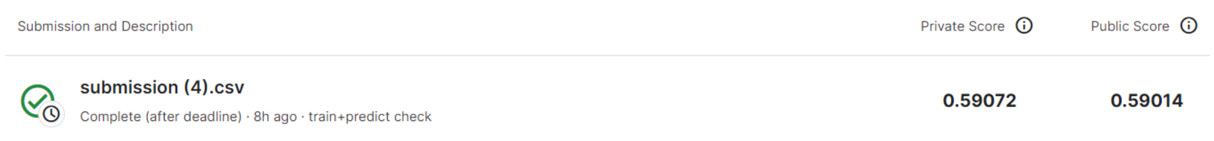
NYCU Introduction to Machine Learning, Final Project

Result in Scoreboard



Submission and Description	Private Score	Public Score
submission (4).csv Complete (after deadline) · 8h ago · train+predict check	0.59072	0.59014

environment details

- Run in google colab (latest version)
- `Python 3.8.16`
- `tensorflow==2.9.2`
- `pandas==1.3.5`
- `numpy==1.21.6`
- `sklearn==1.0.2`
- `category-encoders==2.5.1.post0`
- `optuna==3.0.5`
- `pickle==4.0`

Github link

https://github.com/ben900926/Intro_to_ML_final_project

Model Weight Link

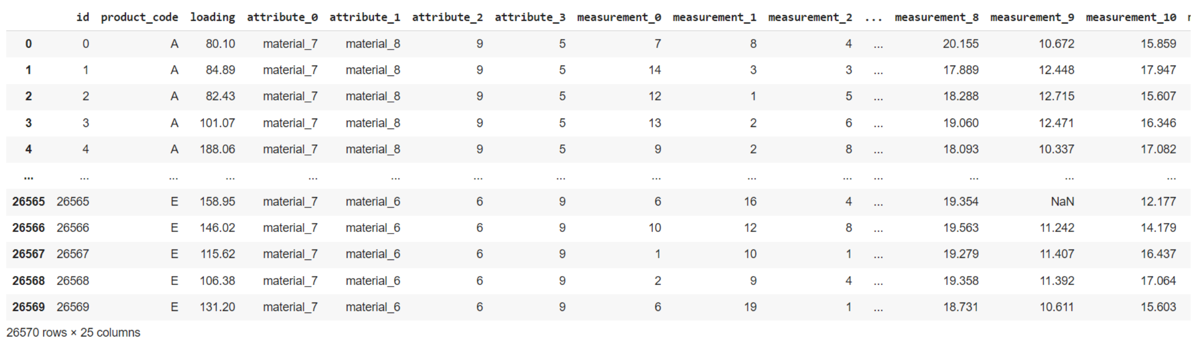
https://drive.google.com/file/d/1-5d3PfMkJr7In3xu5xQbZQsqlqqInJbq/view?usp=sharing

Reference

TPS 08/22 - Hyperparameter Search | Kaggle
(learn how to use optuna package)
AUC Custom Loss Function (Top 4%) | Kaggle
(idea of impututaion of features, and custom loss function)

Introduction

In this task, we are given a series of measurement and attribute values in the train set. We then are asked to predict the probability of failure for each product id, and submission are evaluated based on the area under the ROC curve between the predicted probability and the observed target. Below shows the rough look of the data (train.csv)



	id	product_code	loading	attribute_0	attribute_1	attribute_2	attribute_3	measurement_0	measurement_1	measurement_2	...	measurement_8	measurement_9	measurement_10
0	0	A	80.10	material_7	material_8	9	5	7	8	4	...	20.155	10.672	15.859
1	1	A	84.89	material_7	material_8	9	5	14	3	3	...	17.889	12.448	17.947
2	2	A	82.43	material_7	material_8	9	5	12	1	5	...	18.288	12.715	15.607
3	3	A	101.07	material_7	material_8	9	5	13	2	6	...	19.060	12.471	16.346
4	4	A	188.06	material_7	material_8	9	5	9	2	8	...	18.093	10.337	17.082
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
26565	26565	E	158.95	material_7	material_6	6	9	6	16	4	...	19.354	NaN	12.177
26566	26566	E	146.02	material_7	material_6	6	9	10	12	8	...	19.563	11.242	14.179
26567	26567	E	115.62	material_7	material_6	6	9	1	10	1	...	19.279	11.407	16.437
26568	26568	E	106.38	material_7	material_6	6	9	2	9	4	...	19.358	11.392	17.064
26569	26569	E	131.20	material_7	material_6	6	9	6	19	1	...	18.731	10.611	15.603

26570 rows × 25 columns

As we can see, the data itself is quite noisy. Some of the data are missing (train set has 20273 missing values, whereas test set has 15709 missing), therefore we need to do preprocessing before training the model. I tried two kinds of network: fully connected network and logistic regression model. The later works better with the help of parameter search using optuna module.

Methodology

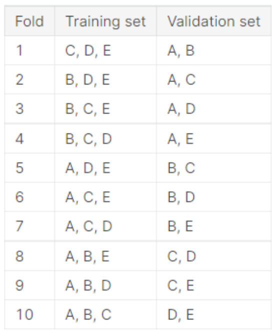
To compare which method suits this task better, I tried two different methods: dense layer network and logistic regression. Each of the network requires different approach to preprocess the data. I will discuss them in detail below respectively.

1. Logistic Regression with various imputation (private score: 0.58731)

This is the first version of my method, and it is different from the one I hand in.

Cross Validation

To validate the prediction effectively and mimic the true testing process, that is, test on a completely different set, I use two product codes for validation and the remaining three as training set (See leftest figure). Using two product codes instead of just one ensures the model is able to generate predictions for multiple codes that rank well. If only one code is used for validation, the result would be poor and is not generalized for different code.



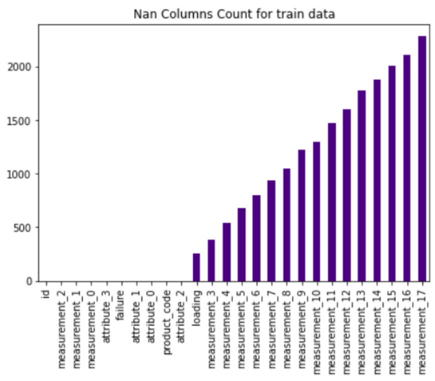
Fold	Training set	Validation set
1	C, D, E	A, B
2	B, D, E	A, C
3	B, C, E	A, D
4	B, C, D	A, E
5	A, D, E	B, C
6	A, C, E	B, D
7	A, C, D	B, E
8	A, B, E	C, D
9	A, B, D	C, E
10	A, B, C	D, E

Impute “loading” using Gaussian probability density function

There are many features in the training set, but not all of them is suitable for training. The below figure shows Nan counts of each attribute. Among all features, “loading” is the most important feature. With “loading” as the only feature, a logistic regression model can produce area under ROC curve of 0.58. As my first trial, I tried to impute loading with the loading value of where probability density functions overlap for loading where failure is 0 and where failure is 1.To be more specific, I assumed log of “loading” sample from a normal distribution. This can be proven by normality hypothesis test with a null hypothesis that the log-loading came from a normal distribution (using scipy’s normal test function).

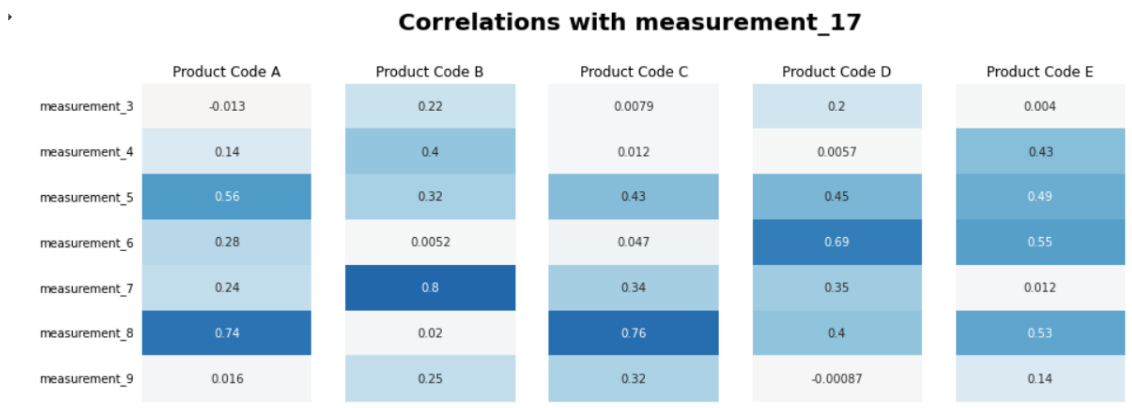
```
Normality test p-values for loading distributions
loading for both targets: 0.68
loading where target=0: 0.73
loading where target=1: 0.98
```

We can see that p-value of “loading” is beyond significance level (0.05), so fail to reject the null hypothesis. Notably, “loading” is the only attribute that satisfies the condition. We still have to impute other features using other methods.



Impute measurements 3 through 9 and 17 using correlations

I referred to some discussions on Kaggle such as the one made by Michael Briant , this inspires me how to impute other features, so that I didn't have to implement all of the features while training network.
To do so, we can instead generate a new feature using a sum product of measurement 3 to 9 and 17. Below shows the correlation between measurements 3 to 9 and 17 by product code:



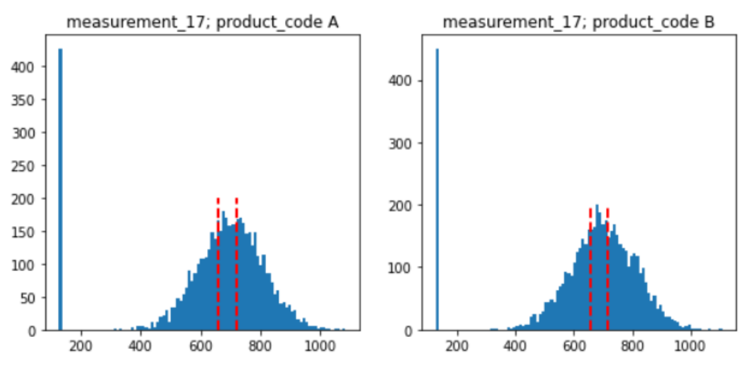
Correlations with measurement_17

	Product Code A	Product Code B	Product Code C	Product Code D	Product Code E
measurement_3	-0.013	0.22	0.0079	0.2	0.004
measurement_4	0.14	0.4	0.012	0.0057	0.43
measurement_5	0.56	0.32	0.43	0.45	0.49
measurement_6	0.28	0.0052	0.047	0.69	0.55
measurement_7	0.24	0.8	0.34	0.35	0.012
measurement_8	0.74	0.02	0.76	0.4	0.53
measurement_9	0.016	0.25	0.32	-0.00087	0.14

The new feature has the following expression:

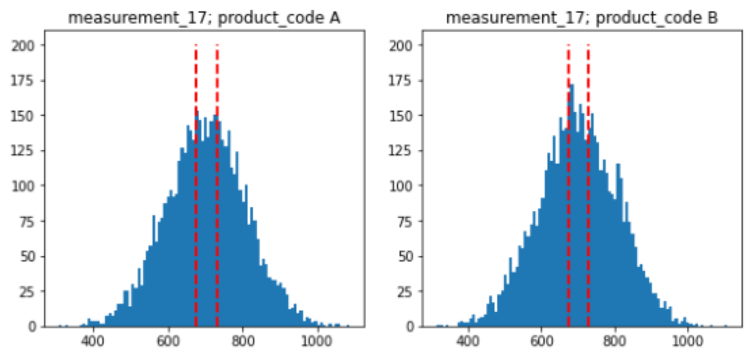
$$Y = \sum_{i=3}^{9} C_i X_i I(C_i \geq 0.1)$$

where Ci is correlation between measurement 17 and Xi is the value, and correlation should larger than 0.1. Then I used this new feature to impute measurement 17 using linear regression. I found that the new feature is highly correlated with measurement 17.
As for remaining Nan values, I imputed them using KNNimputer (neighbor=3) and HuberRegressor (epsilon=1.35, max_iter=500). The reason I used HuberRegressor is that this dataset is pretty noisy and contains many outliers, such as the below figure shows. There're obvious outliers in the left of the quantile.



Scale

Also, I scale the features to make the data fit the linear regression model more, since this kind of model is sensitive to outliers. Attribute 3 is scaled by standard scaler (make mean=0, variance=1) since there is only one value of it for each product code; others are scaled using robust scaler. Below shows the data range after scaling by quantile ranges found by cross validation. You can see that the outliers in the previous figure are gone now.



Custom AUC curve loss function (reference: Optimizing Classifier Performance via an Approximation to the Wilcoxon-Mann-Whitney Statistic)

Instead of the use of “binary crossentropy” loss function, just like most of the binary classification task, I found that following the idea of this paper will produce the better result. The paper introduced a new concept called WMWstatistic, which is computed below:

$$U = \frac{\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} I(x_i, y_j)}{mn}$$

$$I(x_i, y_j) = \begin{Bmatrix} 1 : x_i > y_j \\ 0 : otherwise \end{Bmatrix}$$

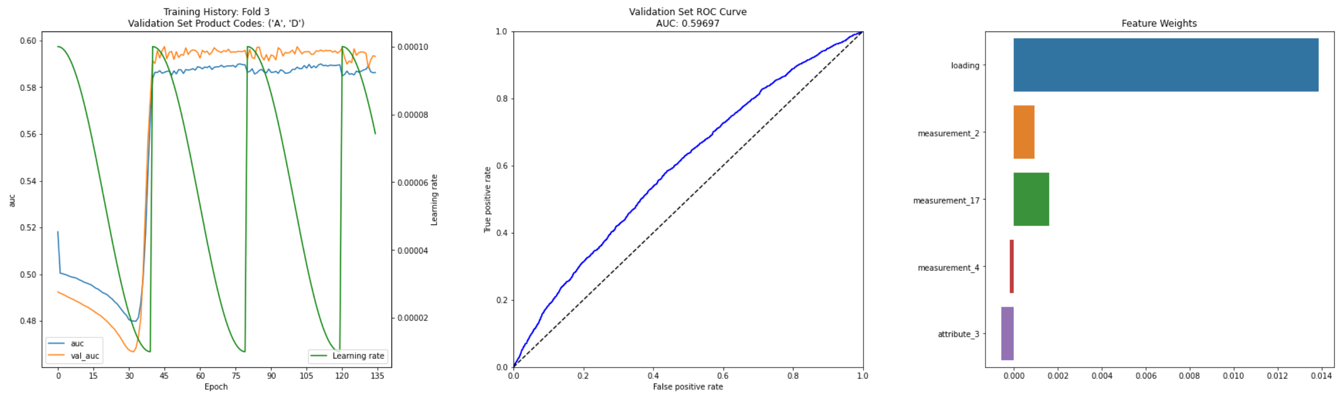
where m, n is the number of positive and negative predictions. When we maximize U, we are increasing the area under ROC curve at the same time. The paper suggests using the approximation indicators:

$$R(x_i, y_j) = \begin{Bmatrix} (-(x_i - y_j - \gamma)^p) : x_i - y_j < \gamma \\ 0 : otherwise \end{Bmatrix}$$

there are two parameters: gamma and p. Following the suggestion from paper, I chose gamma as 0.1 and p as 3.

Define Model

I used cosine decay to schedule learning rate. It combines the advantages of both large and small learning rate, and is able to converge quickly also converge smoothly. I choose 40 as a cycle in cosine decay. The max learning rate is 1e-3 while the min value is 1e-4. In the training step, learning rate changes between them periodically.
As for the model architecture, I only use a single dense layer. I've tried to use more layers, only resulting in poor results. Before training on the whole training set, I perform cross validation to determine the best parameters. Below is the figure of partition of validation. Here, you can see learning rate changing with a period of 40. Also, the loading weight is the most significant. (leftest figure shows training, validation accuracy and learning rate for each epoch; middle figure is the area under ROC curve on validation set; rightest figure shows weight for each feature)



I also recorded the best epochs with the minimum loss. The mean best epoch is 103, and median is 83. In the past 40 epochs, the model is not converge yet. As a result, I decided to run the model on whole training set for 100 epochs for the best performance.

2. Logistic regression with parameter search (private score: 0.59072)

The previous method is not good enough to pass the baseline. Then I remember doing parameter search in homework 4 to obtain the best result for SVM. In this task, I am using the similar model, so parameter search should help to get the better result.

Preprocessing

In this method, I perform the similar preprocessing steps compared to my previous method. Except for I take more features into consideration, such as “m3_null” recording if measurement 3 of each ID is Nan or not. This idea is inspired by Robert Sizemore. If I did not include these features, the model cannot get enough information to pass the baseline (only 0.588) ; also, concatenate both training and testing set while imputation process also help to raise the score.

Also, I use some transformer to process column data. For example, a WOEencoder to transform categorical data (attribute 0) to binary data, and apply log function to “loading” attribute. Using this process to replace the former scale function. Also, I use a percentile prune to remove extreme values.

Model Architecture

In this method, I use logistic regression model from sklearn with l1 penalty. Among l1(0.59072), l2 (0.59019) penalty and XGboost (0.59053), this one has the better performance. Referring to Robert Sizemore, I learn to use optuna package, which is convenient for parameter search. Starting from C=1.0, optuna will search between 1e-10 to 100 for best performance. In the end, I use the model with the parameter I found (C= 0.1635393159306732).

Summary

In this report, I discussed two methods used for the task. The first method focuses on several preprocessing process, including imputation using Gaussian probability density function, correlation and scale function with quantiles. The score is below baseline, however. The current method I am using features parameter search for logistic regression with l1 penalty, which can obtain the best performance proven by experiments. The state-of-the-art method depends on the right parameters more, instead of only novel preprocessing steps.