Transformer model: Competition 2

Introduction

In the competition, the goal is to achieve an F1 score over 55%. To achieve this goal, different feature selection or feature engineering techniques are applied to handle the dataset. As the dataset is imbalanced, different rebalanced techniques also applied during training the transformer model.

Method

Improve sequence window for training

Using the past 30 days information to predict if the railway will be break.

2 days ago (1s)

```
# build windows
def sequence_windows (X, y, past, future):
    X_seq, y_seq = [], []
    for i in range(past, len(X) - future):
        input_seq = X[i-past:i]
        future_seq = y[i:i+future]
        label = 1 if np.any(future_seq==1) else 0
        X_seq.append(input_seq)
        y_seq.append(label)
    return np.array(X_seq), np.array(y_seq)
X_seq, y_seq = sequence_windows(X, y, 30, 30)
print("X_seq shape:", X_seq.shape)
print("y_seq shape:", y_seq.shape)
```

```
X_seq shape: (19940, 30, 8)
y_seq shape: (19940,)
```

Applying different datasets

When training the transformer model with different datasets, the number of input samples is 10000. Also, the model hyperparameters are d_model=256, nhead=8, num_layers=3, dropout=0.1, lr=3e-4, weight_decay=1e-4.

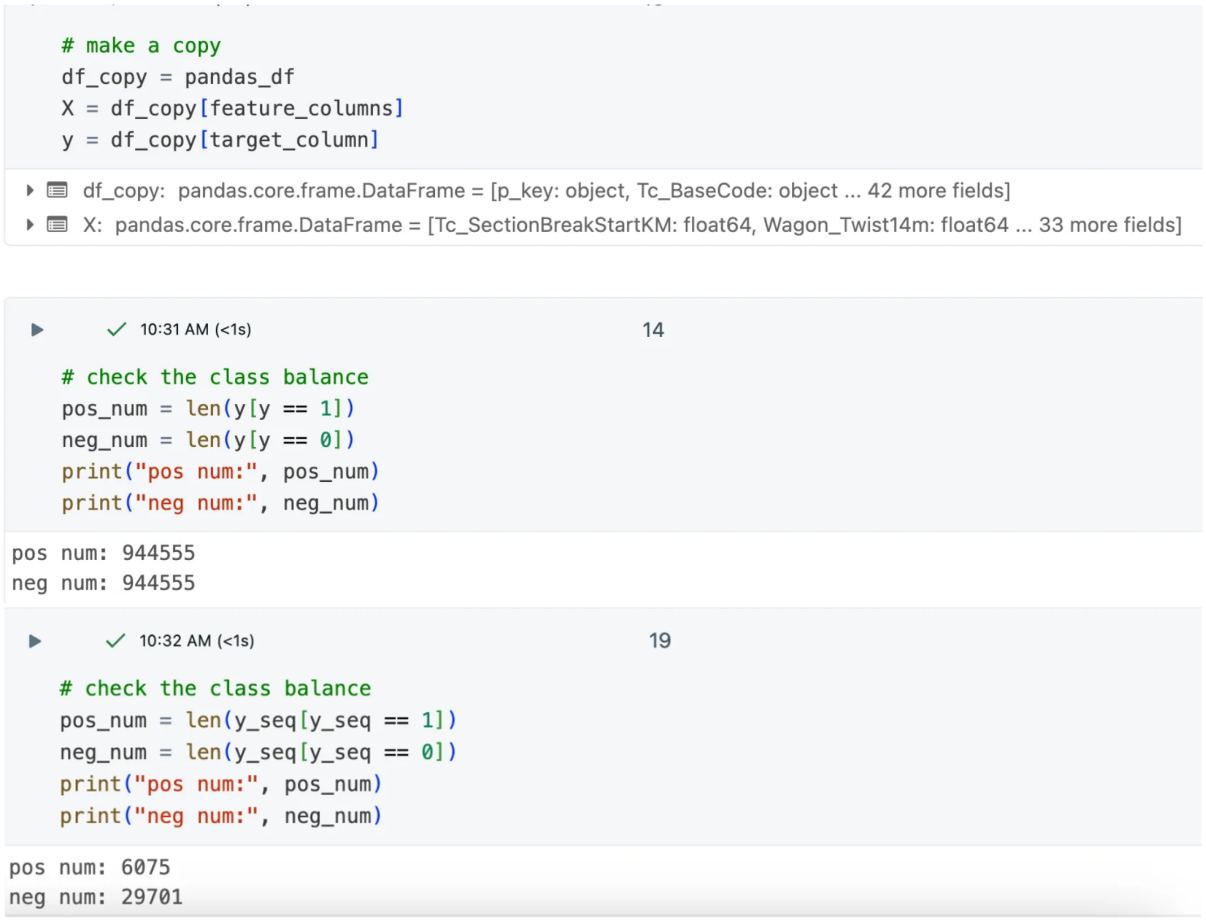
Model	Datasets	Leaderboard score
Transformer	Preprocess training table	Accuracy: 32.84%, AUC_PR: 35.96%, F1_Score: 49.45%
	Fe training	Accuracy: 66.63%, AUC_PR: 34.20%, F1_Score: 13.62%

	Preprocess training with track break date	Accuracy: 32.84%, AUC_PR: 29.51%, F1_Score: 49.45%

Based on the above table, the transformer model with the preprocessing training dataset gets the highest accuracy, AUC_PR, F1_score. Therefore, the following experiments will use the preprocessing training dataset for transformer.

Rebalanced techniques

Due to the imbalanced characteristics of the dataset, our team applied rebalancing techniques for the preprocessing training dataset and the fe_training dataset. When using the stratified_oversample_preprocess_table, the balanced dataset becomes imbalanced after entering the sequence window. Also, the dataset is balanced when inputting the whole dataset, but the computing resources are limited. Therefore, the handling imbalanced dataset techniques will be applied during the model training.



```
# make a copy
df_copy = pandas_df
X = df_copy[feature_columns]
y = df_copy[target_column]
```

```
df_copy:  pandas.core.frame.DataFrame = [p_key: object, Tc_BaseCode: object ... 42 more fields]
X:  pandas.core.frame.DataFrame = [Tc_SectionBreakStartKM: float64, Wagon_Twist14m: float64 ... 33 more fields]
```

```
✓ 10:31 AM (<1s)                                    14
# check the class balance
pos_num = len(y[y == 1])
neg_num = len(y[y == 0])
print("pos num:", pos_num)
print("neg num:", neg_num)
```

```
pos num: 944555
neg num: 944555
```

```
✓ 10:32 AM (<1s)                                    19
# check the class balance
pos_num = len(y_seq[y_seq == 1])
neg_num = len(y_seq[y_seq == 0])
print("pos num:", pos_num)
print("neg num:", neg_num)
```

```
pos num: 6075
neg num: 29701
```

After applying these two techniques, the accuracy and F1_score are the same. However, pos weight has a higher AUC_PR score than the class weight. Therefore, the pos weight method will be applied when training the transformer model.

Rebalanced techniques	**Leaderboard result**
pos weight	Accuracy: 45.16%, AUC_PR: 37.12%, F1_Score: 53.11%
class weight	Accuracy: 45.16%, AUC_PR: 37.07%, F1_Score: 53.11%

Tuning hyperparameters

To find suitable hyperparameters for the transformer model, apply the tuning hyperparameters method. The F1 score is the metric during this competition, so tuning hyperparameters based on it. Also, the early stop mechanism is set to reduce additional training. When the F1 score does not change, it will not train in the next epoch.

When tuning hyperparameters, the number of input samples is 20000. Also, the model hyperparameters are d_model=[128, 256], nhead=[4, 8], num_layers=[2,3], dropout=0.2, lr= 1e-4, weight_decay=3e-4. As the computation resource is limited, if we set epochs to 10, the Python kernel will restart. Therefore, set the epoch to 5 to test. After submission, leaderboard shows Accuracy: 45.16%, AUC_PR: 35.84%, F1_Score: 53.11%.


```
best = tuing_hyperparameters(train_loader, val_loader)
best_model = best['model']
best_thr = best['val_best_thr']
print(f'Best hyperparameters: {best["hparams"]}, Best val f1: {best["val_best_f1"]}, Best threshold: {best_thr}')
```

```
HP 1: hp:{'d_model': 128, 'nhead': 4, 'num_layers': 2, 'dropout': 0.2, 'lr': 0.0001, 'weight_decay': 0.0003, 'epochs': 5},
F1=0.9761 thr=0.0800
HP 2: hp:{'d_model': 128, 'nhead': 4, 'num_layers': 3, 'dropout': 0.2, 'lr': 0.0001, 'weight_decay': 0.0003, 'epochs': 5},
F1=0.9759 thr=0.0000
HP 3: hp:{'d_model': 128, 'nhead': 8, 'num_layers': 2, 'dropout': 0.2, 'lr': 0.0001, 'weight_decay': 0.0003, 'epochs': 5},
F1=0.9759 thr=0.0000
HP 4: hp:{'d_model': 128, 'nhead': 8, 'num_layers': 3, 'dropout': 0.2, 'lr': 0.0001, 'weight_decay': 0.0003, 'epochs': 5},
F1=0.9759 thr=0.0000
HP 5: hp:{'d_model': 256, 'nhead': 4, 'num_layers': 2, 'dropout': 0.2, 'lr': 0.0001, 'weight_decay': 0.0003, 'epochs': 5},
F1=0.9784 thr=0.0400
HP 6: hp:{'d_model': 256, 'nhead': 4, 'num_layers': 3, 'dropout': 0.2, 'lr': 0.0001, 'weight_decay': 0.0003, 'epochs': 5},
F1=0.9793 thr=0.0300
HP 7: hp:{'d_model': 256, 'nhead': 8, 'num_layers': 2, 'dropout': 0.2, 'lr': 0.0001, 'weight_decay': 0.0003, 'epochs': 5},
F1=0.9803 thr=0.0600
HP 8: hp:{'d_model': 256, 'nhead': 8, 'num_layers': 3, 'dropout': 0.2, 'lr': 0.0001, 'weight_decay': 0.0003, 'epochs': 5},
F1=0.9795 thr=0.1500
Best hyperparameters: {'d_model': 256, 'nhead': 8, 'num_layers': 2, 'dropout': 0.2, 'lr': 0.0001, 'weight_decay': 0.0003,
'epochs': 5}, Best val f1: 0.9802802283341983, Best threshold: 0.06
```

Select threshold

There is the same result on a fixed threshold (0.5) and finding the best threshold based on the F1 score. To simplify the training process, the fixed threshold (0.5) method will be applied.

Method	**Leaderboard result**
Fixed threshold (0.5)	Accuracy: 45.16%, AUC_PR: 44.17%, F1_Score: 53.11%
F1 score	Accuracy**:** 45.16%, AUC_PR**:** 44.17%, F1_Score: 53.11%

Overfitting problems

- **Increase penalty and reduce layers**

Reduce the model dimension from 256 to 128, nhead from 8 to 4, and num_layers from 3 to 2. Also, increase the dropout from 0.1 to 0.2. After submission, the F1 score remains 53.11%.

- **Reduce the number of features**

In the initial version, selecting 35 features from the preprocessing training table. To mitigate the overfitting problem, only select 8 core features based on the EDA. The leaderboard shows the AUC_PR score has increased from 36.53% to 54.09%, but the F1_score is still 53.11%.

Submission inference

We create a window for inference submission. The original time-series data is converted into a fixed length for model input. Every entry is grouped by group_key. We first sort the data based on time, remove the duplicate rows, and keep the latest record. Also, we build a entire calendar between the minimium and maximum dates and filled in missing values by median to ensure the dates are entired. For every true observation date, we select features from L date as an inference window. Every window has a unique key with the input shape (N, L, D) for model inference.

Conclusion and Improvement.

The highest F1 score for the transformer model is 53.11% during the competition2. This indicates there is an overfitting problem as the test F1 score always over 80%, but the F1 test score always around 50%. To mitigate this problem, the next step is to increase penalty for

model. Meanwhile, our team will further optimize, analyze, and deal with the datasets. Also, the recorded data may have some mistakes, leading to a wrong score on the leaderboard. Therefore, some tests may need to be retested.