertheless, the idea is to get as diverse models as possible to boost our final ensemble.

2.2.1 Fully connected networks

We tried implemnting simple fully connected networks with `pytorch` experimenting with various number of hidden layers, number of neurons, batch normalisation, dropout rates, weight and bias initialisations, activation functions (`ReLU, LeakyReLU, Swish`) and loss functions.

2.2.2 TabNet

Other than vanilla fully connected networks, we also tried more novel architectures such as TabNet (1) using the `pytorch_tabnet` library which is designed specifically for tabular data with an attention mechanism that learns a sparse mask over input features.

2.2.3 Other model architectures

We are also planning to try CNN architectures and also other non-NN architectures such as K-NN. Nevertheless, so far the best performing single model is the simple fully connected network with architecture shown in Figure 6, with batch normalisation, dropout, `kaiming_normal` weight initialisation, bias initialisation for the final layer corresponding to the frequency of each target label in the training set, `ReLU` activation and `BCEWithLogitsLoss` with label smoothing. This combination of parameters helps to speed up convergence and regularise the training process, especially when considering the label imabalance for the training set.

```
self.hidden_size = [1500, 1250, 1000, 750]
self.dropout_value = [0.5, 0.35, 0.3, 0.25]

self.batch_norm1 = nn.BatchNorm1d(num_features)
self.dense1 = nn.Linear(num_features, self.hidden_size[0])

self.batch_norm2 = nn.BatchNorm1d(self.hidden_size[0])
self.dropout2 = nn.Dropout(self.dropout_value[0])
self.dense2 = nn.Linear(self.hidden_size[0], self.hidden_size[1])

self.batch_norm3 = nn.BatchNorm1d(self.hidden_size[1])
self.dropout3 = nn.Dropout(self.dropout_value[1])
self.dense3 = nn.Linear(self.hidden_size[1], self.hidden_size[2])

self.batch_norm4 = nn.BatchNorm1d(self.hidden_size[2])
self.dropout4 = nn.Dropout(self.dropout_value[2])
self.dense4 = nn.Linear(self.hidden_size[2], self.hidden_size[3])

self.batch_norm5 = nn.BatchNorm1d(self.hidden_size[3])
self.dropout5 = nn.Dropout(self.dropout_value[3])
self.dense5 = nn.utils.weight_norm(nn.Linear(self.hidden_size[3], num_targets))
```

Figure 6: Best fully connected network architecture currently

2.3 Training strategies involving unscored target labels

Other than the conventional way of training the models on the training data and using the trained model to predict the labels of the test data, we also tried other training strategies involving the use of the 402 unscored target labels. This includes using the unscored target labels as meta-features and for transfer learning. Unscored target labels are useful because they share significant correlation with scored target labels, as seen in Figure 7, therefore having information about the unscored target labels could help us predict the scored labels of the test data samples.

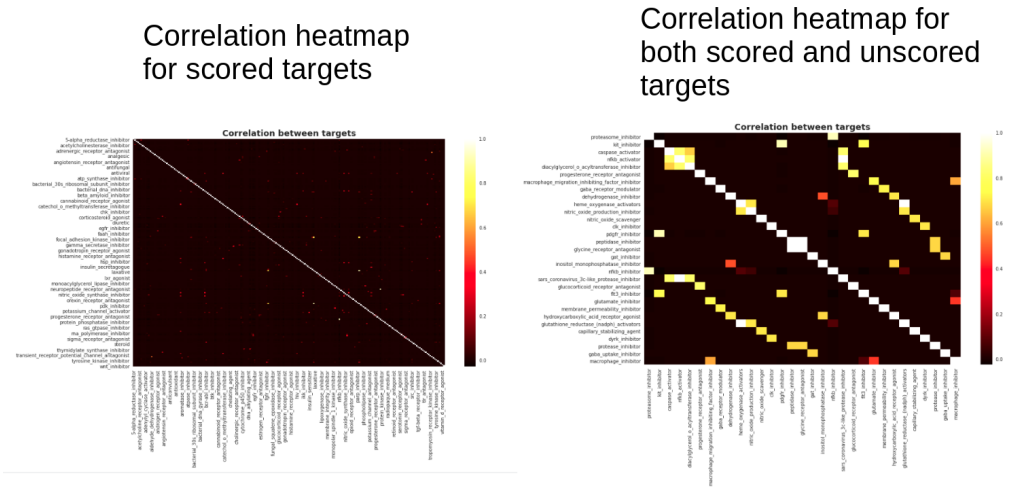

Figure 7: Correlation heatmap of scored target labels only VS correlation heatmap of both scored and unscored target labels combined

2.3.1 Unscored target labels as meta-features

The idea is to first train a fully connected network to predict the 402 unscored target labels of the training dataset, then use the trained network (say `model_1`) to predict the unscored target labels of the test dataset. Note that `model_1` has 402 neurons in the last layer corresponding to the unscored target labels. Then, we train a second model `model_2`, which has 206 neurons in the last layer corresponding to the scored target labels. In addition, `model_2` also has the unscored target labels predicted by `model_1` as input. We call these predicted unscored target labels 'meta-features' that help improve model performance.

2.3.2 Transfer learning: Unscored target labels for pretraining

The idea is we first pretrain a fully connected network with 206+402=608 neurons in the last layer by utilising both the scored and unscored target labels. Then, we freeze the weights in the top layers, replace the last layer with a layer of only 206 neurons, and train the same model again with only the weights in the last layer being tuned, and only the 206 scored target labels involved. This method has also been quite successful.

2.4 CV Evaluation

In the competition, submissions are scored by the log loss:

$$score = -\frac{1}{M}\sum_{m=1}^{M}\frac{1}{N}\sum_{i=1}^{N}[y_{i,m}\log(\hat{y}_{i,m}) + (1-y_{i,m})\log(1-\hat{y}_{i,m})]$$

However, since we are only limited to 3 submissions a day, we also need to devise a way to evaluate our models ourselves, hopefully with a score matching to that of the leaderboard, and we achieve this using `MultilabelStratifiedKFold` from `scikit-learn`. This helps us to average our evaluation results over a number of folds (we found 5-7 suitable) for more accurate evaluation. We also set several seeds (3) and average our results across different seeds for better accuracy.

2.5 Model blending

Model blending is done in the last few days of the competition to combine the predictions of several models together. So far, we simply tried averaging the results of our three best predictions together

(TabNet, fully-connected network with meta-features, fully-connected network with transfer learning) and that has resulted in a surprising boost in our leaderboard position (jumped from 500th to 80th position). We will be trying more sophisticated model blending methods closer to the deadline.

3 Results of best submissions currently

Results of best submissions so far are shown in Figure 8.

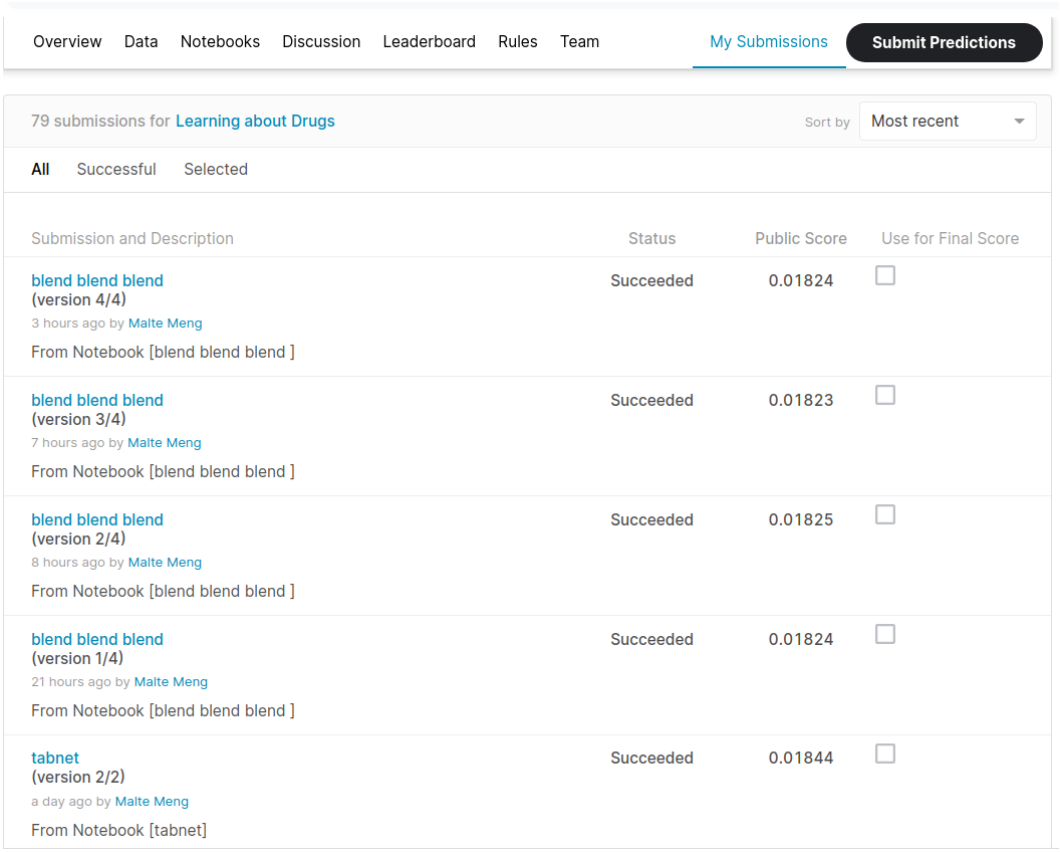

Figure 8: Results of submissions so far

Evidence of leaderboard position on public dataset as of 21 Nov is shown in Figure 9.

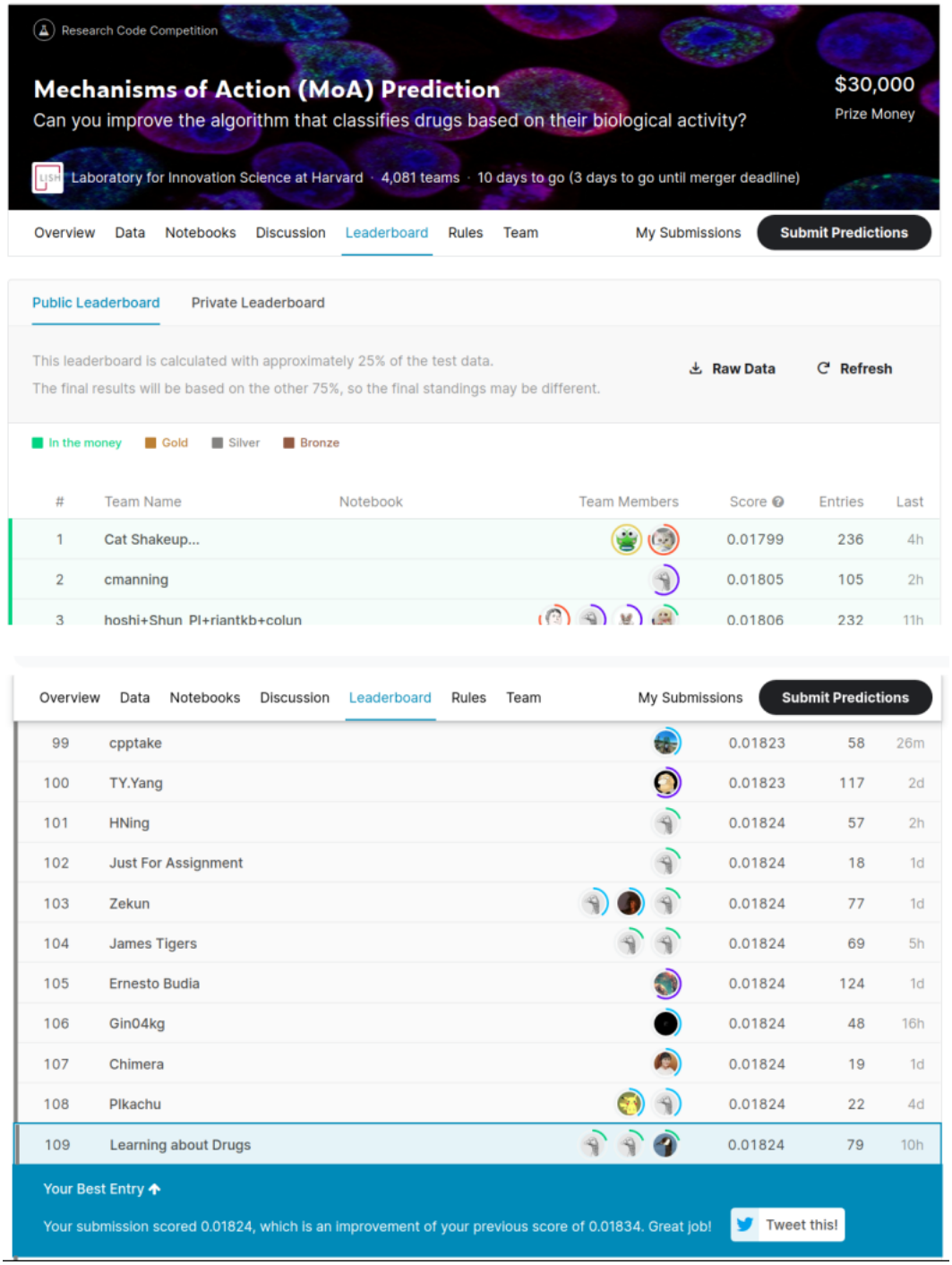

Figure 9: Leaderboard position as of 22 Nov

4 Ideas to try in next ten days

4.1 Model architectures

1. CNN-based architecture
2. K-NN

3. Autoencoder

4.2 Pseudo-labelling

Once we have a confident model prediction score for the public test dataset, we can include the public test data into our training set with the confidently predicted labels as the pseudo-ground truth, hopefully further boosting our scores.

4.3 Model blending

We could try more sophisticated methods of model blending, such as weighted averages and using optimisation to find the optimum weights and perhaps using a neural network with the predictions of several models as input and learning the best way to weight the different model predictions.

References

[1] S. O. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," 2020.