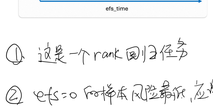
大纲：

1. 比赛评估指标
2. Baseline 建模方法
3. Magic formular
4. Top方案分享

1. 评估指标

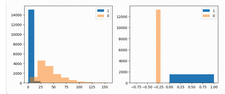


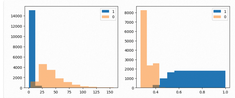
① 这是一个rank回归任务

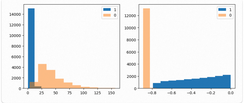
② efs=0 的样本风险最低，应该被判定排在
且不区分它们之间的 rank

③ efs=1 样本 需要区分 rank，即风险排序

2. baseline 建模方法







3. magic function

$$R = p(efs=1) \cdot p(\frac{1}{efs_time} \mid efs=1)$$

通过概率估计风险

1st Place Solution-2 Targets and Ensemble

First, congratulations to the teams that finally "survived" in this competition, and thank you to the participants who shared their experiences and provided help in the forum.

I also trained classifier and regressor models independently and then combined them together with a logic function. Just like 2nd and 4th place solution. The main idea is that when a patient has a high probability of $efs == 0$, we should give Outlier score. Otherwise, we should give risk ranking. These scores and the raw outputs in the form of $1/e_{time}$ for the regressor are used to train a LGBM in the ranking task where $efs == 1$.

Here is the simplified code.

通过概率估计风险

1. Feature Engineering

1. Original features
2. One-hot encoding for all categorical features(but still keep the original category features)
3. Copy continuous features as categorical features

2. CV Strategy

1. The CV is very unstable with different seds. Strange. To make the results comparable, I used 5-folds splits with same seeds for all classifier and regressor model training.

```
y_combine = data['efs'].astype('str')+'_'+data['X']['race_group'].astype('str')
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=888)
skf.split(X_x,combine)
```

1. without one early stopping to reduce the risk of overfitting, especially when I use a large number of folds.

3. Classifier

Target: $F(efs=1)$

Models Used

1. XGBoost, LightGBM, CatBoost
2. NN, TabM
3. GNN
4. NN/TabNet/GNN with pairwise rank loss

tricks

1. One method. I use KNN to find nearest 25 nodes and create edges(use Euclidean Distance),then use GraphSAGE to predict the targets
2. when use rank loss, the position would shift over different fold because rank loss is not sensitive to scale, which may cause lower CV when calculate metric in the entire set. So you need to fix these shifts to keep same mean(mid) prediction value over models trained in difference fold.
3. GNN also have this problem even if I used logloss, I still don't know why.

4. Regressor

Target

Grouped and normalized rank by efs

回归 rank

```
efs_time_norm[efs == 1] = efs_time[efs == 1].rank() / sum(efs == 1)
efs_time_norm[efs == 0] = efs_time[efs == 0].rank() / sum(efs == 0)
```

Models used

XGBoost lightgbm catboost(NN like model didn't work well for me, if any one succeed in NN like model, please let me know.)

Tricks

1. Assign efs as a feature when training and focus on the performance of samples where $efs==1$. Set $efs=1$ when making inference on LB data.
2. apply inverse weight on $efs==1$ and about with 2:8~3:7.
3. Here I use this trick to avoid overfitting. I'd train the regressor only on samples where $efs==1$. When using this model to make inference on samples where $efs==0$, it still should provide reasonable predictions (relative to other samples and ground truth). This is strange because all labels for $efs==0$ is meaningless due to various reasons could cause data censoring. I guess it's because of the SurvivalGAN algorithm. By the way, it turns out that there are similar patterns between samples where $efs==0$ and $efs==1$. So, by adding samples where $efs==0$, the regressor's performance on $efs==1$ improved significantly!

Raw Concordance Index on efs==1

model	c-index
xgboost	0.770229
lightgbm	0.767615
catboost	0.765340

$p(efs=0) \cdot p(efs=1 \mid efs=1)$

5.Merge Function

```
def model_merge(f_mat_RES,Y_mat_CLS,a=2.96,b=1.77,c=0.52):
    '''
    f_mat_RES and Y_mat_CLS need to be scaled to 0~1
    a,b,c need to be tuned with optuna
    '''
    y_fun = (Y_mat_RES*b)*np.log(f_mat_RES)**a)
    s_fun =(f_mat_CLS)*c*(np.abs(f_mat_CLS))**a)
    res = (1-y_fun)*s_fun+y_fun
    res = pd.Series(res).rank()/len(res)
    return res
```

6 Ensembling Method

1. Create combinations between classifiers and regressors, then find the best w_i c_i for the merge function by with 5-folds and then get the merged predictions.

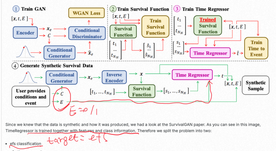
   Here is Stratified Concordance Index of combinations. Ignoring the rank transform in merge function, pretty sure there is no leakage

cls	reg	CV Stratified c-index
xgboost	xgboost	0.693882708644668
xgboost	catboost	0.693390792159249
xgboost	lightgbm	0.692055368954397
catboost	xgboost	0.694677327557885

2. Ensemble the merged predictions using a weighted average with Optuna and 5-folds.
3. Since there are 8 classifiers × 3 regressors = 27 combinations, there is a risk of overfitting. Therefore, I set the Optuna search range for weights to be between 0 and 1, rather than 0 and 1. I think this is a form of regularization.
4. The final CV Stratified c-index is round 0.6965
5. Adding noise to some race groups significantly improved cross-validation performance, but it didn't work on either the public or private LB.

2nd Place Solution

First of all, I want to thank my teammates for this fun ride.



Since we know that the data is synthetic and how it was produced, we had a look at the SurvivalGAN paper. As you can see in this image, TimeRegressor is trained together with features and class information. Therefore we split the problem into two:

- efs classification,
- efs_time regression.

target: efs

For the regression part, we have trained and ensembled 2 models provided efs as an extra feature. Then for the test and validation sets, we make inference by setting efs=1. With this trick, we estimate how long efs time would take if patients efs=1. Other models are ignored and hangup.

efs time

For classification, we use RealMLP, HistGBM and Catboost models with no feature engineering. We ensemble these models by the weights found on CV. We then combine it with regression predictions to calculate the risk scores:

```
df['pred'] = df['pred_time'].rank
df['pred'] = df['pred'] / len(df)
pred_time = 1/df['pred']
```

pred_time 1/efs=1 is regression_prediction in our case.

Additionally, we train a neural network which approximates the competition metric and directly predicts the risk scores. It uses regression prediction from tree models and optimizes the approximate competition metric and auxiliary binary classification loss.

```
def loss_fun(y_risk, y_time, y_efs):
    y_risk, y_time, y_efs = y_risk.ravel(), y_time.ravel(), y_efs.ravel()

    loss = []
    for i in range(len(y_risk)):
        if y_efs[i] == 1:
            risk = y_risk[i]
            t = (y_risk - y_risk[i]).tanh()
            loss.append(t[:i].mean)

    loss = torch.cat(loss).mean()
    return loss
```

We rank ensemble it with our main pipeline.

Here is our inference notebook: https://www.kaggle.com/code/karakasatark2/nd-place-solution-inference

3rd Place Solution

Full code: https://www.kaggle.com/code/churkinnikita/gbm-nn-blend

Best "solo" model (gold medal top-7! Private LB — 0.69694, full training from scratch):

https://www.kaggle.com/code/ukrainskydzibert-competition-nn

First of all, we are grateful to the competition organizers for the possibility to contribute to HCT therapy!

Our solution has the following steps.

1. Validation is the most important one. We made 4-fold CV (as public test part is 25%) and evaluated the score by 20-100 random seeds (depends on the computational complexity). During all this competition, we had excellent CV-LB correlation when measuring average test scores (4 folds). The public "Risk" was approximately at the 75-quantile of the score histogram in our experiments. We emulated and tracked both "public" and "private" scores in this way and created a second submission as more private-oriented. It had a slower LB score but significantly better emulated (and real) private scores.

2. We made a uniform target in the range [0, 1] for uncensored ($efs_i == 1$) observations and in the range [1.1341, 1.3351] for censored ($efs_i == 0$) cases. The target was calculated within train folds and valid folds separately for every race group.

3. Then we divided the train into three parts: separate regressions within uncensored and censored data weighted on the probability of belonging to the classes ($efs_i == 1$) and ($efs_i == 0$).

4. Regression within zeros is the most insignificant part of the composition. For NN models, we just collapse all the $efs_i == 0$ observations to the constant 1.25 (that was tuned to show maximal performance in terms of the concordance index).

5. The target range [0, 1] for the regression task allowed us to train the models via binary cross-entropy loss, which is much more profitable in comparison with MSE/RMSE/MAE errors.

6. Thus, the prediction has the form:

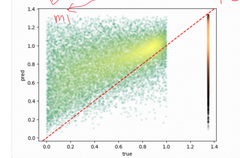
$$y_i^{end} = \mathbf{P}(efs_i = 1)x_i\mathbf{E}(risk|efs_i = 1) + \mathbf{P}(efs_i = 0)x_i\mathbf{E}(risk|efs_i = 0)$$

m2

and we obtain an overall scatter plot of type:

m1

m1 + m2 = 1



7. Our model get 1st place at Catboost, LightGBM, XGBoost, MLP with ODST, and TabM models (separate exemplars of each for regression and classification). NNs were the best at classification and GBMs are the champions for regression. Ranking models have weaker performance in the competition.

8. The data is noisy, and the classic methods of noise reduction worked well. We averaged the models with fixed hyperparameters / architectures over random states and eliminated the observations with giant regression errors from the training. But in the case of classification, the outlier denoising leads to the overfitting (ROC AUC becomes better, but LogLoss worsens at the same time).

9. For the rank-free stage we blended the regressors by convex combination and stacked the classifiers by logistic model. Stacking by logreg always gave particular performance on the private part of the test data that we saw via out-of-fold estimate of the private / private split. All blending weights were optimized simultaneously to maximize the concordance index on 20 OOF predictions (20 random seeds) using Sherman-optimization.

$$\mathbf{P}(efs = 1x) = \sigma(b_0 + w_1\mathrm{CatBoost}_{cl} + w_2\mathrm{LGBM}_{cl} + w_3\mathrm{XGB}_{cl} + w_4\mathrm{MLP}_{cl} + w_5\mathrm{TabM}_{cl})$$
$$\mathbf{E}(risk|efs = 1) = w_1\mathrm{CatBoost}_{r,fs=1} + w_2\mathrm{LGBM}_{r,fs=1} + w_3\mathrm{XGB}_{r,fs=1} + w_4\mathrm{TabM}_{r,fs=1}$$
$$\mathbf{E}(risk|efs = 0) = b_1 + 0.333\,\mathrm{CatBoost}_{r,fs=0} + 0.333\,\mathrm{LGBM}_{r,fs=0} + 0.333\,\mathrm{XGB}_{r,fs=0}$$
$$w_1 + w_2 + w_3 + w_4 = 1$$
$$Concordance(b_0, b_1, w_0, w_1, \ldots, w_5) \to \max$$

The resulting ensemble is:

Overall performance of the components:

Model	CV ("Public" estimate — folds x seeds)	Public LB	Private LB
CatBoost	0.68586	0.69424	0.69577
LightGBM	0.68530	0.69318	0.69572
XGBoost	0.68416	0.69252	0.69493
NN	0.68554	0.69540	0.69664
Blend GBMs	0.68932	0.69608	0.69784
Blend All + LogReg	0.69139	0.69692	0.69937

My teammates disclose the details in the comment section.

4th place: Factorized modelling approach

Full model* code: https://www.kaggle.com/code/herrshuo/4th-place-solution

Private score: 0.69936, Public score: 0.69724, total runtime: 4h

*this version also includes LightGBM model not mentioned below, but impact is minimal in ensemble

For comparison solo model scores (without bagging):

CatBoost: 0.69784, 0.69500, 30min

XGBoost: 0.69765, 0.69536, 5min

TabM: 0.69636, 0.69383, 6min

LightGBM: 0.69793, 0.69596, 5min

~~~~

It was fun to make a small comeback to Kaggle after a long break. Especially given this nice "old school" competition, not too big & tabular dataset. Below is a short summary of my solution.

Introduction

The goal of this competition was to rank patients based on their risk scores for allogeneic HCT transplantation events. As a first step it's useful to think what a perfect solution would look like. In this case that would be:

1. all patients with no event (efs=0) should be at the bottom of the list (no order defined between them)
2. for efs = 1, patients are ordered by the event times

Clearly this ranking problem is non-differentiable, especially as the S1 introduces step function like behavior for the desired ranks. One common way to handle this type of problems is to introduce some sort of soft/smoothed version of the original task. However, in my experience, such approaches tend to make things unstable during training, or just don't really match to the original problem that well if smoothed too much.

So instead, my main idea for the competition was to follow the steps 1-2) directly and divide the problem into two parts:

1. predict the probability of the event,
2. predict the expected ranking position among patients with events

Finally, given these two predictions, we can calculate the expected ranking position as a risk score.

Model formulation:

```
risk_score = P(event_i = 0)*(s0_group/2) + P(event_i = 1)*(s0_group + (1-s0_group)*E[rank\ | event = 1])
, where s0_group = sum_{race_group != group}P(event = 0) / N_group
```

P(event):

- binary classification for efs events
- censored data points are treated as uncertain observations
- weights for those data points are calculated as cumulative densities given by Kaplan-Meier estimator and scaled to [0,5] range.

E[rank% | event = 1]:

- conditional regression model to predict time based rankings for data points where event did happen
- the model is trained using only data points which have efs = 1
- target is rank% = rank(-time)/N
- so, target values are between 0 and 1, 0 = longest time and 1 = shortest
- with all algorithms, I'm using target transformation of the form: inverse_normal_cdf(rank%) "z-score", and then predictions are transformed back to percentages by normal_cdf(pred)

Algorithms:

For both of these tasks, three different models are implemented using the following packages/algorithms:

- TabM
- CatBoost
- XGBoost

So, in total six models are trained. First their predictions are merged by just calculating weighted sums (separately for both time and event using different weights), and then finally risk scores are calculated by the the formula given above.
~~~~