CS116 Technical Report

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

Our team divided the task of reading through the feature definitions to extract valuable insights. We aggregated data based on this baseline notebook: Home Credit Baseline. 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.

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. Here is the code we used for this process:

```
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
```

In addition to the Ridge regression, we also used the LightGBM model to identify the most significant features. Here is the code we used for this process:

```
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.

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.

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.