

# Модель кредитного риск-менеджмента

Выполнил:  
Шишов Дмитрий Геннадьевич

# Анализ входного датасета

Было передано 12 файлов в формате .parquet, в которых хранились 61 признак по 26162717 объектам.

Типы признаков:

- 1) Числовые
- 2) Закодированные
- 3) Бинаризованные



# Подготовка данных к моделированию

- 1) Загрузка данных;
- 2) Подготовка данных к моделированию (дубликаты, пустые значения, кодировка и т.д.);
- 3) Моделирование с параметрами по умолчанию для 1 .parquet файла;
- 4) Тюнинг моделей;
- 5) Моделирование с наилучшими параметрами;
- 6) Выбор финальной модели;
- 7) Финальное моделирование на полном наборе данных.



# Загрузка и преобразование данных

При загрузке каждого паркета происходит преобразование данных в формат int8 для тех столбцов, для которых это ВОЗМОЖНО.

```
def main_df_load(x):  
    df = pd.DataFrame()  
  
    for i in range(x):  
        temp_df = pd.read_parquet(f'train_data/train_data_{i}.pq')  
        for col in temp_df.columns:  
            if np.all((temp_df[col] >= np.iinfo(np.int8).min) & (temp_df[col] <= np.iinfo(np.int8).max)):  
                temp_df[col] = temp_df[col].astype('int8')  
  
        df = pd.concat([df, temp_df])  
  
    del temp_df  
    return df
```

```
df = main_df_load(12)
```

```
df.shape
```

```
(26162717, 61)
```

# Подготовка к моделированию

- 1) Анализ дубликатов;
- 2) Удаление пропусков.

Дубликатов и пустых значений  
в наборе не выявлено

## Удаление дубликатов

```
def drop_duplicates(df):  
    df = df.drop_duplicates()  
    return df
```

```
df = drop_duplicates(df)
```

```
df.shape
```

```
(26162717, 61)
```

## Удаление пустых строк

```
def drop_nan(df):  
    df = df.dropna()  
    return df
```

```
df = drop_nan(df)
```

```
df.shape
```

```
(26162717, 61)
```

# Подготовка к моделированию

## 3) Кодировка категориальных переменных

В кодировке не участвовали бинаризованные столбцы, а также столбцы id и rn.

При кодировке указываем тип выходных данных int8 для экономии памяти.

```
def encoding(df):  
    cols_for_encoder = df.drop(['id', 'rn', 'is_zero_loans5', 'is_zero_loans530',  
                                'is_zero_loans3060', 'is_zero_loans6090',  
                                'is_zero_loans90', 'is_zero_util',  
                                'is_zero_over2limit', 'is_zero_maxover2limit',  
                                'pclose_flag', 'fclose_flag'], axis=1).columns.tolist()  
  
    df = pd.get_dummies(df, columns=cols_for_encoder, dtype='int8')  
    return df
```

```
df = encoding(df)
```

```
df.shape
```

```
(26162717, 411)
```

# Подготовка к моделированию

## 4) Агрегация строк

Теперь, когда все столбцы, кроме первых, двух представлены в виде 0 и 1, можем провести агрегацию через функцию суммы.

Столбец `rn` агрегируем максимальным значением, чтобы получить общее количество продуктов для конкретного пользователя.

Новые значения также переводим в формат `int8`, если это возможно.

```
def agg_func(df):
    max_columns = ['rn']
    sum_columns = df.drop(['id', 'rn'], axis=1).columns.tolist()

    agg_dict = {col: 'max' for col in max_columns}
    agg_dict.update({col: 'sum' for col in sum_columns})

    df = df.groupby('id', as_index=False).agg(agg_dict)

    for col in df.columns:
        if col != 'id':
            if np.all((df[col] >= np.iinfo(np.int8).min) & (df[col] <= np.iinfo(np.int8).max)):
                df[col] = df[col].astype('int8')
            elif np.all((df[col] >= np.iinfo(np.int16).min) & (df[col] <= np.iinfo(np.int16).max)):
                df[col] = df[col].astype('int16')
            else:
                df[col] = df[col].astype('int32')
    return df

df = agg_func(df)

df.shape

(3000000, 411)
```

# Подготовка к моделированию

## 5) Объединение датасета с целевой переменной.

Объединяем два датасета через столбец `id` и используем параметр `inner`, чтобы для каждого пользователя у нас был таргет, и для каждого таргета была история поведения из первого датасета.

```
def merge_targets(df):  
    target_df = pd.read_csv('target/train_target.csv')  
    df = df.merge(target_df, on='id', how='inner')  
    del target_df  
    return df
```

```
df = merge_targets(df)
```

```
df.shape
```

```
(3000000, 412)
```



# Подготовка к моделированию

## б) Генерация новых признаков

Так как столбец `rn` показывает количество продуктов у пользователя, а каждый признак представляет сумму случаев того или иного поведения, мы можем сгенерировать долю каждого поведения относительно количества продуктов.

Новые признаки переводятся в формате `float16`, если это возможно.

```
def new_features(df):  
    for col in df.drop(['id', 'rn', 'flag'], axis=1).columns.tolist():  
        df.loc[:, [f'{col}_to_rn']] = df[col] / df['rn']  
        if np.all((df[f'{col}_to_rn'] >= np.finfo(np.float16).min) & (df[f'{col}_to_rn'] <= np.finfo(np.float16).max)):  
            df[f'{col}_to_rn'] = df[f'{col}_to_rn'].astype('float16')  
        else:  
            df[f'{col}_to_rn'] = df[f'{col}_to_rn'].astype('float32')  
    return df
```

```
df = new_features(df)
```

```
df.shape
```

```
(3000000, 821)
```

# Подготовка к моделированию

## 7) Удаление ненужных столбцов

Удаляем столбец-идентификатор id.

Здесь использовались различные методы отбора признаков (по коэффициенту корреляции, через функции из sklearn, но лучшим образом себя показала регуляризация при моделировании)

```
def drop_columns(df):  
    df = df.drop('id', axis=1)  
    return df
```

```
df = drop_columns(df)
```

```
df.shape
```

```
(3000000, 820)
```

# Подготовка к моделированию

## 8) Разделение данных на тренировочную и обучающую выборки

Данные были разделены в отношении 80/20.

Дисбаланс классов составил примерно 1:30.

```
df_train, df_test = train_test_split(df, test_size=0.2, random_state=42)
x_train = df_train.drop('flag', axis=1)
y_train = df_train['flag']

x_test = df_test.drop('flag', axis=1)
y_test = df_test['flag']

len(y_train[y_train == 0]) / len(y_train[y_train == 1])

27.19383259911894
```



# Моделирование с параметрами по умолчанию

Были выбраны 4 базовые модели.

Во избежании переобучения использовалась стратифицированная кросс-валидация, которая выдерживает исходное соотношение классов в фолдах.

В гиперпараметрах моделей устанавливался параметр, регулирующий баланс классов.

```
models = [RandomForestClassifier(class_weight='balanced'),
           LogisticRegression(class_weight='balanced'),
           LGBMClassifier(verbose=0, class_weight='balanced'),
           CatBoostClassifier(verbose=0, class_weights=[1, 30])]

model_metrics = []
for model in models:
    model.fit(x_train, y_train)

    kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    cv_scores = cross_val_score(model, x_train, y_train, cv=kfold, scoring='roc_auc')

    y_pred = model.predict(x_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, model.predict_proba(x_test)[: , 1])

    model_metrics.append({
        'model': (f'{model}'),
        'cv_mean_roc_auc': round(cv_scores.mean(), 3),
        'accuracy': round(accuracy, 3),
        'precision_score': round(precision, 3),
        'recall': round(recall, 3),
        'f1': round(f1, 3),
        'roc_auc': round(roc_auc, 3)
    })

result_df = pd.DataFrame(model_metrics)
```


# Моделирование с параметрами по умолчанию

Для дальнейшего тюнинга гиперпараметров были выбраны 2 модели:

- 1) LightGBMClassifier
- 2) CatBoostClassifier

result_df							
	model	cv_mean_roc_auc	accuracy	precision_score	recall	f1	roc_auc
0	RandomForestClassifier(class_weight='balanced')	0.706	0.969	0.188	0.006	0.012	0.707
1	LogisticRegression(class_weight='balanced')	0.764	0.711	0.067	0.661	0.121	0.753
2	LGBMClassifier(class_weight='balanced', verbos...	0.762	0.747	0.072	0.620	0.129	0.755
3	<catboost.core.CatBoostClassifier object at 0x...	0.732	0.874	0.093	0.361	0.148	0.729

Критерии выбора:

- 1) Скорость обучения;
  - 2) Значения метрик;
  - 3) Перспективность.
- 

# Тюнинг моделей

Основные тестируемые параметры:

- 1) Количество итераций;
- 2) Глубина деревьев;
- 3) Параметры регуляризации.

```
def catboost_tuning(trial):
    params = {
        'iterations': trial.suggest_int('iterations', 100, 500),
        'depth': trial.suggest_int('depth', 3, 12),
        'learning_rate': trial.suggest_loguniform('learning_rate', 1e-3, 1e-1),
        'l2_leaf_reg': trial.suggest_loguniform('l2_leaf_reg', 1e-8, 10.0),
        'random_strength': trial.suggest_uniform('random_strength', 1e-9, 10),
        'bagging_temperature': trial.suggest_uniform('bagging_temperature', 0.0, 1.0),
        'od_type': 'Iter',
        'od_wait': 20,
        'eval_metric': 'AUC',
        'class_weights': {0: 1.0, 1: 30.0}}

    model = CatBoostClassifier(**params)
    model.fit(
        x_train, y_train,
        eval_set=(x_test, y_test),
        use_best_model=True,
        verbose=0)

    y_pred = model.predict_proba(x_test)[: , 1]
    score = roc_auc_score(y_test, y_pred)

    return score

study = optuna.create_study(direction='maximize')
study.optimize(catboost_tuning, n_trials=30)

catboost_best_param = study.best_params
print(f'Лучшие гиперпараметры: {catboost_best_param}')
```

```
def lgbm_tuning(trial):
    params = {
        'boosting_type': 'gbdt',
        'objective': 'binary',
        'metric': 'auc',
        'verbose': -1,
        'class_weight': 'balanced',

        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.2),
        'n_estimators': trial.suggest_int('n_estimators', 100, 300),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.1, 1),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.1, 1)
    }

    model = LGBMClassifier(**params)
    model.fit(x_train, y_train, eval_set=(x_test, y_test))

    y_pred = model.predict_proba(x_test)[: , 1]
    score = roc_auc_score(y_test, y_pred)

    return score

study = optuna.create_study(direction='maximize')
study.optimize(lgbm_tuning, n_trials=30)

lgbm_best_params = study.best_params
print(f'Лучшие гиперпараметры: {lgbm_best_params}')
```

# Выбор финальной модели

Метрики по обеим моделям улучшились. В качестве финальной выбрана модель LightGBMClassifier.

result_df							
	model	cv_mean_roc_auc	accuracy	precision_score	recall	f1	roc_auc
0	RandomForest_default	0.706	0.969	0.188	0.006	0.012	0.707
1	LogisticRegression_default	0.764	0.711	0.067	0.661	0.121	0.753
2	LightGBMClassifier_default	0.762	0.747	0.072	0.620	0.129	0.755
3	CatboostClassifier_default	0.732	0.874	0.093	0.361	0.148	0.729
5	LightGBMClassifier_tuned	0.766	0.732	0.071	0.652	0.128	0.761
6	CatboostClassifier_tuned	0.760	0.735	0.070	0.629	0.126	0.755

```
models = [LGBMClassifier(**lgbm_best_params),
          CatBoostClassifier(**catboost_best_params)]

for model in models:
    model.fit(x_train, y_train)

    kfold = KFold(n_splits=5, shuffle=True, random_state=42)
    cv_scores = cross_val_score(model, x_train, y_train, cv=kfold, scoring='roc_auc')

    y_pred = model.predict(x_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, model.predict_proba(x_test)[: , 1])

    model_metrics.append({
        'model': (f'{model}_tuned'),
        'cv_mean_roc_auc': round(cv_scores.mean(), 3),
        'accuracy': round(accuracy, 3),
        'precision_score': round(precision, 3),
        'recall': round(recall, 3),
        'f1': round(f1, 3),
        'roc_auc': round(roc_auc, 3)
    })

result_df = pd.DataFrame(model_metrics)
```

# Моделирование на полном объеме данных

```
model = LGBMClassifier(**final_lgbm_best_params)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

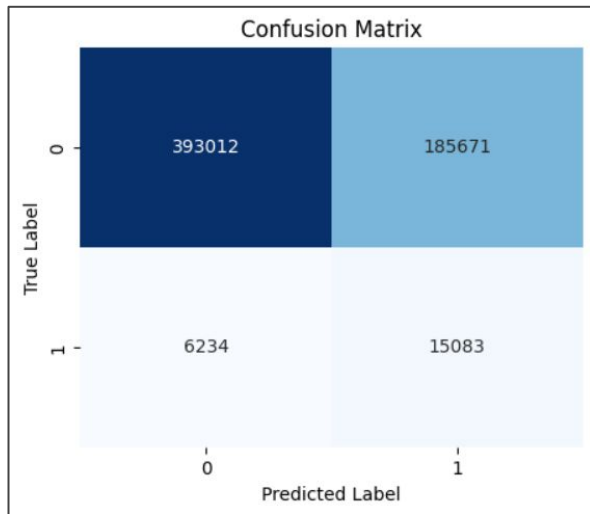
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, model.predict_proba(x_test)[: , 1])

print(classification_report(y_test, y_pred))

print(f"Accuracy: {accuracy:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")
print(f"F1 Score: {f1:.3f}")
print(f"ROC AUC: {roc_auc:.3f}")
```

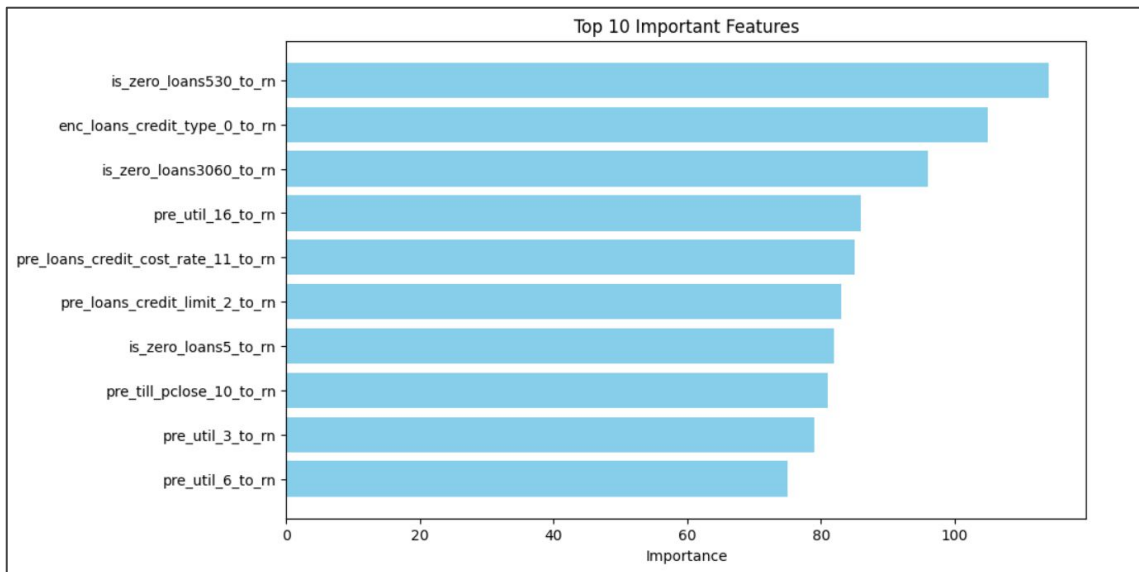
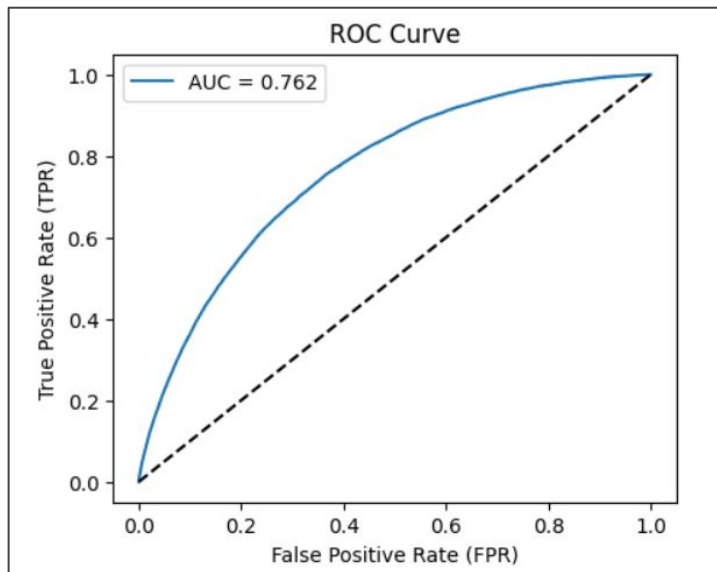
	precision	recall	f1-score	support
0	0.98	0.68	0.80	578683
1	0.08	0.71	0.14	21317
accuracy			0.68	600000
macro avg	0.53	0.69	0.47	600000
weighted avg	0.95	0.68	0.78	600000

Accuracy: 0.680  
Precision: 0.075  
Recall: 0.708  
F1 Score: 0.136  
ROC AUC: 0.762





# Моделирование на полном объеме данных





Спасибо за внимание!