tnarembekov0212@gmail.com
tg: FountainEww
+7-(700)-271-77-18 +7-(916)-964-25-83
Astana. Temirlan Narembekov
tnarembekov0212@gmail.com
tg: FountainEww
+7-(700)-271-77-18 +7-(916)-964-25-83
Astana.

## 1 Задание

Рассмотрим задачу кредитного скоринга: Модель предсказывает вероятность того, что клиент с долгом начнет погашение в течение N=7 дней после напоминания. На основе этой вероятности предложить приоритизацию коммуникаций с точки зрения эффективного способствования к возобновлению платежей клиентами.

Анализ данных EDA 1) Индикатором успешной коммуникации считаем снижение просрочки

OVERDUE в течение 7 дней после коммуникации.

## 2 Выбор и построение Таргета

1.Построим столбец target из 0 и 1, где 1 - если у клиента

OVERDUE уменьшилась в течение N=7 дней после связи, 0 - иначе.

```sql
CREATE TABLE TargetTable AS
WITH RankedData AS (
    SELECT
        ID_NUMBER,
        DATE_FEATURE,
        OVERDUE,
        ROW_NUMBER() OVER (PARTITION BY ID_NUMBER ORDER BY DATE_FEATURE) AS row_num
    FROM DATASET
),
TargetCalculation AS (
    SELECT
        a.ID_NUMBER,
        a.DATE_FEATURE,
        a.OVERDUE,
        CASE
            WHEN EXISTS (
                SELECT 1
                FROM RankedData b
                WHERE
                    b.ID_NUMBER = a.ID_NUMBER
                    AND b.DATE_FEATURE > a.DATE_FEATURE
                    AND b.DATE_FEATURE <= DATE(a.DATE_FEATURE, '+7 days')
                    AND b.OVERDUE < a.OVERDUE
            ) THEN 1
            ELSE 0
        END AS target
    FROM RankedData a
)
SELECT *
FROM TargetCalculation;
```

Для анализа влияния выберем период в 30 дней.

## 3 Подготовка данных

1.Обработаем пропуски:
1)удалим нерелевантные признаки и те, где слишком много пропусков.
2)заполним пропуски в ключевых признаках: в категориальных модой, а в численных среднеарифметическим либо медианой.

## 4 Построение модели

1. Разобьем данные на обучающую и тестовую выборки:

```
unique_clients_in_order = (
    base.drop_duplicates('ID_NUMBER')[['ID_NUMBER', 'DATE_FEATURE']]
    .sort_values('DATE_FEATURE')  # можно опустить, если уже отсортировано
    .reset_index(drop=True)
)
test_size = 0.2
n_clients = len(unique_clients_in_order)
split_idx = int(n_clients * (1 - test_size))
# IDs клиентов для train и test
train_client_ids = unique_clients_in_order.iloc[:split_idx]['ID_NUMBER']
test_client_ids = unique_clients_in_order.iloc[split_idx:]['ID_NUMBER']
train = base[base['ID_NUMBER'].isin(train_client_ids)]
# Test — все записи клиентов из test_client_ids
test = base[base['ID_NUMBER'].isin(test_client_ids)]
target_column = "target"
groups = train['ID_NUMBER'].values

data_train = train.drop(columns=[target_column, 'ID_NUMBER','DATE_FEATURE'])
Y_train = train[target_column]
data_test = test.drop(columns=[target_column, 'ID_NUMBER','DATE_FEATURE'])
Y_test = test[target_column]
```

2. Займемся базовой предобработкой данных - OneHotEncoder() и MinMaxScaler() настраивается по обучающей выборке и применяется к тестовой.

```
categorical = [col for col in data_train.columns if col not in numerical and col != target_column


ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
data_train_numerical = scaler.fit_transform(data_train[numerical])
data_test_numerical = scaler.transform(data_test[numerical])
data_train_categorical = ohe.fit_transform(data_train[categorical])
data_test_categorical = ohe.transform(data_test[categorical])

X_train = np.hstack([data_train_numerical, data_train_categorical])
X_test = np.hstack([data_test_numerical, data_test_categorical])
```

## 3. ВЫБОР И ПОСТРОЕНИЕ МОДЕЛИ

0)Метрика

Прежде всего определим метрику так как строя модель, мы захотим получить ее оптимальную версию, а для этого нужно знать что является мерой качества.

В задачах предсказания вероятностей метрикой выбирают ROC-AUC, так как тут важнее не сами метки, а правильный порядок на объектах. Так же исходя из постановки задачи можно принять "1"(клиент начнет выплату) как положительный класс. Ведь действительно, банк может захотеть рассчитывать на эти выплаты(например ожидать что он выплатит хотя бы среднее от всех прошлых выплат клиента) уже сейчас и задействовать полученные средства в других операциях. Однако это может повлечь и определенные риски:Например, деньги не поступили, а банк уже на них рассчитывает и взял на них некоторые обязательства и в итоге просто не может их исполнить и, возможно, теряет всю прибыль и вклад в операцию. В то время как "0"не дает никаких ожиданий по выплатам, но даже если модель ошибается и клиент все таки выплачивает, то тут риск разве что упустить возможную выгоду. Именно поэтому классы не равнозначны и, помимо ROC-AUC, стоит обращать внимание на PRECISION(как на вторую метрику) так как задача классификации связана с задачей предсказания вероятностей.

1)Базовая модель LogisticRegressionCV()

```
gs = GroupKFold(n_splits=3)
LogR = LogisticRegressionCV(Cs=np.logspace(-7, 7, 25),fit_intercept=True,cv=gs,
                            dual=False,penalty='l2',scoring='roc_auc',solver='lbfgs',tol=0.0001, max_iter=500,
                            class_weight='balanced', n_jobs=None, verbose=0, refit=True,
                            intercept_scaling=1.0,multi_class='deprecated',random_state=seed,##seed
                            l1_ratios=None)

LogR.fit(X=X_train, y=Y_train,groups=groups)
prediction_LogR = LogR.predict(X_test)
y_pred_proba = LogR.predict_proba(X_test)[:, 1]
# Получение выбранного коэффициента регуляризации
best_C = LogR.C_[0]
```

Определимся с гиперпараметрами:

a)L2 регуляризация

б)в Cs укажем сетку для поиска оптимального значения коэффициента регуляризации
в)scroring = 'roc-auc' так как эту метрику мы и хотим максимизировать
г)solver='lbfgs' - выбираем как солвер совместимый с L2(на других совместимых у меня не сходилась оптимизация)
д)max-iter=500 - большое число итераций для лучшей сходимости оптимизатора
е)class-weigth='balanced' - для сбалансировки классов
Остальные по умолчанию
Обучим модель в файле LOGR_CLEANED.py:

```
roc_auc_scorer = make_scorer(roc_auc_score, greater_is_better=True)
precision_scorer = make_scorer(precision_score, greater_is_better=True)

Log_cross = cross_val_score(estimator=LogR,X=X_train,y=Y_train,scoring=roc_auc_scorer,
                            cv=gs,params={"groups": groups})

print("Crossvalidation ROC_AUC scores:", Log_cross)
Log_Cr = cross_val_score(estimator=LogR,X=X_train,y=Y_train,scoring=precision_scorer,
                         cv=gs,params={"groups": groups})
print("Crossvalidation PRECISION scores:", Log_Cr)
```

```
Mounted at /content/drive
/usr/local/lib/python3.11/dist-packages/sklearn/preprocessing/_encoders.py:246: UserWarni
  warnings.warn(
Оптимальное значение C: 1.0
Precision: 0.26101141924959215
Recall: 0.4512489927477841
Roc_Auc: 0.807727791312572
F1: 0.33072493725084895
F_beta:0.2850453018426143
Confusion Matrix:
TP:1120 FN:1362 FP:3171 TN:20160
Crossvalidation ROC_AUC scores: [0.70646045 0.71218133 0.69345481]
Crossvalidation PRECISION scores: [0.13218455 0.13638177 0.13820757]
```

ROC_AUC = 0.8, PRECISION = 0.26.

2) RandomForest

Правильно было бы сузить область поиска оптимальной комбинации гиперпараметров при помощи RandomizedSearchCV() чтобы потом на графиках выбрать несколько значений гиперпараметров с хорошими средними показателями метрики и передать их в GridSearch(), но мощностей моего пк не хватило для обучения. Реализация метода в файле RF_SEARCH.py.
Так что подберем гиперпараметры на небольшой сетке в CV_FOR_RF.py:

```
gs = GroupKFold(n_splits=3)
grid_param = {
              'max_depth': [30,45,12],
              'min_samples_split' : [100,500,35],
              'min_samples_leaf': [20,100,500]
              }
roc_auc_scorer = make_scorer(roc_auc_score, greater_is_better=True)
precision_scorer = make_scorer(precision_score, greater_is_better=True)

RF_PCA = RandomForestClassifier(n_estimators=300,criterion='gini', #30
                                min_weight_fraction_leaf=0.0,max_features='sqrt',
                                max_leaf_nodes=None, min_impurity_decrease=0.0,bootstrap=False, oob_score=False,
                                random_state=seed,verbose=0,warm_start=False,class_weight='balanced',ccp_alpha=0.0,
                                max_samples=None)

grid_search = GridSearchCV(
    estimator=RF_PCA,
    param_grid=grid_param,
    scoring=roc_auc_scorer,
    n_jobs=-1,
    refit='roc_auc',
    cv=gs.split(X_train,Y_train, groups=groups),  # ← вот здесь теперь TimeSeriesSplit
    verbose=0
)
grid_search.fit(X_train,Y_train)
```

```
Mounted at /content/drive
Оптимальное значение max_depth: 30
Оптимальное значение min_samples_split: 500
Оптимальное значение min_samples_leaf: 100
Best CV ROC AUC: 0.7692051409599596
```

а)max_features = ’sqrt’ деревьям рекомедуют передавать корень от числа признаков нежели log2
б) criterion=’gini’ в банковской сфере в похожих задачах используют такую оценочную функцию разбиения.

Далее обучим модель:

```
RF_cross = cross_val_score(estimator=RF_PCA,X=X_train,y=Y_train,scoring=roc_auc_scorer,
                           cv=gs.split(X_train, Y_train, groups=groups))

print("Crossvalidation ROC_AUC scores:", RF_cross)
RF_Cr = cross_val_score(estimator=RF_PCA,X=X_train,y=Y_train,scoring=precision_scorer,
                        cv=gs.split(X_train, Y_train, groups=groups))
print("Crossvalidation PRECISION scores:", RF_Cr)
```

```
Mounted at /content/drive
Precision: 0.2947689345314506
Recall: 0.7401289282836422
Roc_Auc: 0.8647068804958081
F1: 0.4216203809960982
F_beta:0.3350966800437796
Confusion Matrix:
TP:1837 FN:645 FP:4395 TN:18936
Crossvalidation ROC_AUC scores: [0.76411896 0.77588816 0.7676083 ]
Crossvalidation PRECISION scores: [0.15064435 0.16338104 0.15885057]
```

ROC_AUC - 0.86, PRECISION = 0.29

3)XGBoost

Проведем GridSearch() по небольшим сеткам в xgb_gridsearch.py:

```
gs = GroupKFold(n_splits=3)
grid_param = {
              'max_depth': [3,7,12],
              'eta' : [0.2,0.65,1],
              'n_estimators': [300,1000]
              }
params = {
'objective': 'binary:logistic'
        }
xg = XGBClassifier(**params,random_state=seed)
grid_search = GridSearchCV(
    estimator=xg,
    param_grid=grid_param,
    scoring=roc_auc_scorer,
    n_jobs=-1,
    refit='roc_auc',
    cv=gs.split(X_train, Y_train, groups=groups),  # ← вот здесь теперь TimeSeriesSplit
    verbose=0
)
grid_search.fit(X_train,Y_train)
print("Оптимальное значение max_depth:", grid_search.best_params_['max_depth'])
print("Оптимальное значение eta:", grid_search.best_params_['eta'])
print("Оптимальное значение n_estimators:", grid_search.best_params_['n_estimators'])
print("Best CV ROC AUC:", grid_search.best_score_)
```

```
Mounted at /content/drive
Оптимальное значение max_depth: 3
Оптимальное значение eta: 1
Оптимальное значение n_estimators: 1000
Best CV ROC AUC: 0.5603483703788936
```

✓ 11m 10s completed at 3:39 PM

Обучим модель в crossvalidation_XGB.py:

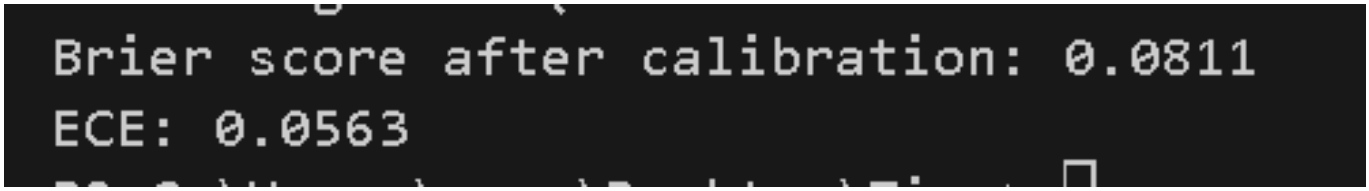
```
117   full_np = np.concatenate([X_train, X_test], axis=0) # full_np = np.vstack([X_train,
118   Y_full = np.concatenate([Y_train, Y_test], axis=0).ravel()#Y_
119   roc_auc_scorer = make_scorer(roc_auc_score, greater_is_better=True)
120   precision_scorer = make_scorer(precision_score, greater_is_better=True)
121   gs = GroupKFold(n_splits=3)
122   xgb_cross = cross_val_score(estimator=xg,X=full_np,y=Y_full,scoring=roc_auc_scorer,
123                               cv=gs.split(X_train, Y_train, groups=groups))
124   print("Crossvalidation ROC_AUC scores:", xgb_cross)
125   xgb_cross_2 = cross_val_score(estimator=xg,X=full_np,  y=Y_full,
126                                 scoring=precision_scorer,
127                                 cv=gs.split(X_train, Y_train, groups=groups))
128   print("Cross-validation PRECISION scores:", xgb_cross_2)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\user> cd C:\Users\user\Desktop\Tima
PS C:\Users\user\Desktop\Tima> python crossvalidation_XGB.py
Precision: 0.31989247311827956
Recall: 0.14383561643835616
Roc_Auc: 0.8308873911450084
F1: 0.19844357976653695
F_beta:0.2569824359343507
Confusion Matrix:
TP:357 FN:2125 FP:759 TN:22572
Crossvalidation ROC_AUC scores: [0.55390405 0.5620352  0.55521602]
Cross-validation PRECISION scores: [0.20596373 0.22260372 0.22075123]
```

ROC_AUC - 0.83, PRECISION - 0.31.

4)Остановимся на модели RandomForestClassifier() и Проведем калибровку вероятностей в rf.py:

```
196  from sklearn.metrics import brier_score_loss
197
198  brier = brier_score_loss(Y_test, y_proba)
199  print(f"Brier score: {brier:.4f}")
200
201  from sklearn.calibration import CalibratedClassifierCV
202
203  # Обучаем калибратор на уже обученной модели
204  calibrator = CalibratedClassifierCV(RF_PCA, method='isotonic', cv='prefit')
205  calibrator.fit(X_train, Y_train)
206  # Теперь получаем хорошо откалиброванные вероятности
207  calibrated_proba = calibrator.predict_proba(X_test)[:, 1]
```



## 5 5.АНАЛИЗ РЕЗУЛЬТАТОВ

I.Найдем самую эффективную коммуникацию с точки зрения предсказанных моделью вероятностей. Возьмем среднеарифметическое всех вероятностей отдельно для каждой из COMMUNICATION_TYPE_P, COMMUNICATION_TYPE_C,
COMMUNICATION_TYPE_S и сравним их в rf.py: Наиболее эффективный тип COMMUNICATION_TYPE_C со средней вероятностью 0.4583. Затем COMMUNICATION_TYPE_S(0.22) и
COMMUNICATION_TYPE_P(0.18).

II. Теперь рассмотрим долю тех коммуникаций, чью вероятности превышают порог в 0.7:

Здесь получится точно та же приоритизация и можно определить COMMUNICATION_TYPE_C как самую эффективную коммуникацию.

III.На самом деле, стоило бы попарно проверить статистическую гипотезу Стьюдента о равенстве средних в двух зависимых выборках(Действительно, в тестовое множество могут попадать клиенты которым присылают несколько типов коммуникаций, и получается, что в разных выборках,например для COMMUNICATION_TYPE_P и COMMUNICATION_TYPE_C, содержатся коммуникации отправленные одному и тому же человеку) чтобы проверить что
COMMUNICATION_TYPE_C действительно имеет большую среднюю вероятность. К тому же Центрально-предельная теорема позволяет это сделать для больших выборок.
Но для начала - Проведем статистические гипотезы Стьюдента ttest_rel с нулевой гипотезой о равенстве средних не на предсказанных моделью вероятностях, а на фактических метках в исходном датасете.И таким образом выясним у кого доля "1"больше.

Но для начала проведем Тест Пирсона chi2_contingency с нулевой гипотезой о независимости признака COMMUNICATION_TYPE и таргета, ведь если они независимы, то ни о какой эффективности не идет речи т.к. коммуникации не влияют.statistic_hypothesis_CHE2.py

```
period = pd.read_csv(file_path,low_memory=False)
# Создание таблицы сопряженности для notificationname
contingency_table = pd.crosstab(period['COMMUNICATION_TYPE'], period['target'])

# Выполнение теста
chi2, p, dof, expected = chi2_contingency(contingency_table)

# print(f"Chi-square statistic for notifcationtype: {chi2}")
# print(f"P-value: {p:.30f}")
```

p_value меньше любого стандартного уровня значимости $\alpha$ , поэтому гипотеза о независимости отклоняется.

Теперь проведем тест Стьюдента о равенстве средних в зависимых выборках сгруппировав их по ID_NUMBER в statistic_hypothesis_CHE2.py
Статистика критерия имеет вид:

$$t = \frac{\bar{d}}{s_d/\sqrt{n}},$$

где

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i,$$

$$s_d = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(d_i - \bar{d})^2}.$$

Значит если t>0(t<0), то первая(вторая) переданная в ttest_rel выборка имеет большее среднее, т.к. очередность определяет знак t.

```
p_data = period[period['COMMUNICATION_TYPE'] == 'P']
c_data = period[period['COMMUNICATION_TYPE'] == 'C']
s_data = period[period['COMMUNICATION_TYPE'] == 'S']
# Объединяем данные по contract_number
paired_data_PC = pd.merge(
    p_data[['ID_NUMBER', 'target']].rename(columns={'target': 'p_target'}),
    c_data[['ID_NUMBER', 'target']].rename(columns={'target': 'c_target'}),
    on='ID_NUMBER',
    how='inner'  # Только клиенты, получившие и PUSH, и CALL на протяжении всего периода
)
paired_data_PS = pd.merge(
    p_data[['ID_NUMBER', 'target']].rename(columns={'target': 'p_target'}),
    s_data[['ID_NUMBER', 'target']].rename(columns={'target': 's_target'}),
    on='ID_NUMBER',
    how='inner'  # Только клиенты, получившие и PUSH, и SMS на протяжении всего периода
)
paired_data_CS = pd.merge(
    c_data[['ID_NUMBER', 'target']].rename(columns={'target': 'c_target'}),
    s_data[['ID_NUMBER', 'target']].rename(columns={'target': 's_target'}),
    on='ID_NUMBER',
    how='inner'  # Только клиенты, получившие и CALL, и SMS на протяжении всего периода
)
t_stat, p_value = ttest_rel(paired_data_PC['p_target'], paired_data_PC['c_target'])
t_stat, p_value = ttest_rel(paired_data_PS['p_target'], paired_data_PS['s_target'])
t_stat, p_value = ttest_rel(paired_data_CS['c_target'], paired_data_CS['s_target'])
```

```
 T-statistic = -46.96664788151819, P-value = 0.0
T-statistic = 6.902452896912295, P-value = 5.135903103733l7e-12
T-statistic = 51.99934768504777, P-value = 0.0
```

Таким образом получаем ту же приоритизацию.

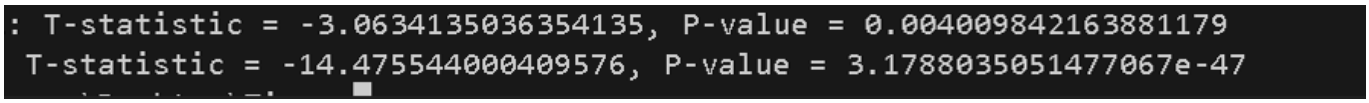
III. Проделаем то же самое для вероятностей добавив их и восстановив DATE_FEATURE в тестовом множестве в файле rf.py.
Сохраним новый дополненный тестовый датасет в X_data__with_probabilitiesRF.csv.
Тест Пирсона на независимость probability и COMMUNICATION_TYPE

```
period = pd.read_csv(file_path,low_memory=False)
period['prob_bin'] = pd.cut(period['probability'], bins=5)
# Создание таблицы сопряженности для notificationname
contingency_table = pd.crosstab(period['COMMUNICATION_TYPE'], period['prob_bin'])

# Выполнение теста
chi2, p, dof, expected = chi2_contingency(contingency_table)

# print(f"Chi-square statistic for notifcationtype: {chi2}")
# print(f"P-value: {p:.30f}")
```

```
Chi-square statistic for notifcationtype: 263.8371927478728
P-value: 0.000000000000000000000000000000
```

Таким образом, отклоняем нулевую гипотезу о независимости на уровне значимости $\alpha = 0.01(0.05)$

Далее Тест Стьюдента для COMMUNICATION_TYPE_P VS COMMUNICATION_TYPE_C И COMMUNICATION_TYPE_P VS COMMUNICATION_TYPE_S соответсвенно.




Отклоняем обе гипотезы о равенстве средних в выборках вероятностей соответствующих COMMUNICATION_TYPE_P И COMMUNICATION_TY так же COMMUNICATION_TYPE_P И COMMUNICATION_TYPE_S

При этом средняя вероятность для TYPE_P меньше средних вероятностей для TYPE_C и TYPE_S.
t

Заметим, что в тестовом множестве нет клиентов получивших и COMMUNICATION_TYPE_S и COMMUNICATION_TYPE_C, значит, соответсвующие выборки независимы, а нам остается только проверить гипотезу Ливена о равенстве дисперсий и затем ttest_ind для независимых выборок.



Таким образом, COMMUNICATION_TYPE_C имеет самую большую среднюю вероятность, затем COMMUNICATION_TYPE_S, и только потом COMMUNICATION_TYPE_P.

# 6 РЕКОМЕНДАЦИИ

1)Отказаться от коммуникаций в Субботу
2)Увеличить коммуникации эффективного TYPE_OF_COMM COMMUNICATION_TYPE_C и равномерно распределить их по дням недели.