CS116 Technical Report

Team members

- **Bùi Hồng Sơn** - 22521246
- **Đồng Minh Quân** - 22521176
- **Nguyễn Minh Sơn** - 22521254

Introduction

The objective of the competition hosted on Kaggle Home Credit, is to develop a predictive model that can accurately determine the likelihood of clients defaulting on their loans. The evaluation criteria for this competition are designed to favor solutions that demonstrate stability over time.

Data and Feature Engineering

Data Overview

Our team divided the task of reading through the feature definitions to extract valuable insights but because of the lack understanding about finance we don't get much values from it.

We overview the data and make some plot. Our analysis led us to several key findings:

1. The distribution of the target column is heavily unbalanced, with 97% of the values being 0.
2. The COVID-19 pandemic appears to have significantly affected the number of observations.
3. Some columns have a high percentage of null values.

We aggregated data based on this baseline notebook: Home Credit Baseline.

Feature Engineering

Unfortunately, we did not allocate enough time to this critical step. Our feature engineering was based on the Home Credit baseline. We tried to modify the `filter_cols` function to exclude columns with more than 98% null values, but this resulted in an Out of Memory (OOM) error. We attempted to resolve this by first reading the train set, training the model, and then reading the test set, but the issue persisted.

Feature Selection

We implemented a ridge regression to identify the most significant features. In addition to the Ridge regression, we also used the LightGBM model to identify the most significant features.

Ridge Regression

We used the following approach for ridge regression:

```
X = df_train.drop(columns=["target", "case_id"])
y = df_train["target"]
from sklearn.linear_model import Ridge
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

# Define preprocessor
int_types = ['int8', 'int16', 'int32', 'int64', 'uint8', 'uint16', 'uint32', 'uint64']
float_types = ['float32', 'float64']

numerical_cols = X.select_dtypes(include=int_types + float_types).columns
categorical_cols = X.select_dtypes(include=['object']).columns

# Define imputers: mean imputation for numerical columns and most frequent for categorical column
num_imputer = SimpleImputer(strategy='mean')
cat_imputer = SimpleImputer(strategy='most_frequent')

# Define preprocessor
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[('imputer', num_imputer), ('scaler', StandardScaler())]), numeric
        ('cat', Pipeline(steps=[('imputer', cat_imputer), ('encoder', OneHotEncoder())]), categor
    ])

clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('classifier', Ridge())])

clf.fit(X, y)
ridge_estimator = clf.named_steps['classifier']

# Get the coefficients
coefficients = ridge_estimator.coef_
feature_importance = [(feature, coef) for feature, coef in zip(X.columns, coefficients)]
features_df = pd.DataFrame(feature_importance, columns=['features', 'coef'])
features_df['abs_coef'] = features_df['coef'].abs()
```

```
top_features = features_df.nlargest(300, 'abs_coef')
top_feature_names = top_features['features'].values
```

LightGBM

We used the following approach for LightGBM:

```
from lightgbm import LGBMClassifier

model = LGBMClassifier()
model.fit(df_train_selected, y)

# Get feature importances
feature_importances = model.feature_importances_

# Get feature names
feature_names = model.feature_name_

# Create a DataFrame
df_feature_importances = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
})


# Sort by importance
df_feature_importances = df_feature_importances.sort_values('Importance', ascending=False)

top_features = df_feature_importances['Feature'].head(300)
```

Model

Our team copied this notebook and tried to improve it: Credit Risk Prediction with LightGBM and CatBoost. We did not invest much effort into tuning the hyperparameters. The most impactful modification was increasing the number of splits in `StratifiedGroupKFold` from 5 to 10, which improved our score by 0.002.

Evaluation

The competition has its own unique metric for evaluating submissions. Therefore, we decided to create and use a custom metric instead of the standard AUC. We referred to this notebook for this purpose: Home Credit Credit Risk Model Stability Discussion.

```python
def gini_stability_custom_metric(y_pred, y_true, week):

    '''
    :param y_pred:
    :param y_true:
    :param week:
    :return eval_name: str
    :return eval_result: float
    :return is_higher_better: bool
    '''

    w_fallingrate = 88.0
    w_resstd = -0.5

    base = pd.DataFrame()
    base['WEEK_NUM'] = week
    base['target'] = y_true
    base['score'] = y_pred
    gini_in_time = base.loc[:, ["WEEK_NUM", "target", "score"]]\
        .sort_values("WEEK_NUM")\
        .groupby("WEEK_NUM")[["target", "score"]]\
        .apply(lambda x: 2*roc_auc_score(x["target"], x["score"])-1 if len(np.unique(x["target"])) 

    x = np.arange(len(gini_in_time))
    y = gini_in_time
    a, b = np.polyfit(x, y, 1)
    y_hat = a*x + b
    residuals = y - y_hat
    res_std = np.std(residuals)
    avg_gini = np.mean(gini_in_time)

    final_score = avg_gini + w_fallingrate * min(0, a) + w_resstd * res_std

    return 'gini_stability', final_score, True
```

Metric Hack

We saw a discussion about this problem and decided to give it a try. It improved our score on the public test but did not significantly impact on the private set.

Explaination

The evaluation metric involves calculating a Gini score for each week, fitting a linear regression through these scores, and penalizing models with decreasing Gini scores over time (negative slope). The standard

deviation of the residuals from the regression is also penalized.

A condition is established based on the week number, and the score is reduced by 0.02 (and clipped at 0 to ensure the scores remain within valid bounds). This strategy aims to enhance the stability of the model's predictions over time by lowering the scores in the earlier weeks. The hope is that this will result in a smaller standard deviation, thereby reducing the penalty score.

```
X_test: pd.DataFrame = df_test.set_index("case_id")
X_test[cat_cols] = X_test[cat_cols].astype("category")

y_pred: pd.Series = pd.Series(model.predict_proba(X_test)[:, 1], index=X_test.index)

df_subm["score"] = y_pred
df_subm["WEEK_NUM"] = df_test.set_index("case_id")["week_num"]

display(df_subm)

condition = df_subm["WEEK_NUM"] < (df_subm["WEEK_NUM"].max() - df_subm["WEEK_NUM"].min())/2 + df_
df_subm.loc[condition, 'score'] = (df_subm.loc[condition, 'score'] - 0.02).clip(0)

df_subm = df_subm.drop(columns=["WEEK_NUM"])

df_subm.to_csv("submission.csv")
```

What Didn't Work

1. Adding `class_weight='balanced'` resulted in a decrease in the score.
2. Using a custom metric instead of AUC.
3. Using 300 most important features based on ridge regression and LightGBM.
4. Filtering null values with a threshold of 0.98, which gave us more than 100 new features but failed due to OOM issues.

Conclusion

While we achieved some improvements, our overall approach faced several challenges, particularly with handling the large dataset and optimizing feature engineering. Future efforts should focus on more robust feature selection and feature engineering to enhance model performance.

The highest notebook score we have is about 0.513. Then a notebook This is the way appear and we have try to improve the score but it stay the same. Finally we got 0.51507 fromm that notebook.