AI 6102 Term paper Ongoing Kaggle competition

Mechanisms of Action (MoA) Prediction

Sun Guangbiao G2003483F Peng Minyi G2001910L Zhao Meng G2001852F

Outline

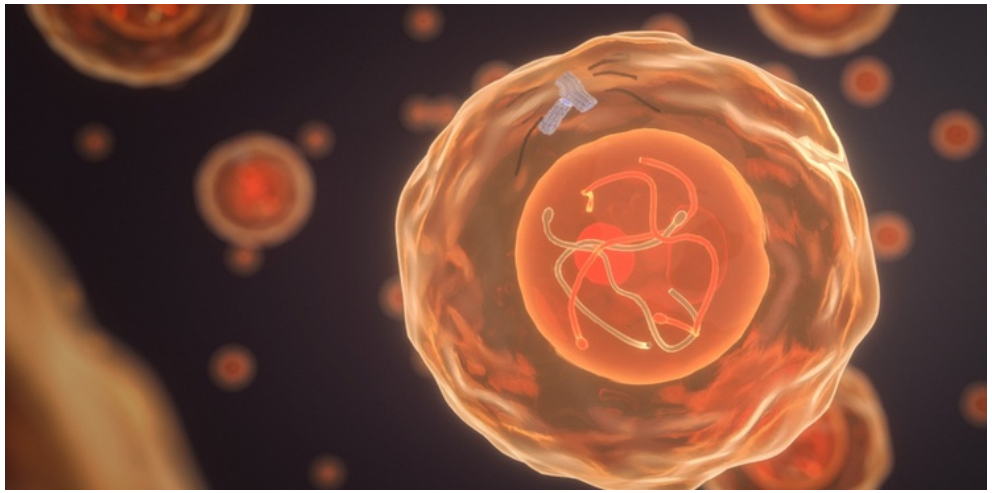
Part 1. Problem statement of the task

Part 2. Challenges of the problem and solutions

Part 3. Experiments and results

Part 4. Further research

Part 5. Conclusion



Part1. Problem statement of the task

Part 1.Problem statement of the task

Background

MoA The term **mechanism of action** means the biochemical interactions through which a drug generates its pharmacological effect. Scientists know many MoAs of drugs, for example, an antidepressant may have a selective serotonin reuptake inhibitor, which affects the brain serotonin level.

The Connectivity Map, a project within the Broad Institute of MIT and Harvard, the Laboratory for Innovation Science at Harvard (LISH), and the NIH Common Funds Library of Integrated Network-Based Cellular Signatures (LINCS), present this challenge with the goal of advancing drug development through improvements to MoA prediction algorithms

Part 1.Problem statement of the task

Basic Task

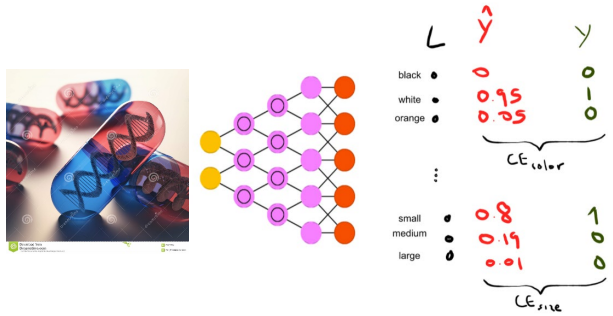
Basic task

The aim of this challenge is to “classify drugs based on their biological activity”.

Pharmaceutical drug discovery aims to identify certain proteins that are associated with a specific disease, and then to develop molecules that can target those proteins. The MoA of a molecule encodes its biological activity. Our dataset describes the responses of 100 different types of human cells to various drugs. Those response patterns will be used to classify the MoA response

Multi-label classification

This is a multi-label classification problem. Drugs can have multiple MoA annotations which describe binary responses from different cell types in different ways. So this is a multi-label classification task as such the rows (i.e. drug samples) can have multiple MoA’s (i.e. more than one target class can be active)



Multi-label classification task

Part 1.Problem statement of the task

Evaluation

Binary Cross-Entropy loss

This is a multi-label binary classification problem, Based on the MoA annotations, the accuracy of solutions will be evaluated on the average value of the logarithmic loss function(Binary Cross-Entropy loss) applied to each drug-MoA annotation pair. and metric used for the evaluation is mean columnwise log loss. For every row, a probability that the sample had a positive response for each target, has to be predicted. For N rows and M targets, there will be N×M predictions. Submissions are scored by the log loss:

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

Where

N is the number of rows (i=1,⋯,N),
M is the number of targets (m=1,⋯,M),
$\hat{y}_{i,m}$ is the predicted probability of the ith row and mth target
$y_{i,m}$ is the ground truth of the ith row and mth target (1 for a positive response, 0 otherwise).

The equation to calculate the output score

Part 1.Problem statement of the task

Dataset

Glimpse

The data is based on a new technology that measures simultaneously (within the same samples) human cells' responses to drugs in a pool of 100 different cell types (thus solving the problem of identifying ex-ante, which cell types are better suited for a given drug). In addition, We have access to MoA annotations for more than 5,000 drugs in this dataset

Components

The dataset consists of different features of **gene expression data**, and **cell viability data** as well as multiple targets of mechanism of action (MoA)

	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4	g-5	g-6
1	id_000644bb2	trt_cp	24	D1	1.062	0.5577	-0.2479	-0.6208	-0.1944	-1.012	-1.022
2	id_000779bfc	trt_cp	72	D1	0.0743	0.4087	0.2991	0.0604	1.019	0.5207	0.2341
3	id_000a6266a	trt_cp	48	D1	0.628	0.5817	1.554	-0.0764	-0.0323	1.239	0.1715
4	id_0015fd391	trt_cp	48	D1	-0.5138	-0.2491	-0.2656	0.5288	4.062	-0.8095	-1.959
5	id_001626bd3	trt_cp	72	D2	-0.3254	-0.4009	0.97	0.6919	1.418	-0.8244	-0.28
6	id_001762a82	trt_cp	24	D1	-0.6111	0.2941	-0.9901	0.2277	1.281	0.5203	0.0543
7	id_001bd861f	trt_cp	24	D2	2.044	1.7	-1.539	5.944	-2.167	-4.036	3.695
8	id_0020d0484	trt_cp	48	D1	0.2711	0.5133	-0.1327	2.595	0.698	0.5846	-0.2633
9	id_00224bf20	trt_cp	48	D1	-0.3014	0.5545	-0.2576	-0.139	-0.6487	-0.6057	-0.7549
10	id_0023f063e	trt_cp	48	D2	-0.063	0.2564	-0.5279	-0.2541	-0.0182	-1.537	-0.218

	sig_id	5-alpha_reductase_inhibitor	11-beta-hsd1_inhibitor	acat_inhibitor	acetylcholine_receptor_agonist	acetylcholi
1	id_000644bb2	0	0	0	0	0
2	id_000779bfc	0	0	0	0	0
3	id_000a6266a	0	0	0	0	0
4	id_0015fd391	0	0	0	0	0
5	id_001626bd3	0	0	0	0	0
6	id_001762a82	0	0	0	0	0
7	id_001bd861f	0	0	0	0	0
8	id_0020d0484	0	0	0	0	0
9	id_00224bf20	0	0	0	0	0
10	id_0023f063e	0	0	0	0	0

Example of the dataset and targets

Part 1.Problem statement of the task

Training Dataset

Features

Features for the training set. Features **g-** signify **gene expression** data, and **c-** signify **cell viability data**. **cp_type** indicates samples treated with a compound (cp_vehicle) or with a control perturbation (ctrl_vehicle); control perturbations have no MoAs; cp_time and cp_dose indicate treatment duration (24, 48, 72 hours) and dose (high or low).

	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4	g-5
0	id_000644bb2	trt_cp	24	D1	1.0620	0.5577	-0.2479	-0.6208	-0.1944	-1.0
1	id_000779bfc	trt_cp	72	D1	0.0743	0.4087	0.2991	0.0604	1.0190	0.5
2	id_000a6266a	trt_cp	48	D1	0.6280	0.5817	1.5540	-0.0764	-0.0323	1.2
3	id_0015fd391	trt_cp	48	D1	-0.5138	-0.2491	-0.2656	0.5288	4.0620	-0.8
4	id_001626bd3	trt_cp	72	D2	-0.3254	-0.4009	0.9700	0.6919	1.4180	-0.8

	c-89	c-90	c-91	c-92	c-93	c-94	c-95	c-96	c-97	c-98	c-99	dataset
20	0.6685	0.2862	0.2584	0.8076	0.5523	-0.1912	0.6584	-0.3981	0.2139	0.3801	0.4176	train
36	0.3192	-0.4265	0.7543	0.4708	0.0230	0.2957	0.4899	0.1522	0.1241	0.6077	0.7371	train
19	-0.0873	-0.7250	-0.6297	0.6103	0.0223	-1.3240	-0.3174	-0.6417	-0.2187	-1.4080	0.6931	train
50	0.2431	-2.0990	-0.6441	-5.6300	-1.3780	-0.8632	-1.2880	-1.6210	-0.8784	-0.3876	-0.8154	train
54	-0.1824	0.0042	0.0048	0.6670	1.0690	0.5523	-0.3031	0.1094	0.2885	-0.3786	0.7125	train

Example of the training set

Number of examples in training set: 23814 entries
Number of features in training set： 875 features
Including 3 Treatment features, 772 gene features and 100 cell features

Part 1.Problem statement of the task

Test set

Testset scale

The test data contains the same features as the train data, and has about 4k rows compared to the 24k rows of the training data. 4000 samples vs 200 targets is not a very high ratio

	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4	g-5	g-6
1	id_0004d9e33	trt_cp	24	D1	-0.5458	0.1306	-0.5135	0.4408	1.55	-0.1644	-0.214
2	id_001897cda	trt_cp	72	D1	-0.1829	0.232	1.208	-0.4522	-0.3652	-0.3319	-1.882
3	id_002429b5b	ctl_vehicle	24	D1	0.1852	-0.1404	-0.3911	0.131	-1.438	0.2455	-0.339
4	id_00276f245	trt_cp	24	D2	0.4828	0.1955	0.3825	0.4244	-0.5855	-1.202	0.5998
5	id_0027f1083	trt_cp	48	D1	-0.3979	-1.268	1.913	0.2057	-0.5864	-0.0166	0.5128
6	id_0042c1364	ctl_vehicle	24	D1	-0.1561	-0.2362	-0.3048	1.598	-0.7165	0.2701	-0.9814

c-87	c-88	c-89	c-90	c-91	c-92	c-93	c-94	c-95	c-96	c-97	c-98	c-99
-0.0035	-0.5184	-0.3485	0.0981	0.7978	-0.143	-0.2067	-0.2303	-0.1193	0.021	-0.0502	0.151	-0.775
0.5913	-0.4333	0.1234	-0.119	-0.1852	-1.031	-1.367	-0.369	-0.5382	0.0359	-0.4764	-1.381	-0.73
-0.5693	0.5062	-0.1925	-0.2261	0.337	-1.384	0.8604	-1.953	-1.014	0.8662	1.016	0.4924	-0.1942
-1.178	0.1909	-1.232	0.126	0.157	-0.1784	-1.12	-0.4325	-0.9005	0.8131	-0.1305	0.5645	-0.5809
0.224	0.8949	0.8668	0.4965	0.7578	-0.158	1.051	0.5742	1.09	-0.2962	-0.5313	0.9931	1.838
-0.4117	0.32	0.3245	-0.416	0.4199	0.0587	0.6118	0.1123	0.6559	-0.0676	0.8351	0.9947	0.2858

Example of the test set

Number of examples in testing set: 3982 entities
Number of features in testing set: 875 features
Including 3 Treatment features, 772 gene features and 100 cell features

Part 1.Problem statement of the task

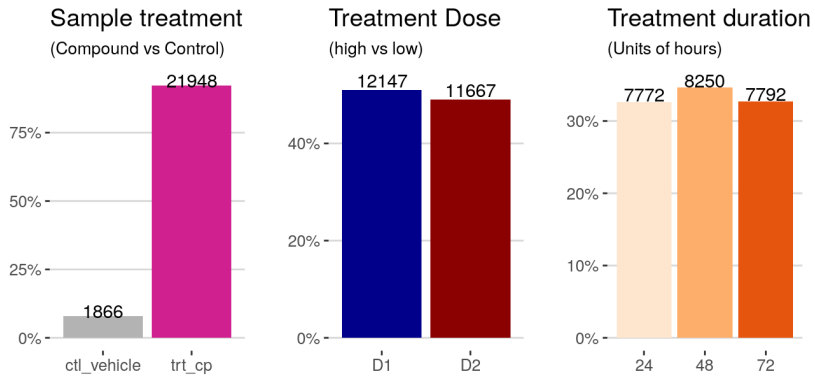
What do the features mean

Categorical features for treatment

Those are the predictor features that describe more generally **how the sample was treated**, in terms of **dose**, **duration**; and whether it was a **“real” treatment or a control**.

	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4	g-5	g-6
1	id_0004d9e33	trt_cp	24	D1	-0.5458	0.1306	-0.5135	0.4408	1.55	-0.1644	-0.214
2	id_001897cda	trt_cp	72	D1	-0.1829	0.232	1.208	-0.4522	-0.3652	-0.3319	-1.882
3	id_002429b5b	ctl_vehicle	24	D1	0.1852	-0.1404	-0.3911	0.131	-1.438	0.2455	-0.339
4	id_00276f245	trt_cp	24	D2	0.4828	0.1955	0.3825	0.4244	-0.5855	-1.202	0.5998
5	id_0027f1083	trt_cp	48	D1	-0.3979	-1.268	1.913	0.2057	-0.5864	-0.0166	0.5128
6	id_0042c1364	ctl_vehicle	24	D1	-0.1561	-0.2362	-0.3048	1.598	-0.7165	0.2701	-0.9814

Treatment features



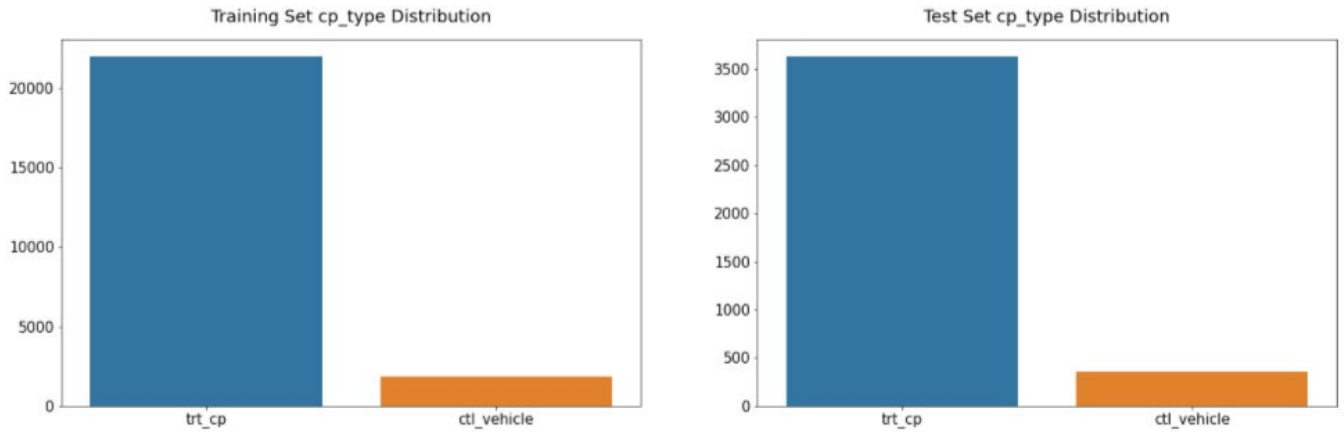
Contrast by 3 categorical features

Part 1.Problem statement of the task

What do the features mean

Categorical features for treatment

cp_type cp_type is the first categorical feature in the dataset and it is a binary feature. It either means that samples are treated with a compound (trt_cp) or with a control perturbation (ctl_vehicle). Samples treated with control perturbations have no MoAs, thus all of their scored and non-scored target labels are zeros. However, all zero labeled samples are not entirely treated with a control perturbation, more than 1/3 of the compound samples are also labeled as zeros.



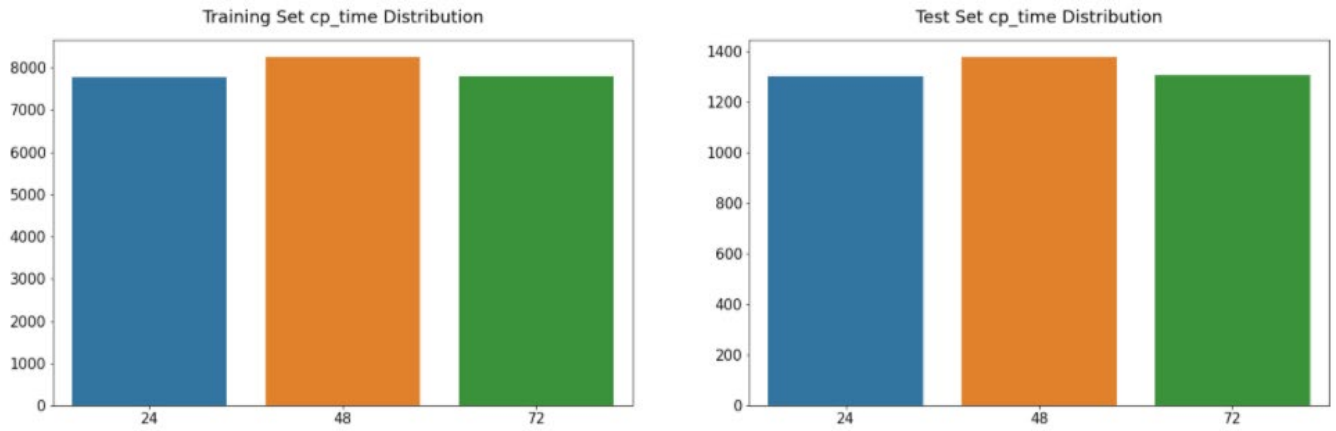
Training set and test set cp_type distribution

Part 1.Problem statement of the task

What do the features mean

Categorical features for treatment

cp_time cp_time is the second categorical feature in the dataset and it has three unique values; 24, 48 and 72 hours. It indicates the treatment durations of the samples. Sample counts of different cp_time values are very consistent and close to each other in different targets. Sample counts are either extremely close to each other or 48 is slightly higher than the others.



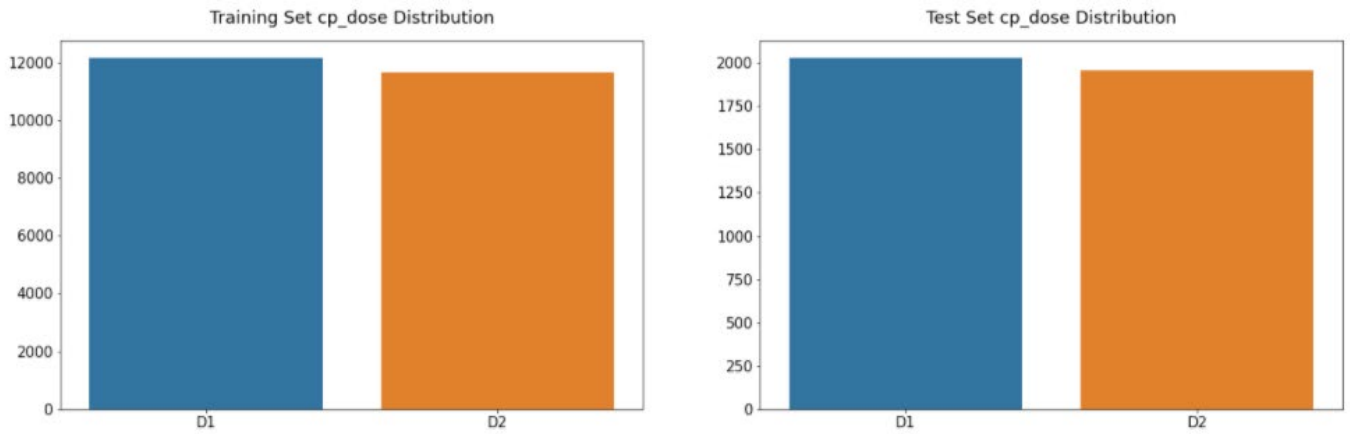
Training set and test set cp_time distribution

Part 1.Problem statement of the task

What do the features mean

Categorical features for treatment

cp_dose cp_dose is the final categorical feature in the dataset and it is also a binary feature. It indicates whether the dose of the samples are either low (D1) or high (D2). Sample counts of different cp_dose values are very consistent and close to each other in different targets. Sample counts are even closer to each other compared to cp_time and lots of targets have equal sample counts for both doses.



Training set and test set cp_dose distribution

Part 1.Problem statement of the task

What do the features mean

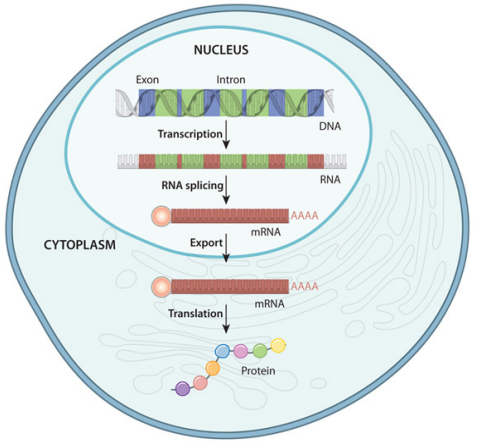
Gene expression

Gene expression is the process by which the information encoded in a gene is used to direct the assembly of a protein molecule. The cell reads the sequence of the gene in groups of three bases.

g-x columns refers to the gene expression. We have 772 column specific for genes properties (i.e. 772 such different features)

	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4	g-5	g-6
1	id_0004d9e33	trt_cp	24	D1	-0.5458	0.1306	-0.5135	0.4408	1.55	-0.1644	-0.214
2	id_001897cda	trt_cp	72	D1	-0.1829	0.232	1.208	-0.4522	-0.3652	-0.3319	-1.882
3	id_002429b5b	ctl_vehicle	24	D1	0.1852	-0.1404	-0.3911	0.131	-1.438	0.2455	-0.339
4	id_00276f245	trt_cp	24	D2	0.4828	0.1955	0.3825	0.4244	-0.5855	-1.202	0.5998
5	id_0027f1083	trt_cp	48	D1	-0.3979	-1.268	1.913	0.2057	-0.5864	-0.0166	0.5128
6	id_0042c1364	ctl_vehicle	24	D1	-0.1561	-0.2362	-0.3048	1.598	-0.7165	0.2701	-0.9814

Gene expression features



a gene is used to direct the assembly of a protein molecule

Part 1.Problem statement of the task

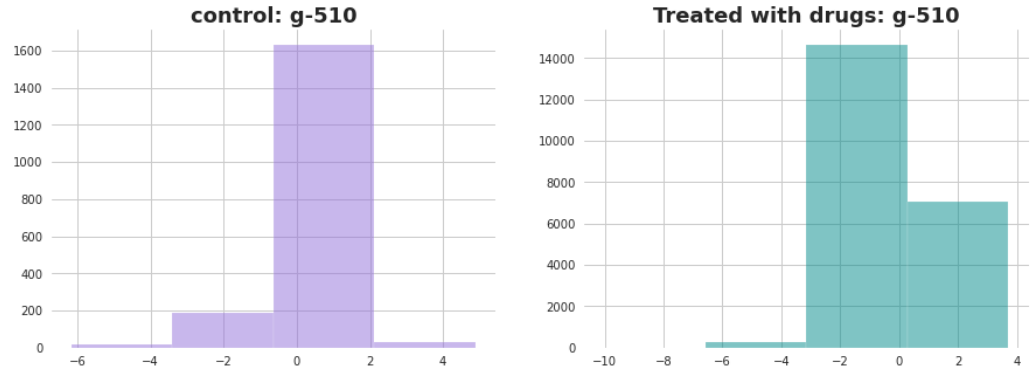
What do the feature values mean

Gene expression

Though **anonymous**, the values for each gene expression and cell viability data has real meanings.

Gene expression is the amount and type of proteins that are expressed in a cell at any given point in time. Gene expression level is based on a protocol similar to L1000(the Landmark genes) which is a high-throughput gene expression assay that measures the mRNA transcript abundance of 978 "landmark" genes from human cells.

Gene expression levels are calculated by the ratio between the expression of the target gene (i.e., the gene of interest) and the expression of one or more reference genes (often household genes). let's compare the samples treated with the drugs and the control samples.



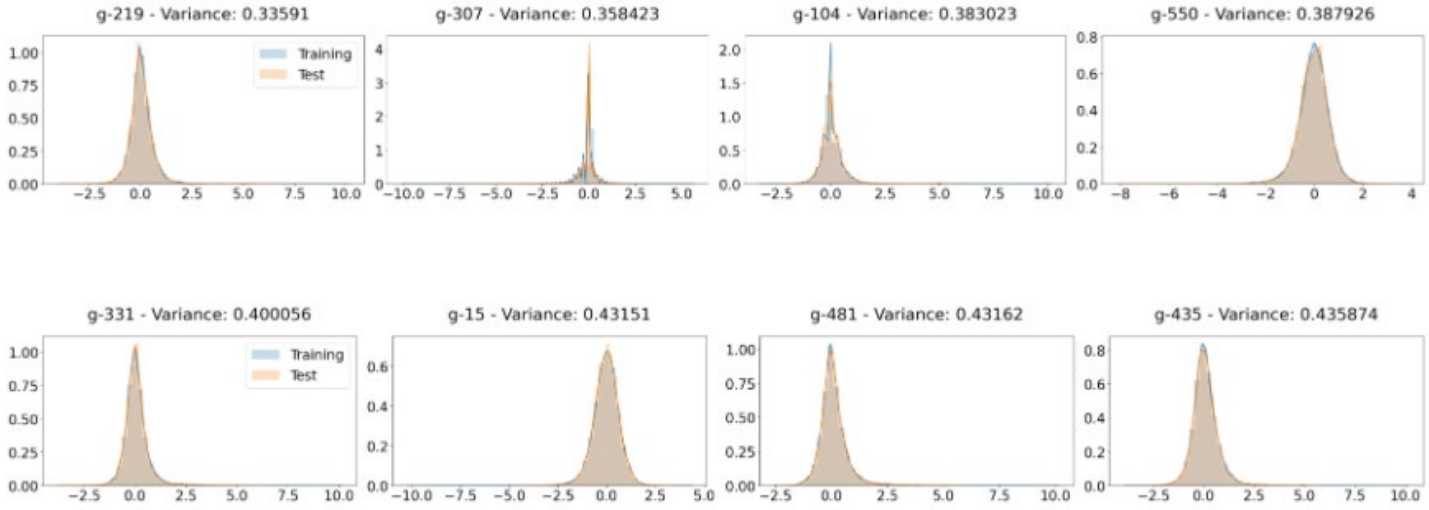
expression of the target gene before and after drugs are introduced

Part 1.Problem statement of the task

What do the feature values mean

Gene expression value distribution

Feature distributions are visualized from low to high variance. Features with variance between 0.3 and 0.4 can be removed since they contain the least information. For other ranges, feature removal should be done with trial and error based on model performances.



Examples of variances of continuous features

Part 1.Problem statement of the task

What do the feature values mean

Cell viability

Cell viability is a measure of the proportion of live, healthy cells within a population. Cell viability assays are used to determine the overall health of cells, optimize culture or experimental conditions, and to measure cell survival following treatment with compounds, such as during a drug screen

	sig_id	cp_type	cp_time	cp_dose	g-0	g-1	g-2	g-3	g-4	g-5	g-6
1	id_0004d9e33	trt_cp	24	D1	-0.5458	0.1306	-0.5135	0.4408	1.55	-0.1644	-0.214
2	id_001897cda	trt_cp	72	D1	-0.1829	0.232	1.208	-0.4522	-0.3652	-0.3319	-1.882
3	id_002429b5b	ctl_vehicle	24	D1	0.1852	-0.1404	-0.3911	0.131	-1.438	0.2455	-0.339
4	id_00276f245	trt_cp	24	D2	0.4828	0.1955	0.3825	0.4244	-0.5855	-1.202	0.5998
5	id_0027f1083	trt_cp	48	D1	-0.3979	-1.268	1.913	0.2057	-0.5864	-0.0166	0.5128
6	id_0042c1364	ctl_vehicle	24	D1	-0.1561	-0.2362	-0.3048	1.598	-0.7165	0.2701	-0.9814

c-87	c-88	c-89	c-90	c-91	c-92	c-93	c-94	c-95	c-96	c-97	c-98	c-99
-0.0035	-0.5184	-0.3485	0.0981	0.7978	-0.143	-0.2067	-0.2303	-0.1193	0.021	-0.0502	0.151	-0.775
0.5913	-0.4333	0.1234	-0.119	-0.1852	-1.031	-1.367	-0.369	-0.5382	0.0359	-0.4764	-1.381	-0.73
-0.5693	0.5062	-0.1925	-0.2261	0.337	-1.384	0.8604	-1.953	-1.014	0.8662	1.016	0.4924	-0.1942
-1.178	0.1909	-1.232	0.126	0.157	-0.1784	-1.12	-0.4325	-0.9005	0.8131	-0.1305	0.5645	-0.5809
0.224	0.8949	0.8668	0.4965	0.7578	-0.158	1.051	0.5742	1.09	-0.2962	-0.5313	0.9931	1.838
-0.4117	0.32	0.3245	-0.416	0.4199	0.0587	0.6118	0.1123	0.6559	-0.0676	0.8351	0.9947	0.2858

Cell viability features

Part 1.Problem statement of the task

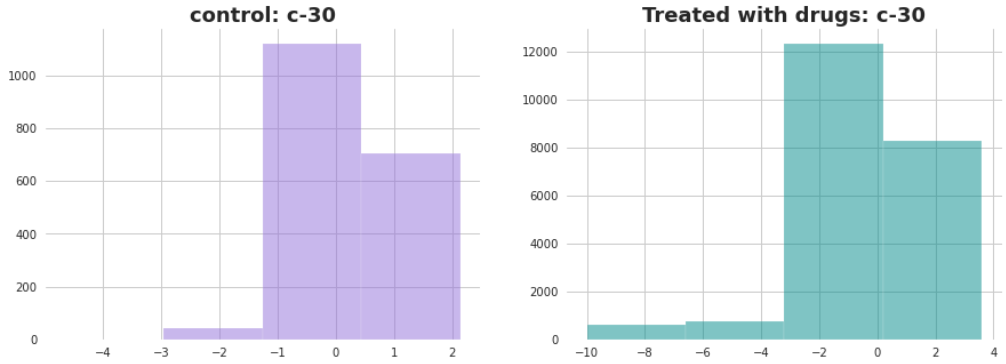
What do the features values mean

Cell viability

Cell Viability can be distinguished from the all-or-nothing states of life and death by the use of a quantifiable index that ranges between the integers of 0 and 1 or, if more easily understood, the range of 0% and 100%. Viability can be observed through the physical properties of cells, tissues, and organs. Some of these include mechanical activity, motility, such as with spermatozoa and granulocytes, the contraction of muscle tissue or cells, mitotic activity in cellular functions, and more.

A high negative cell viability measure reflects a high fraction of killing. High negative values = High number of dead cells. High positive values = High number of living cells.

Here, we have values in the range -10 and 6 because the data were z-scored and then normalized using a procedure called **quantile normalization.**



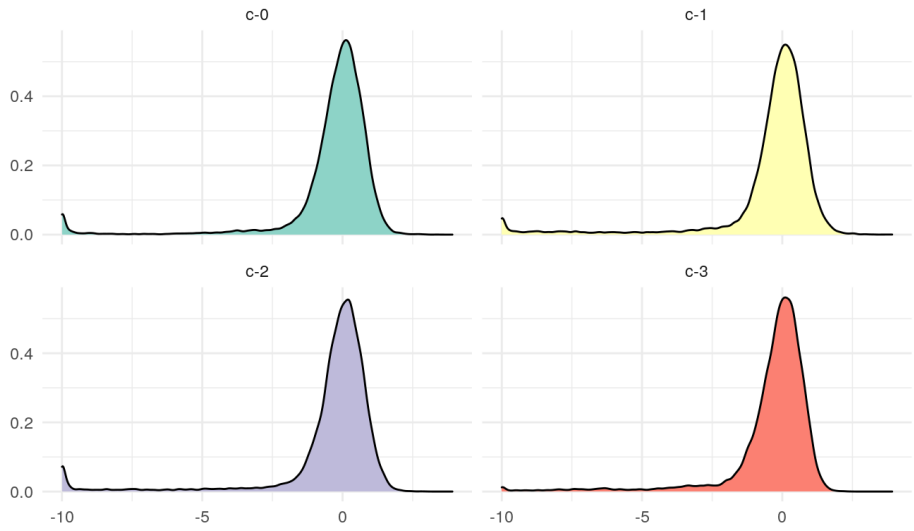
the difference between the cell viability in a control and treated sample.

Part 1.Problem statement of the task

What do the feature values mean

Cell viability feature value distribution

the cell viability, labelled from “c-0” to “c-99”; 100 features. Their distributions look as follows:



Distribution for cell viability deatures

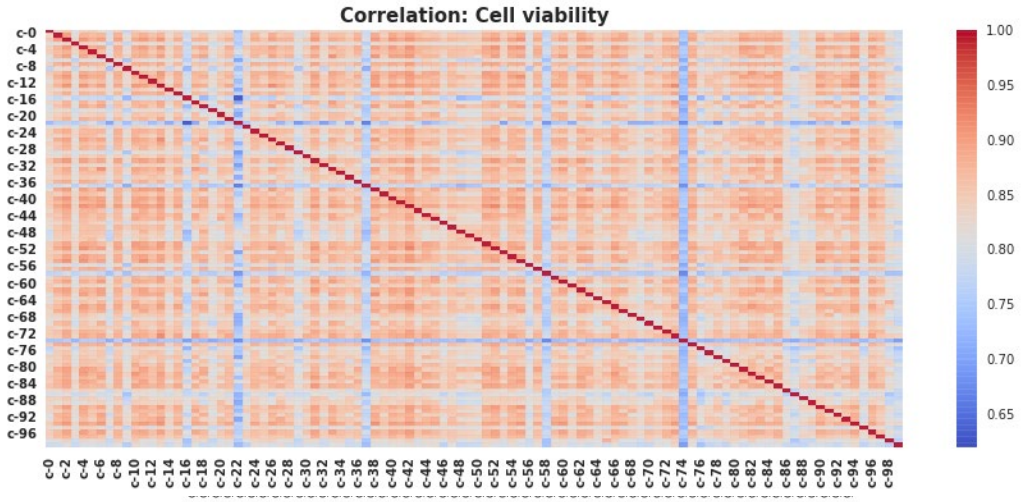
Part 1.Problem statement of the task

Interactions within sets of features

Cell viability correlation

Now we can do the same analysis for the cell viability features. Here we have fewer features (“only” 100) to begin with. First, the overview correlation plot. Cell viability feature **correlations are extremely high**. This can be related to drug effects being similar on most cells lines. Lowest correlations (0.65) are the black lines which belong to c-37, c-58, c-69, c-74 and c-76 features due to their unusual distributions.

We find Values vary between 0.75 and 0.9. Those are reasonably strong correlations, while we can see come negative outliers (if they are real outliers)



Cell viability features correlations

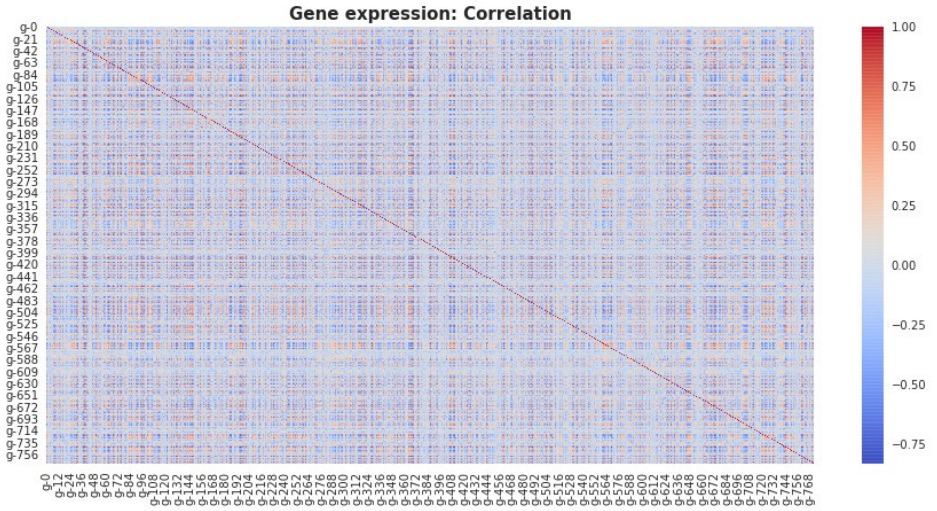
Part 1.Problem statement of the task

Interactions within sets of features

Gene expression correlation

Here are the correlations between the first 200 gene features. It looks like there's an order in which the features are at least somewhat correlated or anti-correlated; and we find that some features show very little correlation at all within those 25% of total columns.

Gene expression feature correlations are more **diverse** compared to cell viability feature correlations. Correlation coefficients vary from -0.75 to 1, because of different distribution types.



Gene expression features correlations

Part 1.Problem statement of the task

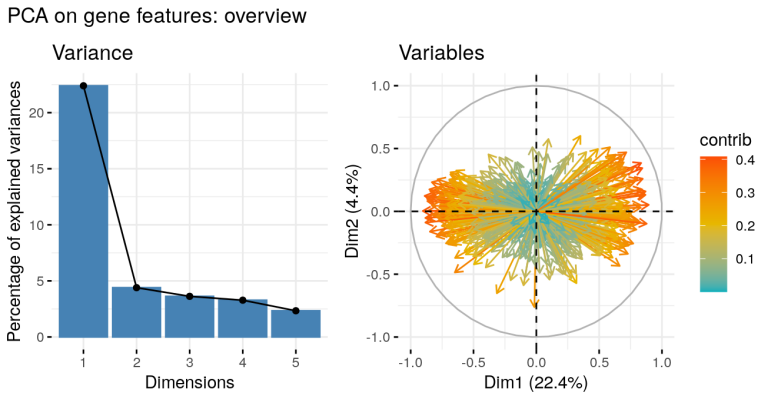
Analysis by PCA

Gene expression features

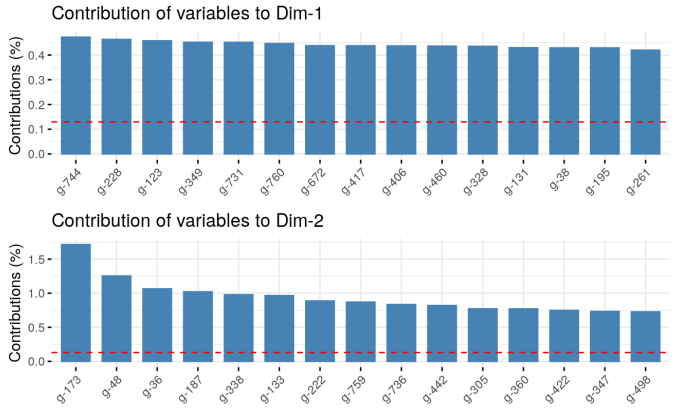
Our first plot will show the amount of variance explained by each PC for the first 5 PCs (a “Scree Plot”), alongside the direction and magnitude (colour) of the contribution of the original features to the first 2 PCs. We find:

1) There’s about 25% of variance in the 1st PC, with the other PCs contributing less than 5% each; which is a notable drop.

2) The variables plot shows this dominance of PC1 by most features having contributions close to the horizontal axis. The strength of the contributions appears to vary quite smoothly within the displayed range.



PCA on gene features: overview



PCA on gene features: variable contributions

Part 1.Problem statement of the task

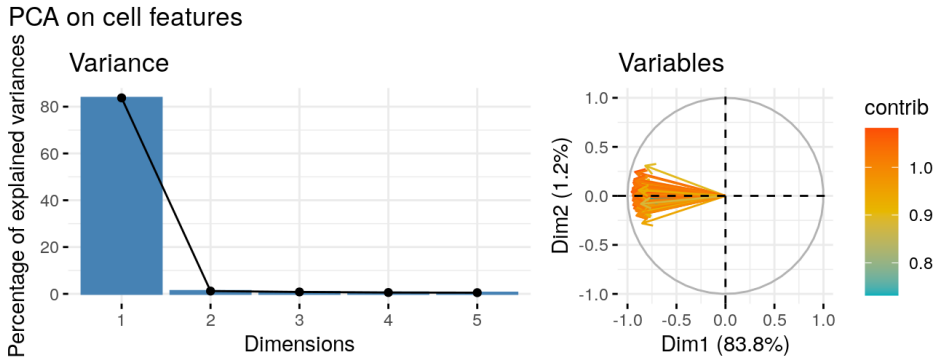
Analysis by PCA

Cell viability features

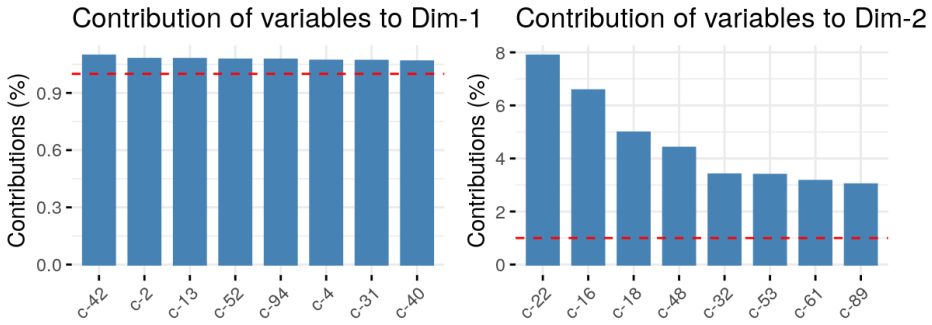
Now we can do the same thing for the cell features. Given the strong correlations that we have seen in Figs.below, I expect the PCA to look quite different from the gene feature findings. First the overview and variable contributions:

We find those high correlations show themselves clearly in the dominance of the 1st PC:

Contribution plots are similar to the gene features, in that many variables contribute a lot to PC1 (in a near uniform fashion) and there's a gradual decline in contribution strength for PC2.



PCA on gene features: overview



PCA on gene features: variable contributions

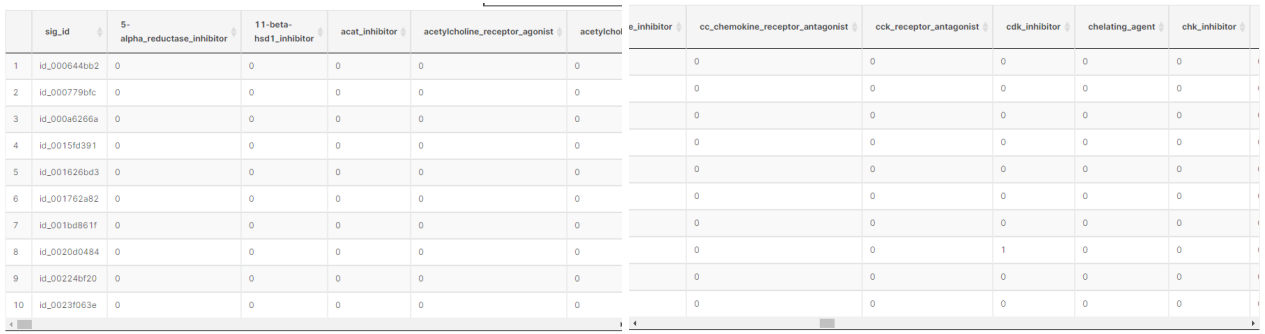
Part 1.Problem statement of the task

Targets

Scored and non-scored targets

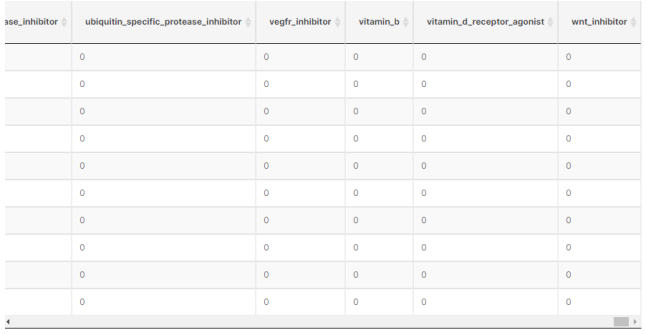
Target features are categorized into two groups; scored and non-scored target features, and features in both of those groups are binary. The final score is based on the scored target features but non-scored group can still be used for model evaluation, data analysis and feature engineering.

The target response are rather wide, too; with just over 200 different binary outputs. As would be expected, this table is pretty sparse（Percentage of non-zero target class values: 0.343%）



	sig_id	5-alpha_reductase_inhibitor	11-beta-hsd1_inhibitor	acat_inhibitor	acetylcholine_receptor_agonist	acetylchol	e_inhibitor	cc_chemokine_receptor_antagonist	cck_receptor_antagonist	cdk_inhibitor	chelating_agent	chk_inhibitor
1	id_000644bb2	0	0	0	0	0		0	0	0	0	0
2	id_000779bfc	0	0	0	0	0		0	0	0	0	0
3	id_000a6266a	0	0	0	0	0		0	0	0	0	0
4	id_0015fd391	0	0	0	0	0		0	0	0	0	0
5	id_001626bd3	0	0	0	0	0		0	0	0	0	0
6	id_001762a82	0	0	0	0	0		0	0	0	0	0
7	id_001bd861f	0	0	0	0	0		0	0	0	0	0
8	id_0020d0484	0	0	0	0	0		0	0	1	0	0
9	id_00224bf20	0	0	0	0	0		0	0	0	0	0
10	id_0023f063e	0	0	0	0	0		0	0	0	0	0

Scored targets：207 targets



ase_inhibitor	ubiquitin_specific_protease_inhibitor	vegfr_inhibitor	vitamin_b	vitamin_d_receptor_agonist	wnt_inhibitor
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0

Non-Scored：407 targets

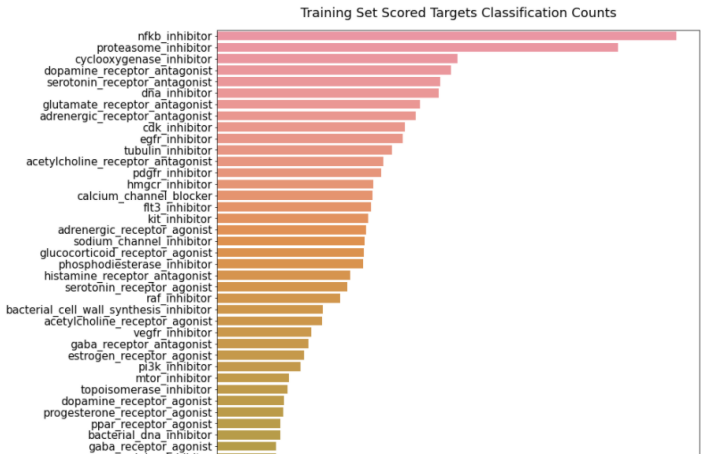
Part 1.Problem statement of the task

Targets

Scored targets

The most commonly classified scored targets are nfkb inhibitor, proteasome inhibitor, cyclooxygenase inhibitor, dopamine receptor antagonist, serotonin receptor antagonist and dna_inhibitor, and there are more than 400 samples classified to each of them. The most rarely classified scored targets are atp-sensitive potassium channel antagonist and erbb2 inhibitor, and there is only one sample classified to each of them. A similar classification distribution is expected in test set.

There are lots of scored targets classified with the same number of times which suggests there might be a relationship between them.



Sub-figure of training set scored targets classification counts

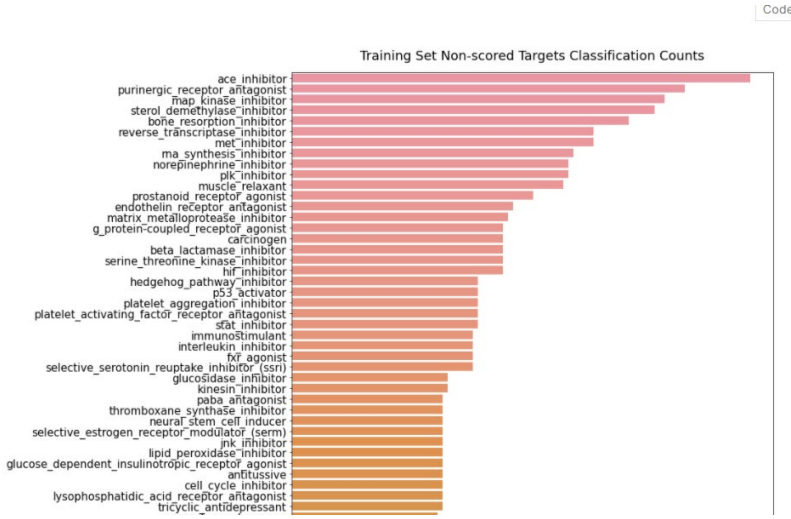
Part 1.Problem statement of the task

Targets

Non-scored targets

The most commonly classified non-scored targets are ace inhibitor, purinergic receptor antagonist, map kinase inhibitor, sterol demethylase inhibitor, and there are more than 70 samples classified to each of them. There are 71 non-scored targets that are classified to 0 samples. The classification counts of non-scored targets are very different than scored targets because most of the training samples are not classified to any of them.

Targets classified same number of times is more significant in non-scored targets and they are more likely to be in a relationship.



Sub-figure of training set non-scored targets classification counts

Part 1.Problem statement of the task

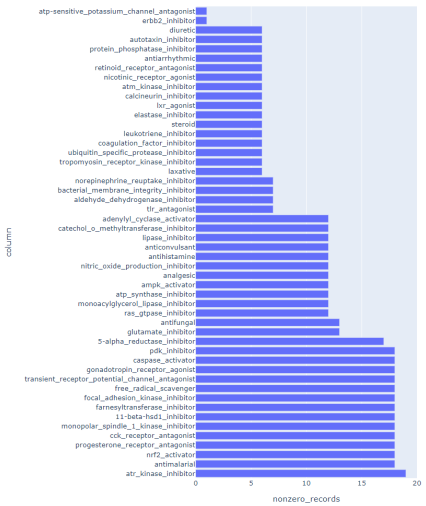
Targets

Issues

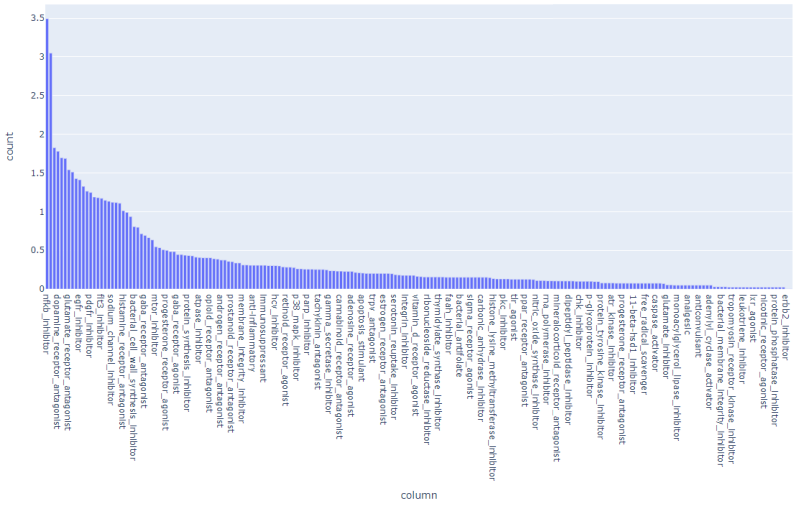
Below are some scored targets which are used to train the main model. As we can see, the targets are very imbalanced and there are only a few positive examples in some labels.

We can see that at least 50 target columns have number pf positive samples less than 20 (about 0.1%) !

The biggest number of positive samples for 1 target column is 3.5%. So we deal here with highly imbalanced data.



Columns with the lowest number of positive samples (top 50)



Percent of positive records for every column in target

Part2. Challenges of the problems and solutions

Part2. Challenges of the problem and solutions

Challenges of the problem

Challenges on Data issues

- Highly corelated features among cell features
- Highly imbalanced data with less positive samples
- Continuous features are between different ranges and have different variance.
- How to make full use of non-scored data

Challenges on Model issues

- Model selection and hyperparameters tuning
- Overfitting issue (issues commonly appears out of deeper neural network and data shortage)

Part2. Challenges of the problem and solutions

Challenges and solutions

Challenges

- Highly corelated features among cell features
- Highly imbalanced data with less positive samples
- Continuous features are between different ranges and have different variance

Solutions

Feature engineering

- adding: introducing as much important information of features
- pruning: eliminate redundant features which have low variance
- rescaling: use gaussian rank scaler which yields more robust results. Gaussian rank scaled data have symmetrical tails unlike others

Part2. Challenges of the problem and solutions

Challenges and solutions

Challenges

How to make full use of non-scored data

Solutions

two-stage training

To extract as much information from non-scored data, here we define our neural network model which consists of several dense, dropout and batch-norm layers. I used different activations after my dense layers.

This network would first be trained on non-scored targets and then transfer weights to train another model with scored targets.

Part2. Challenges of the problem and solutions

Challenges and solutions

Challenges

Model selection and Hyperparameter tuning

Solutions

neurol network

Use neural network as backbones so larger number of features and be better generalized without losing information, So far, neural network models appear to be performing best in this competition

grid search

Use **different combinations** of Hyperparameters and choose the best according to **cross validation score** as well as **score on leaderboard**

Part2. Challenges of the problem and solutions

Challenges and solutions

Challenges

Overfitting and overconfident

Solutions

Smoothing the labels

prevent the network from becoming over-confident and has some sort of regularization effect. It seems this method works well here

Blending

We blend the results of all models using averaging. Blending may result in score improvement taking into the effect of all the models with slight differences.

Choose different models for blending to avoid high covariances between similar architecture of models

Part3. Experiments and results

Part3. Experiments and results

Feature engineering

Add PCA features

Introducing extra feathers by PCA (with extra proportion of 0.95 of the main components)

Number of features increases from 877 to 1526

```
# GENES
n_comp = 600  #<--Update

data = pd.concat([pd.DataFrame(train_features[GENES]), pd.DataFrame(test_features[GENES])])
data2 = (PCA(n_components=n_comp, random_state=42).fit_transform(data[GENES]))
train2 = data2[:train_features.shape[0]]; test2 = data2[-test_features.shape[0]:]

train2 = pd.DataFrame(train2, columns=[f'pca_G-{i}' for i in range(n_comp)])
test2 = pd.DataFrame(test2, columns=[f'pca_G-{i}' for i in range(n_comp)])

# drop_cols = [f'c-{i}' for i in range(n_comp,len(GENES))]
train_features = pd.concat((train_features, train2), axis=1)
test_features = pd.concat((test_features, test2), axis=1)
```

```
#CELLS
n_comp = 50  #<--Update

data = pd.concat([pd.DataFrame(train_features[CELLS]), pd.DataFrame(test_features[CELLS])])
data2 = (PCA(n_components=n_comp, random_state=42).fit_transform(data[CELLS]))
train2 = data2[:train_features.shape[0]]; test2 = data2[-test_features.shape[0]:]

train2 = pd.DataFrame(train2, columns=[f'pca_C-{i}' for i in range(n_comp)])
test2 = pd.DataFrame(test2, columns=[f'pca_C-{i}' for i in range(n_comp)])

# drop_cols = [f'c-{i}' for i in range(n_comp,len(CELLS))]
train_features = pd.concat((train_features, train2), axis=1)
test_features = pd.concat((test_features, test2), axis=1)
```

```
train_features.shape
```

```
(23814, 1526)
```

Experiment on adding PCA features

Part3. Experiments and results

Feature engineering

Adding statistical features

The sum, mean, standard deviation ,kurtosis, skew of feature of gene, cell and the sum of gene and cell

All together there are additional 5*3 = 15 new features

```
def fe_stats(train, test):

    features_g = list(train.columns[4:776])
    features_c = list(train.columns[776:876])

    for df in train, test:
        df['g_sum'] = df[features_g].sum(axis = 1)
        df['g_mean'] = df[features_g].mean(axis = 1)
        df['g_std'] = df[features_g].std(axis = 1)
        df['g_kurt'] = df[features_g].kurtosis(axis = 1)
        df['g_skew'] = df[features_g].skew(axis = 1)
        df['c_sum'] = df[features_c].sum(axis = 1)
        df['c_mean'] = df[features_c].mean(axis = 1)
        df['c_std'] = df[features_c].std(axis = 1)
        df['c_kurt'] = df[features_c].kurtosis(axis = 1)
        df['c_skew'] = df[features_c].skew(axis = 1)
        df['gc_sum'] = df[features_g + features_c].sum(axis = 1)
        df['gc_mean'] = df[features_g + features_c].mean(axis = 1)
        df['gc_std'] = df[features_g + features_c].std(axis = 1)
        df['gc_kurt'] = df[features_g + features_c].kurtosis(axis = 1)
        df['gc_skew'] = df[features_g + features_c].skew(axis = 1)

    return train, test

train_features,test_features=fe_stats(train_features,test_features)
```

Experiment on adding statistical features

Part3. Experiments and results

Feature engineering

Add clustering features

To enhance importance between data with similar features

Concatenate cluster information as additional features

Here we use 35 clusters of gene features and 5 clusters of cell features

and apply the cluster information to each item as extra input features

```
from sklearn.cluster import KMeans
def fe_cluster(train, test, n_clusters_g = 35, n_clusters_c = 5, SEED = 123):

    features_g = list(train.columns[4:776])
    features_c = list(train.columns[776:876])

    def create_cluster(train, test, features, kind = 'g', n_clusters = n_clusters_g):
        train_ = train[features].copy()
        test_ = test[features].copy()
        data = pd.concat([train_, test_], axis = 0)
        kmeans = KMeans(n_clusters = n_clusters, random_state = SEED).fit(data)
        train[f'clusters_{kind}'] = kmeans.labels_[:train.shape[0]]
        test[f'clusters_{kind}'] = kmeans.labels_[train.shape[0]:]
        train = pd.get_dummies(train, columns = [f'clusters_{kind}'])
        test = pd.get_dummies(test, columns = [f'clusters_{kind}'])
        return train, test

    train, test = create_cluster(train, test, features_g, kind = 'g', n_clusters = n_clusters_
g)
    train, test = create_cluster(train, test, features_c, kind = 'c', n_clusters = n_clusters_
c)
    return train, test

train_features ,test_features=fe_cluster(train_features,test_features)
```

Experiment on adding clustering features

Part3. Experiments and results

Feature engineering

Pruning features

eliminate redundant features which have low variance

we can find that threshold=0.80 results in better performance

```
from sklearn.feature_selection import VarianceThreshold


var_thresh = VarianceThreshold(0.8)  #<-- Update
data = train_features.append(test_features)
data_transformed = var_thresh.fit_transform(data.iloc[:, 4:])

train_features_transformed = data_transformed[ : train_features.shape[0]]
test_features_transformed = data_transformed[-test_features.shape[0] : ]


train_features = pd.DataFrame(train_features[['sig_id','cp_type','cp_time','cp_dose']].values.r
eshape(-1, 4),\
                              columns=['sig_id','cp_type','cp_time','cp_dose'])

train_features = pd.concat([train_features, pd.DataFrame(train_features_transformed)], axis=1)


test_features = pd.DataFrame(test_features[['sig_id','cp_type','cp_time','cp_dose']].values.res
hape(-1, 4),\
                             columns=['sig_id','cp_type','cp_time','cp_dose'])

test_features = pd.concat([test_features, pd.DataFrame(test_features_transformed)], axis=1)

train_features.shape
```

```
(23814, 1040)
```

Experiment on pruning features

Part3. Experiments and results

Feature engineering

Feature rescaling

Gaussian rank scaler

Feature scaling can both improve model performance and speed up convergence at the same time. In order to observe scaling effects of different scalers, 25 cell viability and 25 gene expression features are randomly selected and scaled with StandardScaler, MinMaxScaler, and GaussRankScaler. Min max scaled data are between 0 and 1, while standard scaled data and gaussian rank scaled data are zero centric. Zero centric data perform better in algorithms like PCA, on the other hand min max scaled data may perform better in neural networks.

Min max scaler and standard scaler are heavily affected by outliers, however gaussian rank scaler yields more robust results. Gaussian rank scaled data have symmetrical tails unlike others.

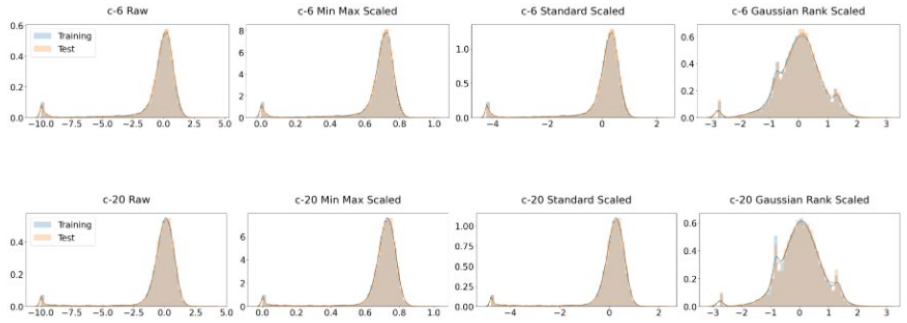
```
# RankGauss - transform to Gauss

for col in (GENES + CELLS):

    transformer = QuantileTransformer(n_quantiles=100,random_state=0, output_distribution="normal")   # from optimal commit 9
    vec_len = len(train_features[col].values)
    vec_len_test = len(test_features[col].values)
    raw_vec = train_features[col].values.reshape(vec_len, 1)
    transformer.fit(raw_vec)

    train_features[col] = transformer.transform(raw_vec).reshape(1, vec_len)[0]
    test_features[col] = transformer.transform(test_features[col].values.reshape(vec_len_test, 1)).reshape(1, vec_len_test)[0]
```

Experiments on Gaussian rank scaler



Effects of different scalers

Part3. Experiments and results

Feature engineering results

Results for feature engineering

Before (ranking 45%)

```
Model :  CV score: 0.01678     LB score:0.01865
```

After(ranking 38%)

```
Model :  CV score: 0.01621     LB score:0.01856
```

Part3. Experiments and results

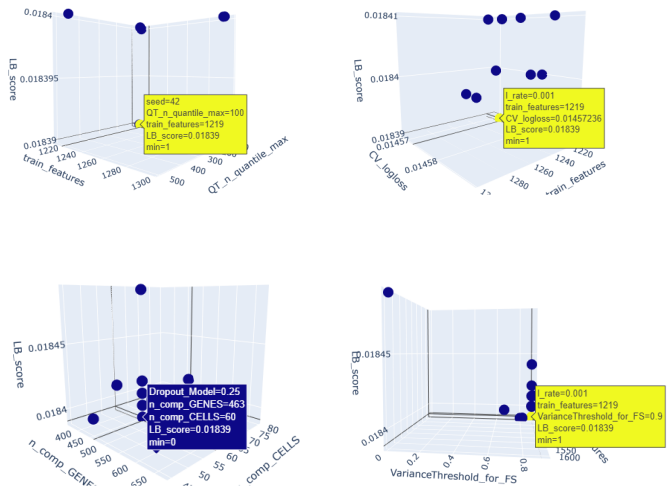
Hyperparameter tuning

3 layer neurol network

Basic backbone 3 layer NN with dropout and batch-norm， Leaky ReLU activation function

Use different combinations of hyperparameters for tuning. Then we can find the best Hyperparameter combinations are

Seed= 42 Variance threshold=0.9, train features = 1219 learning rate = 0.001 Dropout = 0.25



Effects on different combination of hyperparameters

```python
class Model2(nn.Module):
    def __init__(self, num_features, num_targets, hidden_size):
        super(Model2, self).__init__()
        self.batch_norm1 = nn.BatchNorm1d(num_features)
        self.dense1 = nn.utils.weight_norm(nn.Linear(num_features, hidden_size))

        self.batch_norm2 = nn.BatchNorm1d(hidden_size)
        self.dropout2 = nn.Dropout(Dropout_Model)
        self.dense2 = nn.utils.weight_norm(nn.Linear(hidden_size, hidden_size))

        self.batch_norm3 = nn.BatchNorm1d(hidden_size)
        self.dropout3 = nn.Dropout(Dropout_Model)
        self.dense3 = nn.utils.weight_norm(nn.Linear(hidden_size, num_targets))

    def forward(self, x):
        x = self.batch_norm1(x)
        x = F.leaky_relu(self.dense1(x))

        x = self.batch_norm2(x)
        x = self.dropout2(x)
        x = F.leaky_relu(self.dense2(x))

        x = self.batch_norm3(x)
        x = self.dropout3(x)
        x = self.dense3(x)

        return x
```

Experiments on NN model

Part3. Experiments and results

Hyperparameter tuning results

Results for Hyperparameter tuning

Before (ranking 38%)

```
Model :  CV score: 0.01621     LB score:0.01856
```

After(ranking 34%)

```
Model :  CV score: 0.01585     LB score:0.01845
```

Part3. Experiments and results

Two-stage training

Transfer learning

we first trained on non-scored targets and then we transfer weights to train another model with scored targets.

```
def run_training(fold_id, seed_id):
    seed_everything(seed_id)

    train_ = process_data(train)
    test_ = process_data(test)

    kfold_col = f'kfold_{seed_id}'
    trn_idx = train_[train_[kfold_col] != fold_id].index
    val_idx = train_[train_[kfold_col] == fold_id].index

    train_df = train_[train_[kfold_col] != fold_id].reset_index(drop=True)
    valid_df = train_[train_[kfold_col] == fold_id].reset_index(drop=True)

    fine_tune_scheduler = FineTuneScheduler(EPOCHS)
```

Stage 1

```
    pretrained_model = Model(num_features, num_all_targets)
    pretrained_model.to(DEVICE)

    # Train on scored + nonscored targets
    train_model(pretrained_model, 'ALL_TARGETS', all_target_cols)
```

Stage 2

```
    # Load the pretrained model with the best loss
    pretrained_model = Model(num_features, num_all_targets)
    pretrained_model.load_state_dict(torch.load(f"ALL_TARGETS_FOLD{fold_id}_.pth"))
    pretrained_model.to(DEVICE)

    # Copy model without the top layer
    final_model = fine_tune_scheduler.copy_without_top(pretrained_model, num_features, num_all_
targets, num_targets)
```

```
    # Fine-tune the model on scored targets only
    oof = train_model(final_model, 'SCORED_ONLY', target_cols, fine_tune_scheduler)

    # Load the fine-tuned model with the best loss
    model = Model(num_features, num_targets)
    model.load_state_dict(torch.load(f"SCORED_ONLY_FOLD{fold_id}_.pth"))
    model.to(DEVICE)
```

Experiments on two-stage training

Part3. Experiments and results

Two-stage training results

Results for two-stage training

Before (ranking 34%)

```
Model :  CV score: 0.01585     LB score:0.01845
```

After(ranking 27%)

```
Model :  CV score: 0.01562     LB score:0.01839
```

Part3. Experiments and results

Label smoothing

A technique to make model less overconfident

One-hot encoded labels encourages largest possible logit gaps to be fed into the softmax function. Intuitively, large logit gaps combined with the bounded gradient will make the model less adaptive and too confident about its predictions.

In contrast, smoothed labels encourages small logit gaps, as demonstrated by the example below. It is shown in [3] that this results in better model calibration and prevents overconfident predictions.

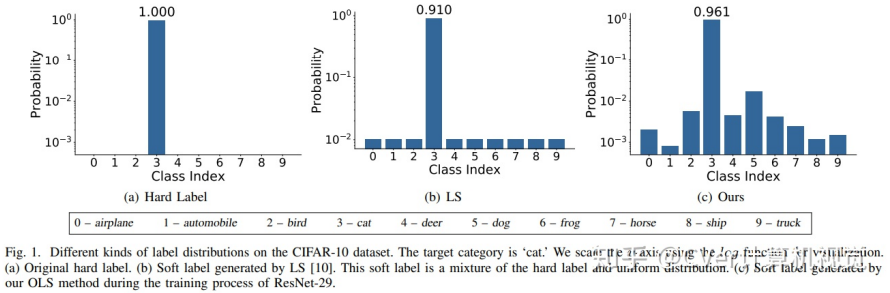


Fig. 1. Different kinds of label distributions on the CIFAR-10 dataset. The target category is 'cat.' We scale the vertical axis using the *log* function for visualization. (a) Original hard label. (b) Soft label generated by LS [10]. This soft label is a mixture of the hard label and uniform distribution. (c) Soft label generated by our OLS method during the training process of ResNet-29.

Label smoothing in NN

```python
def _smooth(targets:torch.Tensor, n_labels:int, smoothing=0.0):
    assert 0 <= smoothing < 1

    with torch.no_grad():
        targets = targets * (1.0 - smoothing) + 0.5 * smoothing

    return targets
```

Experiments on label smoothing

Part3. Experiments and results

Label smoothing results

Results for label smoothing

Before (ranking 27%)

```
Model :  CV score: 0.01562     LB score:0.01839
```

After(ranking 15%)

```
Model :  CV score: 0.01556     LB score:0.01832
```

Part3. Experiments and results

Blending

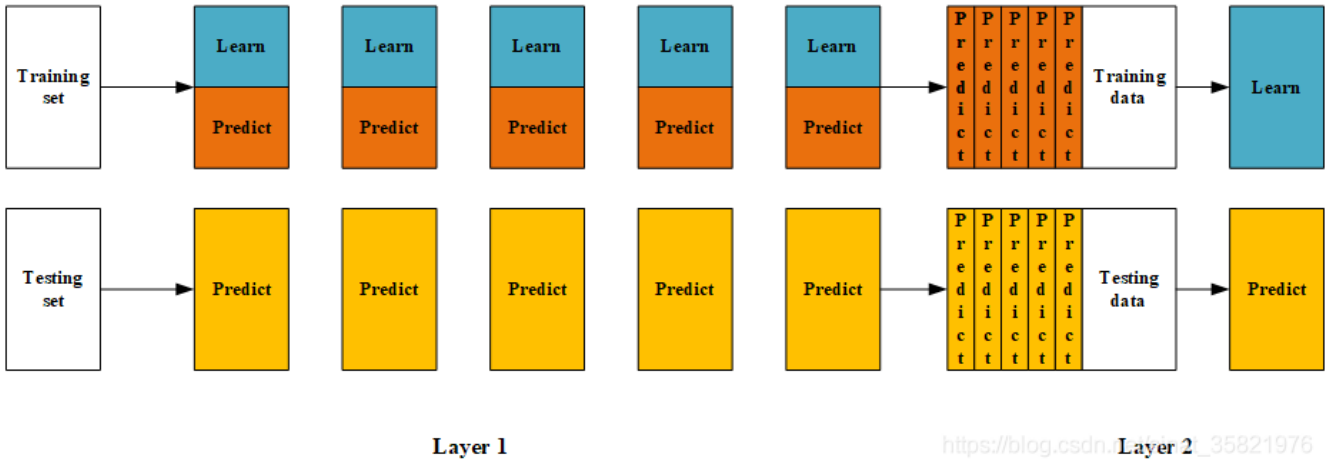
Four models blending

We blend the results of all models using weighted averaging.

Blending may result in score improvement taking into the effect of all the models with slight differences.

We Choose 4 different models for blending to avoid high covariances between similar architecture of models

Including TabNet and Resnet two-stage NN (with deeper layers)and one stage NN(with shallower layers)



Blending for different models

Part3. Experiments and results

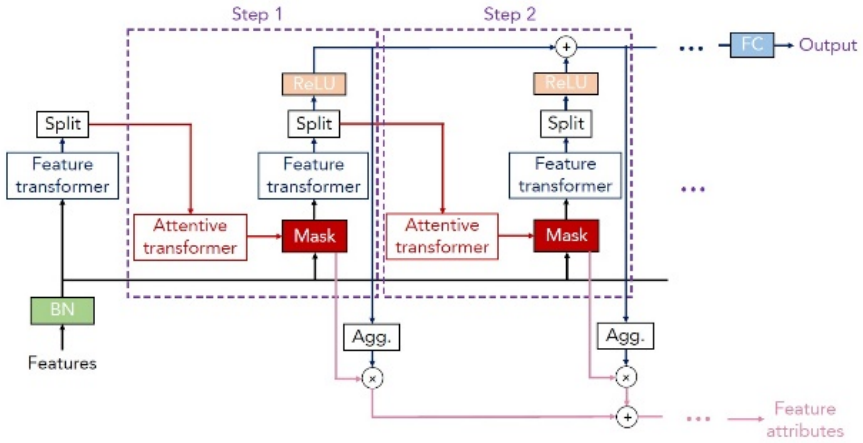
Blending

TabNet and Resnet

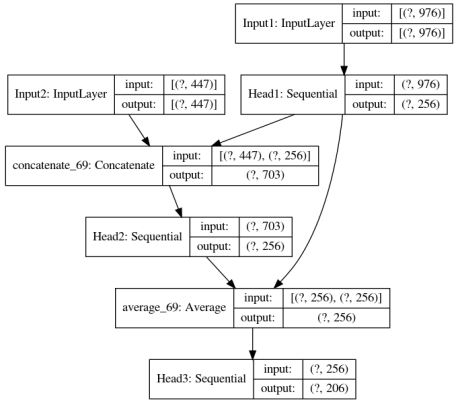
TabNet uses sequential attention to choose which features to reason from at each decision step, enabling interpretability and more efficient learning as the learning capacity is used for the most salient features

TabNet outperforms other neural network and decision tree variants on a wide range of non-performance-saturated tabular datasets

Resnet We use 1-D residual blocks to skip connections for deeper layer to alleviate gradients vanishing problem



Architecture for TabNets



Architecture for Resnet

Part3. Experiments and results

Blending results

Results for blending(final)

we do weighted average according to their CV score:

Val (model_all) = Val(M1)*0.40+Val(M2)*0.20 +Val(M3)*0.20 +Val(M4)*0.20

Before (ranking 15%)

```
Model :  CV score: 0.01556     LB score:0.01832
Mode2 :  CV score: 0.01565     LB score:0.01841
Mode3 :  CV score: 0.01588     LB score:0.01843
Mode4 :  CV score: 0.01610     LB score:0.01847
```

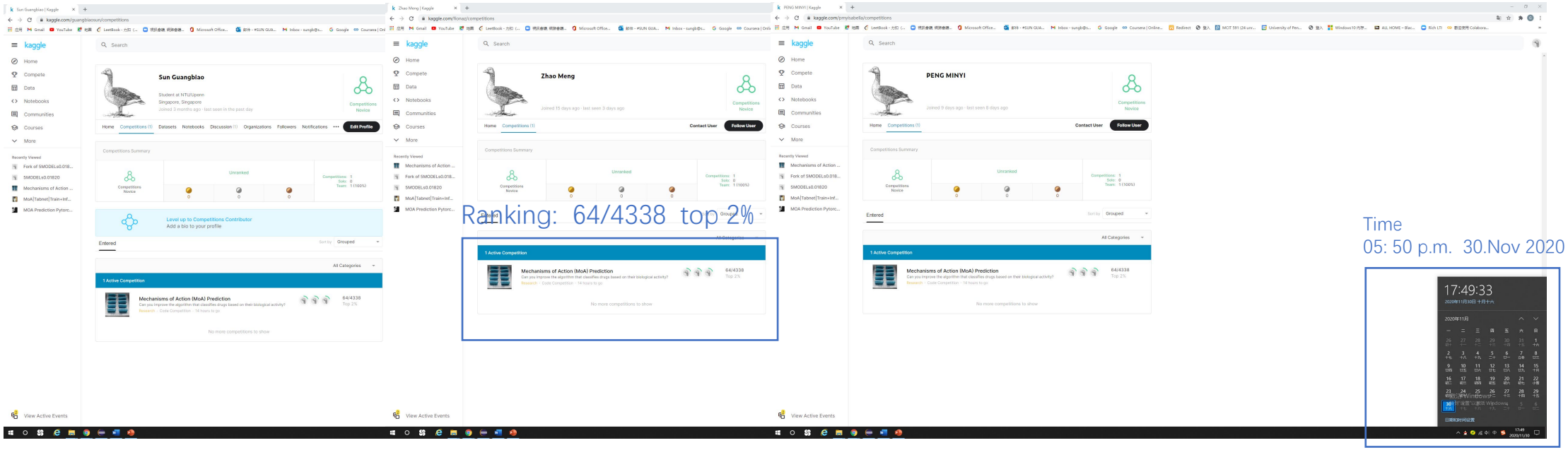
After(ranking 2%)

```
Mode_all :  CV score: 0.01535  LB score:0.01819
```

Part3. Experiments and results

Final score

We have achieved 64/4338，ranking 2% of the whole competitors （before 05：50 p.m. 30.Nov 2020）



Screenshot of final ranking and timestamp

Part 4. Further research

Part 4. Further research

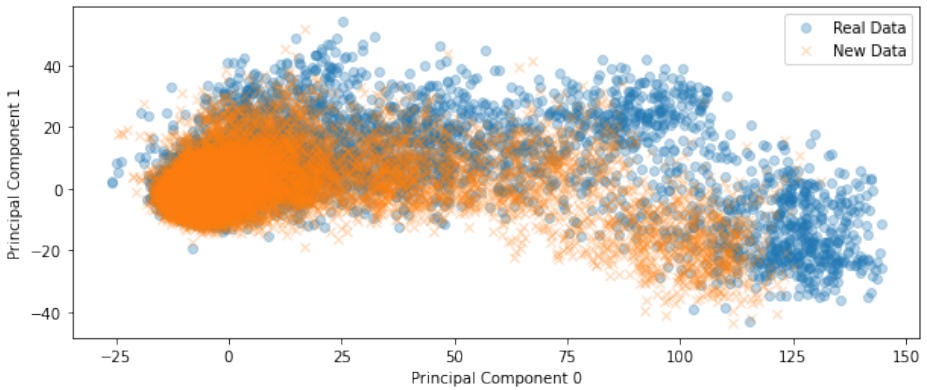
Data augmentation

Domain knowledge for data augmentation

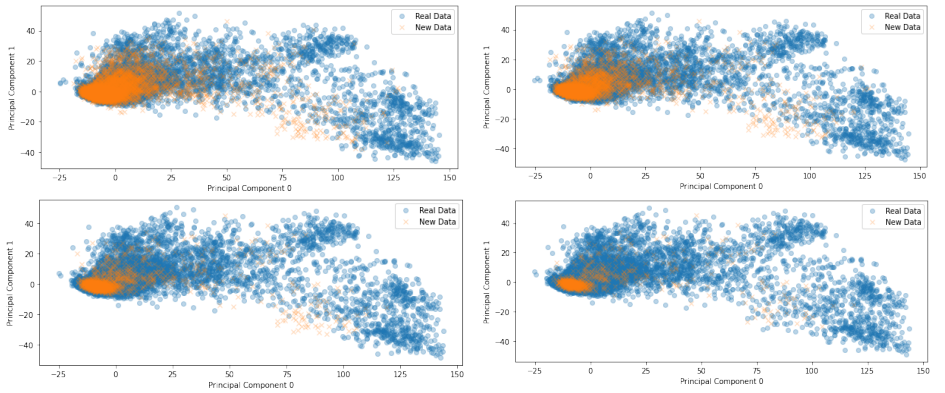
-Throwing more data at deep learning consistently improves performance

-High-level idea: the provided 23814 training examples can be used to predict the outcome of 43748 experiments without actually performing those experiments. We attempts to find compounds that don't interact using a very simple strategy: use compound-pairs with strictly different mechanisms of action

-All cells were treated with 1 compound. Imagine treating the cells with 2 compounds. If the 2 compounds don't interact, we believe the outcome of the experiment can be computed using the provided "1 compound data". There are at least $\frac{3000 \cdot 2999}{2} \approx 4.5 \cdot 10^6$ compound-pairs, so if just 1% of the compound-pairs are don't interact we can predict the outcome of 45000 experiments.



Generating “new” data according to domain knowledge



Different cut-off effects on “new” data generation

Part 5. Conclusion

Part 5. Conclusion

What we have learned from the competition

-**Feature engineering** is an important part for choosing respective model and feeding proper data into it

-**Domain knowledge** helps a lot in key features selection

-We should make full use **information provided beyond the training data**, such as non-scored data, or extra data information from other professional website

-Different type of **Neural networks** are effective when dealing with relatively larger feature space

-**Ensemble** can be a much effective tool to reduce overfitting issue and achieve better performance

Roles of team members

We cooperate as a team to do the following tasks together

- Background information collection
- Exploration of data analysis
- Framework design and update
- Code contribution to test and compete
- Presentation for the final report

The End