ICR Competition Records

Pu Tan

Competition link: https://www.kaggle.com/competitions/icr-identify-age-related-conditions

Goal of the Competition
The goal of this competition is to predict if a person has any of three medical conditions. You are being asked to predict if the person has one or more of any of the three medical conditions (Class 1), or none of the three medical conditions (Class 0). You will create a model trained on measurements of health characteristics. To determine if someone has these medical conditions requires a long and intrusive process to collect information from patients. With predictive models, we can shorten this process and keep patient details private by collecting key characteristics relative to the conditions, then encoding these characteristics.

Goal: Discover the relationship between measurements of certain characteristics and potential patient conditions.

This documentation outlines my approach during the competition. The details may differ slightly from the actual implementation, with the full code available here [link].

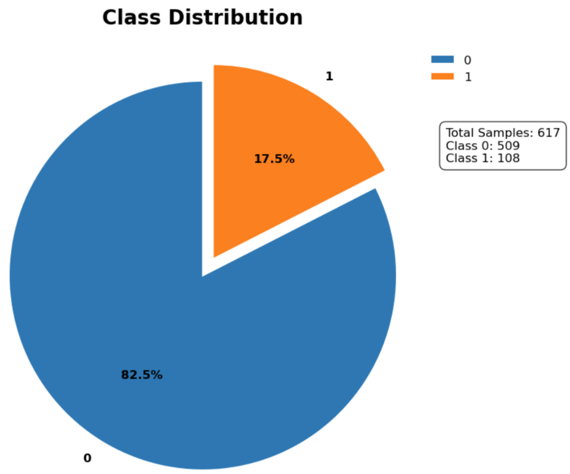
The feature engineering techniques are described first, highlighting the key steps taken to construct informative features from the data. This includes [missing value imputation, normalization, discretization, feature selection based on feature crossing and feature importance analysis, XGBoost tuning, EDA driven engineering, etc.].

Additional aspects of the methodology are then covered, including data preprocessing, hyperparameter tuning strategies, model evaluation metrics, and other considerations for this competition. Dilution records and other supplementary material are referenced in context where applicable.

While the documentation summarizes the techniques at a high-level, the full code contains the specifics for reproducibility. The goal is to provide an overview of the methodological process, with comprehensive details in the codebase. Suggestions to enhance formality and clarity of this competition writeup are welcomed.

1. EDA feature engineer

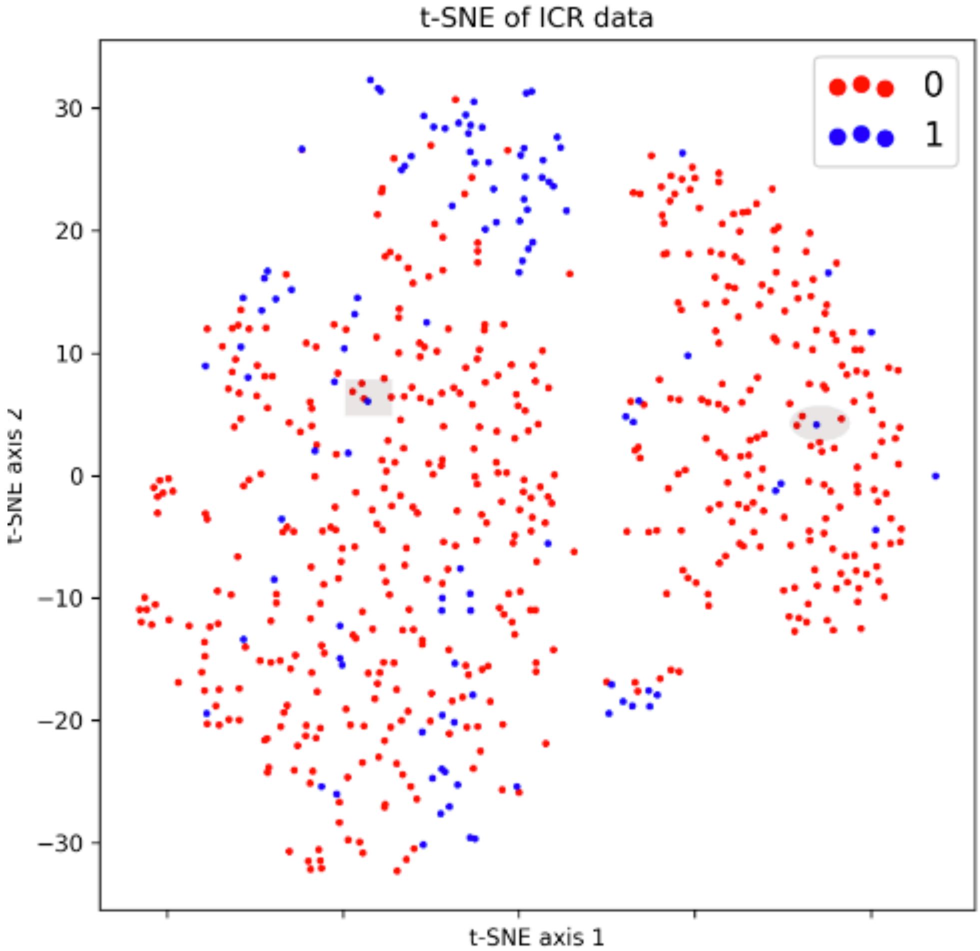
1、class distribution



2、feature distribution

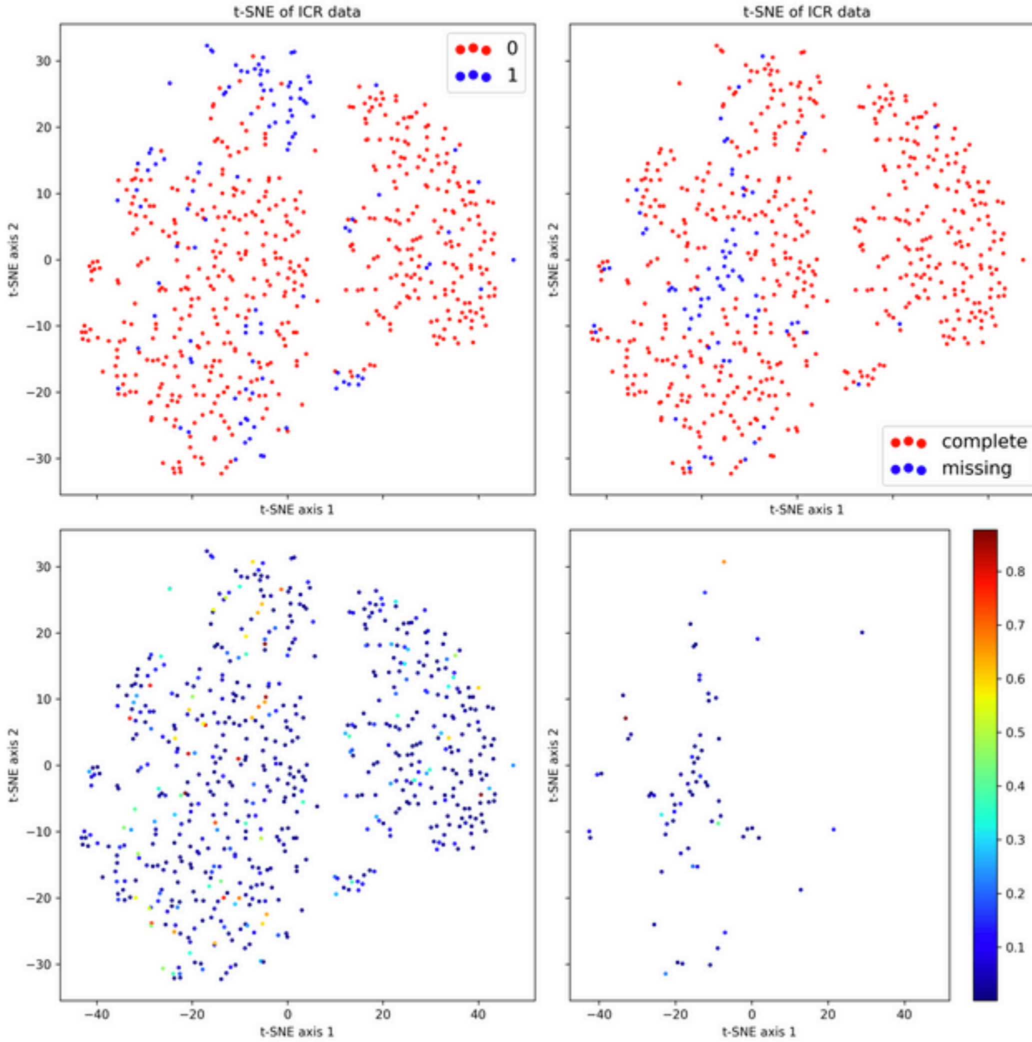
t-SNE to reduces high-dimensional data to two or three dimensions. (note: t-SNE focuses more on retaining the local features of the original data. Points that are close together in high-dimensional space are projected to be close together in the low-dimensional space as well.)

The distributions of the positive and negative classes are shown. For class 1, some points have most of their neighbors as class 0. These points are hard to predict, especially the blue points in the left rectangle in the bottom figure, which almost overlap with a red point. Their 56 features are almost the same.



Lower Left: Out-of-fold validation is performed on the training data to observe the error rate in predicting the true class labels. It shows that most of the predictions are fairly accurate (indicated in blue).

Lower Right: Focusing only on the points with missing values, the predictions are mostly good.



The idea is to identify points that are difficult to predict and then create a model to predict whether a point will be hard to forecast.

useful idea

code

Some say that by only considering relevant variables, the different classes in this graph can become more distinct.

Pertains to the distribution of different categories as well as the distribution of points with missing values.

2. Data Processing

2.1 Missing Value

Two rows have many missing values; most such rows also have outliers. Simple mean or median imputation could be problematic, especially for rows with multiple outliers.

Note on BQ: Using a simple mean or median to fill in those 60 values could negatively impact the results, especially in rows with multiple outliers.

```
---  ------  --------------  -----
0    Id      617 non-null    object
1    AB      617 non-null    float64
2    AF      617 non-null    float64
3    AH      617 non-null    float64
4    AM      617 non-null    float64
5    AR      617 non-null    float64
6    AX      617 non-null    float64
7    AY      617 non-null    float64
8    AZ      617 non-null    float64
9    BC      617 non-null    float64
10   BD      617 non-null    float64
11   BN      617 non-null    float64
12   BP      617 non-null    float64
13   BQ      557 non-null    float64
14   BR      617 non-null    float64
15   BZ      617 non-null    float64
16   CB      615 non-null    float64
```

2.1.1 Method: KNNImputer
https://blog.csdn.net/tMb8Z9Vdm66wH68VX1/article/details/130177587

```python
imp = KNNImputer()
labels = train["Class"]
train = train.drop(columns="Class")
data = imp.fit_transform(train)
tmp = pd.DataFrame(columns=train.columns, data=data)
train = pd.concat([tmp, labels], axis=1)train
```

Note:

1. Calculate the correlation between the imputed columns and the class label, and choose different imputation methods accordingly.
   Filling with value 1.331155 changes correlation, meaning increases precision of outcome, - by 3%

```python
df_experimental =  YOURDATASETNAME.loc[:,['BQ','EL' ,'Class']].copy()

df_experimental["BQ_NEW"]=df_experimental['BQ'].fillna(ds["BQ"].mean())
ds["BQ_NEW"]=np.where(ds['BQ']>0,ds['BQ'],1.331155)

df_experimental["BQ_NEW"].corr(df_experimental["Class"])
ds["BQ_NEW"].corr(ds["Class"])
```

2. Separate the data by class, use k-nearest neighbors (KNN) to impute missing values, and then combine the data back together.

2.1. Directly set to -1 and bucketize; missing values form their own separate category.

3. A missing value handling method mentioned in the comments section [Kaggle Discussion](https://www.kaggle.com/competitions/icr-identify-age-related-conditions/discussion/410843).

Hey, just wanted to share more on this!

Here are other alternatives for replacing the `NaN` values

```
# replace the NaN values with the mean(average) value
df.fillna(df.mean())

# replace NaN values with the mode (most frequent) value
df.fillna(df.mode().iloc[0])
```

Another famous way is also by using the `interpolate()` method, this method is very powerful as it supports different methods;

method : {'linear', 'time', 'index', 'values', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'barycentric', 'krogh', 'polynomial', 'spline', 'piecewise_polynomial', 'from_derivatives', 'pchip', 'akima'}

Here's a quick example:

```
df = pd.DataFrame({'a':[1, 2, 3, np.nan], 'b':[4, np.nan, 6, 7]})
df
```

	a	b
0	1.0	4.0
1	2.0	NaN
2	3.0	6.0
3	NaN	7.0

2.2 Normalizations

2.1.1. StandardScaler (Mean = 0, Standard Deviation = 1)

- Method: Subtract the mean and divide by the standard deviation. The transformed data follows a standard normal distribution with a mean of 0 and a standard deviation of 1.
- Transformation Function: x = (x-mean) / std
- Applicability: Suitable for data that already follows a normal distribution.
- Impact of Outliers: Somewhat robust to outliers, although outliers still affect the calculation of mean and standard deviation.

2.2.2. MinMaxScaler (0-1 Scaling)

- Method: Scale the features to fall within a given minimum and maximum value range, typically between 0 and 1. This is a linear transformation of the original data.
- Transformation Function: x = (x-min) / (max-min)
- Applicability: Suitable for data with a relatively stable range. If new data points alter the max/min values, rescaling will be necessary.
- Impact of Outliers: Highly sensitive to outliers, as they can distort the minimum and maximum values used for scaling.

2.2.3. RobustScaler (Quantile Scaling)

- Method: This scaler removes the median and scales the data based on the Interquartile Range (IQR). The IQR is the range between the 1st quartile (25th percentile) and the 3rd quartile (75th percentile).

- Applicability: Suitable for data that contains many outliers.

- Impact of Outliers: The RobustScaler minimizes the impact of outliers by scaling using the IQR, thereby making it robust to outliers.

By using the IQR for scaling, the RobustScaler is less influenced by extreme values and can be particularly useful when the dataset contains many outliers. This method is often recommended for data sets where the features are not normally distributed and contain a large number of outlier values.

refer

```Python
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
sc = StandardScaler() # MinMaxScaler or RobustScaler
X_train[numeric_columns] = sc.fit_transform(X_train[numeric_columns])
X_test[numeric_columns] = sc.transform(X_test[numeric_columns])
##通过调用 transform 方法，可以使用前面获得的样本均值和方差来对数据做标准化处理
```

2.3 Binning

	a	b	c
1			
2			
3			

	a	b	c	d	e		
1	NA	0.2					
2	0.1	0.3					
3	0.3	0.4					

	a	b
1		0.2
2	0.1	0.3
3	0.3	0.4

0.2-0.4 equidistant 0.1

0.2 a

0.3 b

b1 b2 b3 b4

1   0   0

0   1   0

	a	b
1	A	0.2
2	b	0.3
3	b	0.4

Numerical to categorical

Equidistant, Equal Frequency, Log and then take Integer.

Only perform log transformation on specific columns.

```
Python
log_cols = [_ for _ in X_train.columns if _ not in ['EJ', 'BN', 'CW', 'EL', 'GL']]
X_train.loc[:, log_cols] = np.log1p(X_train.loc[:, log_cols])
X_test.loc[:, log_cols] = np.log1p(X_test.loc[:, log_cols])
```

2.4 Unbalanced Data

https://www.kaggle.com/competitions/icr-identify-age-related-conditions/discussion/412507

2.4.1.down/up sampling

```
Python
def random_under_sampler(df):
#if sampling_method == 'under':
    neg, pos = np.bincount(df['Class'])
    one_df = df.loc[df['Class'] == 1]  #108
    zero_df = df.loc[df['Class'] == 0] #509

    zero_df = zero_df.sample(n=pos) #108

    undersampled_df = pd.concat([zero_df, one_df])#216
    return undersampled_df.sample(frac = 1)

train_good = random_under_sampler(train)
```

Choose oversample

2.4.2. Oversampling, Data Synthesis

```python
## 2. oversampling ->
#SMOTE- generate new sample via interpolation
if sampling_method == 'over':
    X = train[selected_cols]
    y = train['Class']

    smote = SMOTE(k_neighbors=5)
    # fit_resample
    X_resampled, y_resampled = smote.fit_resample(X, y)
    print(X_resampled.shape, y_resampled.shape)
    X_resampled["Class"] = y_resampled
    train = X_resampled

#RandomOverSampler- Oversample by duplicating some of the original minority class samples
ros = RandomOverSampler(random_state=42)
train_ros, y_ros = ros.fit_resample(train_pred_and_time, greeks.Alpha)
```

> I am confused by over sampling here. We first oversample the dataset and then split it into folds for cross-validation. But as a result, some samples might occur in both training and validation for a given fold, distorting the loss.
>
> I did over sampling after the split(inside training function) but my validation loss sky rocketed from 0.05-0.06 to 1.65-1.75. Is my reasoning correct or I did something dumb and shouldn't be surprised with the CV loss exploding?

2.4.3. Solve this issue from the perspective of loss, look for some solutions like multitask MMOE.

Focal loss GHM loss

2.4.4. Class weight: scale_pos_weight can only be used for binary classification problems.

```python
LGBMClassifier(class_weight='balanced')
XGBClassifier(scale_pos_weight=4.71)
CatBoostClassifier(auto_class_weights='Balanced')
LogisticRegression(class_weight='balanced')
LinearDiscriminantAnalysis(priors=[0.5, 0.5])
```

3. Feature Selection

3.1 important feature initial election

3.1.1 vif:

[Use Variance Inflation Factor (VIF) for feature selection]

```Python
def check_vif(df):
    vifs = [variance_inflation_factor(df, i) for i in range(df.shape[1])]
    vif_df = pd.DataFrame({"features":df.columns, "VIF" : vifs})
    vif_df = vif_df.sort_values(by="VIF", ascending=False)
    remove_col = vif_df.iloc[0, 0]
    top_vif = vif_df.iloc[0, 1]
    return vif_df, remove_col, top_vif

 # remove all features when VIF is over 10.
if apply_vif:
    top_vif = 100

    while(top_vif > 5):
        vif_df, remove_col, top_vif = check_vif(train)
        print(remove_col, top_vif)
        if top_vif < 5:
            break
        train = train.drop(columns=remove_col)

    display(train)
```

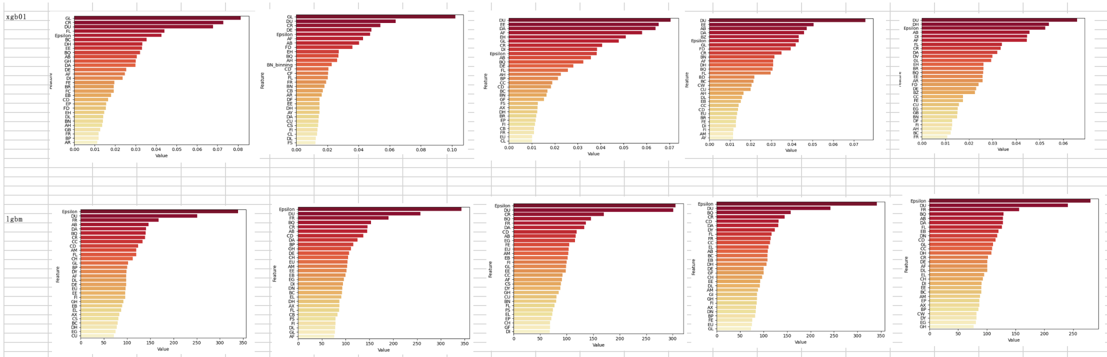
3.1.2 Random Forest for feature importance

```Python
X = train.drop(columns=["Class"])
y = train['Class']

if feature_selection:
    rf_param_grid = {'n_estimators': 100, 'max_depth': 10, 'max_features': 0.7}
    rf = RandomForestClassifier(random_state=42, n_jobs=-1)

    rf.fit(X, y)
    print("Train ACC : %.4f" % accuracy_score(y, rf.predict(X)))
    fi_df = pd.DataFrame({'feature':X.columns,
'importance':rf.feature_importances_})
    selected_cols = fi_df.sort_values(by="importance",
ascending=False)[:m]["feature"].values

    display(selected_cols)

    X = train[selected_cols]
    display(X)
```

3.1.3 XGB

3.1.4 EDA

EJ:column EJ is a redundant column in train data. All information of column EJ is contained in column EH. Whenever column EJ=A then EH=0.003042 and whenever column EJ=B then EH>=0.006084. So we can drop column EJ without losing any train data.



3.2 Feature Crossing

3.2.1.direct crossing

3.1.2.crossing after bucketing

3.3 Feature Importance

xgb_models.feature_importances_

Xgb tuning

Python
```python
import optuna
import xgboost as xgb
#trial.suggest_categorical
#trial.suggest_float
#trial.suggest_int
#binary:logistic
#binary:logitraw
#1. Define an objective function to be maximized.
def objective(trial):
# 2. Suggest values of the hyperparameters using a trial object.
    params = {
    'n_estimators' : trial.suggest_int('n_estimators',2000,3000),
    'max_depth':  trial.suggest_int('max_depth',3,8),
    'min_child_weight': trial.suggest_float('min_child_weight', 2,4),
    "learning_rate" : trial.suggest_float('learning_rate',1e-4, 0.2),
    'subsample': trial.suggest_float('subsample', 0.2, 1),
    'gamma': trial.suggest_float("gamma", 1e-4, 1.0),
    "colsample_bytree" : trial.suggest_float('colsample_bytree',0.2,1),
    "colsample_bylevel" : trial.suggest_float('colsample_bylevel',0.2,1),
    "colsample_bynode" : trial.suggest_float('colsample_bynode',0.2,1),
    }
    xgbmodel_optuna = XGBClassifier(**params,random_state=seed,tree_method
= "gpu_hist",eval_metric= "auc")
```

```
    xgbmodel_optuna.fit(X,y)
    cv = cross_val_score(xgbmodel_optuna, X, y,
cv=4,scoring='neg_log_loss').mean()
    return cv
 # 3. Create a study object and optimize the objective function.
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100,timeout=1200)
```

4. Competition Records

(Final model→ Daily To-do->Submitted Models →Tried Neural Network Methods → Loss Function Metrics → Complementary Dataset→ Post-processing → Experiment Results → Useful Resources)

Final model:
lgbm cv 0.17 lb 0.16

Tabpfn cv 0.23 lb

Xgb3 cv 0.17-0.19   lb

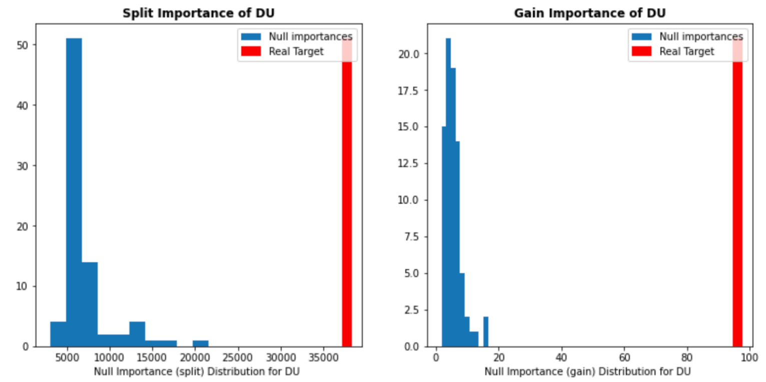
Daily To-do

date	question	todo
0708	Xgb tuning; smote; regu; binning	
0709	note	BQ col= age
0720	1. Feature importance(including age) 2. Outlier deletion, cv score	1. Prediction result visualization 2. BN w/o normalization 3. Cv 0.08 0.06 4. TabPFN—cv- tuning 5. Feature importance analysis 6. Discussion board

note

```
01)
=Config.lr,weight_decay=Config.weight_decay)
```

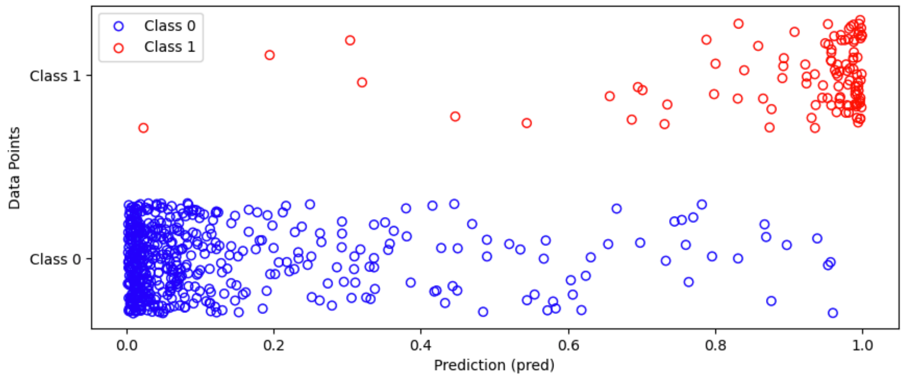
```
display_distributions(actual_imp_df_=actual_imp_df, null_imp_df_=null_imp_df, feature_='DU')
```



Submitted

date	model	cv	lb
0728	3xgb+lgbm+post（0.86， 0.14)+oversample(外部）+pred_alpha	0.0312	0.16
0727	3xgb+lgbm+tabpfn+post（0.86, 0.14)+pre_class	0.53	0.14
	3xgb+lgbm+tabpfn+post（0.86, 0.14)+pre_class+oversample(inner)	0.47	0.14
0726	3xgb+lgbm+pred_class	**0.1685**	**0.17**
	3xgb+age_col feature augmentation	0.17	0.19
	3xgb	0.21	0.21
0731	3xgb+lgbm+ change **missing value imputation method + internal oversampling + pred_class**	**0.1659**	**0.19**
0801	3XGBoost models + LightGBM + Change missing value imputation method + internal oversampling + predict class + add column (whether BQ is empty); known that when BQ is	0.1634	

	empty, class is 0		
	3 XGBoost models + LightGBM + Change missing value imputation method + internal oversampling + predict class + add column (whether BQ is empty; known that when BQ is empty, class is 0) + remove EJ.	**0.1580**	0.19
	3XGBoost models + LightGBM + TabNet + Change missing value imputation method + internal oversampling + predict class + add column (whether BQ is empty; known that when BQ is empty, class is 0) + remove EJ.	0.1628	



Unsubmitted local comparison

date	model	cv	comment
0728	3xgb+lgbm+oversample(inner) +pred alpha	0.1850	
	3xgb+lgbm+oversample(inner) +pred class	0.1741	**scale_pos_weight= 4.71**
	3xgb+lgbm+pred alpha	0.1905	
	3xgb+lgbm+pred class	0.1685	
	3xgb+lgbm+tabpfn+overwrite_warning =True+pred class	0.1769	
	3xgb+lgbm+tabpfn+pred class	0.1769	

NN

1、tabnet

date	model	cv	figure
0730	tabnet	0.2478	
	wide&deep	0.3241	
	wide&deep+date	0.3231	
	wide&deep+DU binning	**0.2676**	
	wide&deep+DU binning+check BQ null	0.2541	

Issue

1. Currently, whether it's XGBoost or TabNet, the loss during training is AUC, not our target metric of balanced log loss. Will this have an impact?

2.

Posted 6 days ago · Posted on Version 11 of 11

Hey,

How did you know to return the best model of the cv splits instead of training the model on the whole training set (after checking the CV score)?

I trained this model on the whole training set (ros) and the LB was 0.19.

Reply

AI Andrij Posted 5 days ago · Posted on Version 11 of 11 TOPIC AUTHOR

I kept this from the original notebook. Yes I got similar result with yours. That is why I claim that the result is overfitted. But who knows

Reply

3. fitting with all date or other
4. divide into n folds and choose the model that fits best on those n folds to be the final model
5. TBD: looks like fitting based on the last fold each time

Loss function

Balanced Log Loss Explained

https://www.kaggle.com/competitions/icr-identify-age-related-conditions/discussion/422442

Many discussion posts and many notebooks (including all the highest scoring public notebooks with LB = 0.06) use an incorrect balanced log loss formula

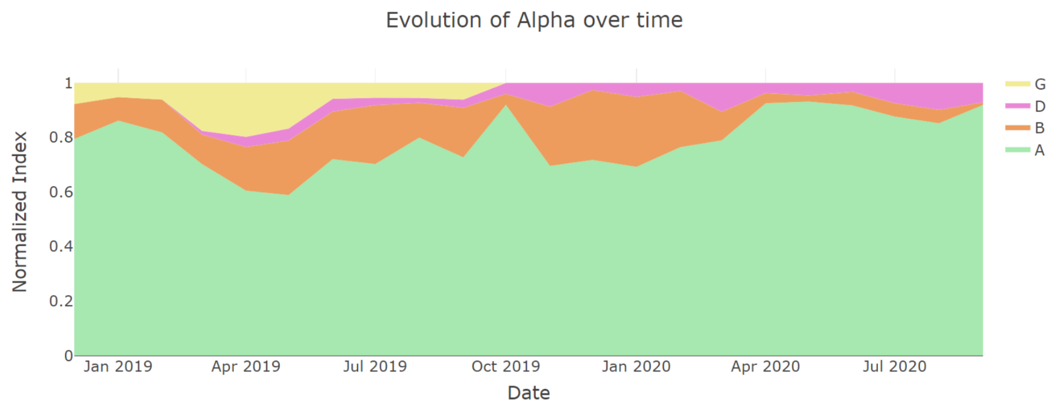
Greeks data

4

For dates labeled as 'Unknown,' the Class is always 0.

	count	mean
yy		
2012	2	0.000000
2014	7	0.571429
2016	1	1.000000
2017	1	1.000000
2018	51	0.372549
2019	232	0.284483
2020	179	0.094972
Unknown	144	0.000000

2、As time passes, both G and B are decreasing. Why adding time to the test set would improve lb & cv.



greeks.Alpha and class

Alpha as a label for stratification, predicting class. ❌

Use alpha as a label for stratification, predict alpha, and calculate the accuracy of the class

Post-processing

Find threshold

1. predict class

	1	2	3	4	5
后处理	0.12731669832	0.096960668	0.1024347719	0.10165025524	0.13
non-postprocessi	0.16940620482	0.1435341768	0.3226167816	0.26167892656	0.19

Click the image to view the sheet.

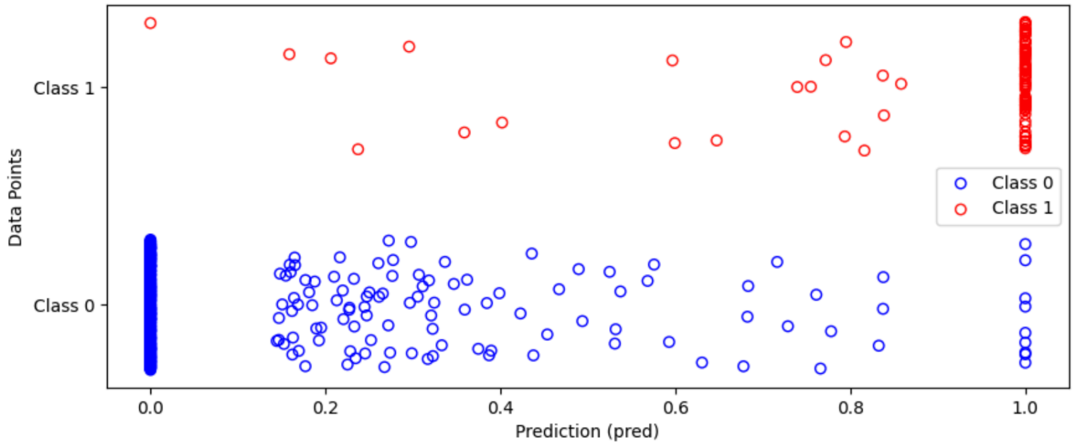
use lb0.06 reference

2. 3xgb+lgbm ：

preds[preds > 0.86] = 1

preds[preds < 0.14] = 0

test_local_cv: [0.5655853642925102, 0.4071467407172298, 1.050574755207553, 0.3216936238663481, 0.4697278935147276]
test_Mean local_cv: 0.5629456755196738



Experiment result

1. 7.3 base_model xgb&lr

5-fold：random_state=40

Xgb

random_state=2018	1	2	3	4	5
train	0.01077743	0.01081776	0.01036247	0.01041261	0
test	0.14278312	0.342354	0.68670688	0.47820438	0

Click the image to view the sheet.

7.6 add class weight

random_state=2018 scale_pos_weight=4.71	1	2	3	4
train	0.003719216	0.003721784	0.003668091	0.00361547
test	0.147055704	0.391139414	0.60542986	0.48874494

Click the image to view the sheet.

7.6 Concatenate timestamp (greeks data) and fill in missing time based on knn

random_state=2018	1	2	3	4
train	0.010638742	0.010940443	0.0105856	0.01052099
test	0.221994874	0.329986663	0.673071533	0.52497244

Click the image to view the sheet.

random_state=2018 scale_pos_weight=4.71	1	2	3	4
train	0.00362558	0.00376731	0.003633349	0.00368726
test	0.142031723	0.332132937	0.590785685	0.43507424

Click the image to view the sheet.

LR

random_state=2018	1	2	3	4	5
train	0.27278933592	0.3166995038	0.66190212	0.62160373	0
test	0.74473898	0.85324665	0.71035313	1.11542062	0

Click the image to view the sheet.

2. 7.7 base_model xgb

	1	2	3	4	5
train	0.008604902	0.008800244	0.008501293	0.008459605	0.008779566
test	0.155429558	0.225886882	0.385359589	0.303962305	0.197305064

Click the image to view the sheet.

3. 7.7 base_model TabPFN

	1	2	3	4	5
train	0.03615195	0.036904294	0.035600958	0.036051742	0.034138018
test	0.185073664	0.156581786	0.435751442	0.323056886	0.267471521

Click the image to view the sheet.

4. 0.5 xgb 0.5tabpfn

	1	2	3	4	5
train	0.023131628	0.023341732	0.02226455	0.023520882	0.021914571
test	0.146299786	0.137509774	0.323040864	0.279781237	0.217247675

Click the image to view the sheet.

5. 0.5 xgb 0.5tabpfn，greeks[alpha] as feature

But when calculating the loss, use the class, that is, add up the last three probabilities of the predicted alpha as the probability for class 1

	1	2	3	4	5
train	0.016833836	0.019621811	0.014889674	0.017429469	0.023452402
test	0.355563402	0.13949585	0.271312494	0.149551264	0.156900803

Click the image to view the sheet.

Useful Resource

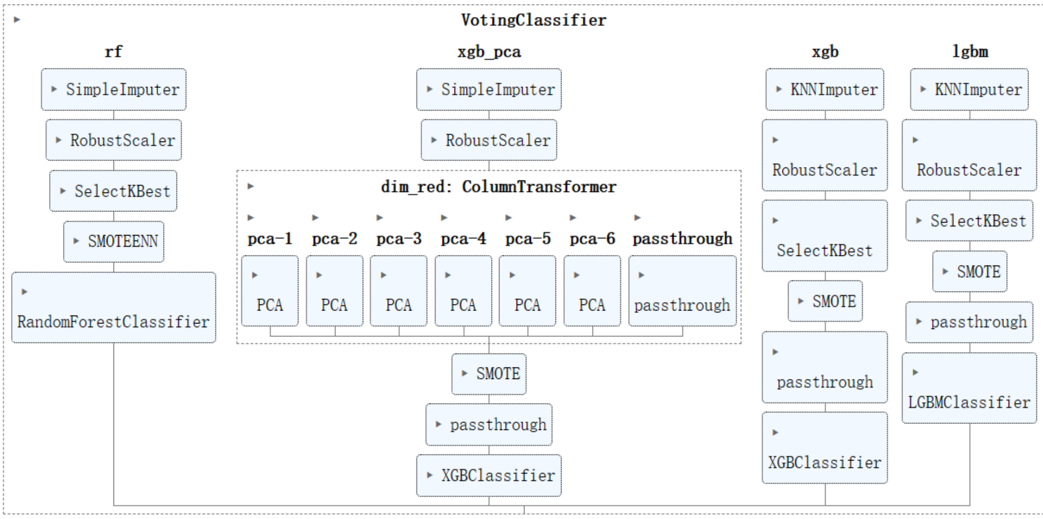
> svm machine and lr model didn't perform well. & svc 코드 파라미터를 확인해봐야할 것 같음.
>
> greek data doesn't improve performance. ----⟶ use it only for EDA
>
> k-neighbors in sampling_method could be change
>
> GAN(Generative Adversarial Network) method could be useful ⇒ generating a similar samples amout of minoirity class
>
> LGBM & TabPFN don't need to be preprocess of scaling.
>
> K-fold could use only 3~4. 5 makes performance dropping
>
> if using greeks as y label? it could be useful to use multi-labelstratified k fold
>
> use GPU PLZ when you are using deep learning model (cuml)
>
> need EDA for greeks data

[Fork of SVC+RF+LR+XGB with Auto]

https://www.kaggle.com/code/samuelpark97/fork-of-for-beginner-svc-rf-lr-xgbwith-auto

https://www.kaggle.com/code/moritzm00/icr-xgb-lgbm-pipeline-hyperparameter-tuning

]:



VotingClassifier
rf
SimpleImputer
RobustScaler
SelectKBest
SMOTEENN
RandomForestClassifier
xgb_pca
SimpleImputer
RobustScaler
dim_red: ColumnTransformer
pca-1
pca-2
pca-3
pca-4
pca-5
pca-6
passthrough
PCA
PCA
PCA
PCA
PCA
PCA
passthrough
SMOTE
passthrough
XGBClassifier
xgb
KNNImputer
RobustScaler
SelectKBest
SMOTE
passthrough
XGBClassifier
lgbm
KNNImputer
RobustScaler
SelectKBest
SMOTE
passthrough
LGBMClassifier