Предсказание оттока клиентов для «ТелеДом»

Введение

В условиях высокой конкуренции на рынке телекоммуникационных услуг операторы связи такие как «ТелеДом» сталкиваются с проблемой оттока клиентов, что негативно сказывается на доходах и репутации. В целях удержания клиентов и минимизации потерь, компания стремится внедрить меры, направленные на предотвращение отказов от услуг.

Для достижения этой цели планируется разработать модель машинного обучения, способную предсказывать вероятность разрыва договора клиентами на основе анализа их персональных данных, тарифов и используемых услуг. Модель будет обучена на исторических данных о клиентах, что позволит выявить ключевые факторы, способствующие оттоку.

Эта инициатива позволит идентифицировать потенциальных клиентов, готовых отказаться от услуг, и предоставит возможность предложить им индивидуальные промокоды и специальные условия, что в свою очередь может повысить уровень удовлетворенности и лояльности клиентов. В конечном итоге, успешная реализация данного проекта будет способствовать улучшению финансовых показателей компании и укреплению её позиций на рынке.

Загрузка исходных данных

Используемые библиотеки

```
!pip install numpy==1.21.6 -q
!pip install pandas==1.3.5 -q
!pip install scipy==1.9.3 -q
!pip install contourpy==1.0.7 -q
!pip install numba==0.56.0 -q
!pip install matplotlib==3.8.4 -q
!pip install seaborn==0.13.2 -q
!pip install scikit-learn==1.4.0 -q
!pip install shap -q
!pip install phik -q
!pip install catboost -q

import phik
import shap
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.model_selection import  GridSearchCV, train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder,
StandardScaler, MinMaxScaler, TargetEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import confusion_matrix, roc_auc_score,
accuracy_score
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from catboost import CatBoostClassifier
```

Загрузка данных

```
contract = pd.read_csv('/datasets/contract_new.csv')
personal = pd.read_csv('/datasets/personal_new.csv')
internet = pd.read_csv('/datasets/internet_new.csv')
phone = pd.read_csv('/datasets/phone_new.csv')
```

Первичное ознакомление с данными

Для удобства создадим словарь датафреймов.

```
dfs = {'contract':contract, 'personal':personal,
      'internet':internet, 'phone':phone}

for df in dfs:
    print(df)
    display(dfs[df].head(),dfs[df].info())
```

Есть информация о 7000 пользователях, среди которых некоторые используют только телефонию, другие — только интернет, а некоторые — и то, и другое. Однако не все столбцы данных имеют корректные типы.

Предобработка данных

Объединение таблиц

Прежде чем выполнять предобработку данных обьединим таблицы.

```
for df in dfs:
    print(df,dfs[df].duplicated('customerID').sum())
```

Все пользователи имеют уникальный ID, дубликатов нет.

```
contract ['customerID'].isin(personal['customerID']).sum()
```

Все пользователи есть в обеих таблицах

```
df = contract.merge(personal , on='customerID')

df = df.set_index('customerID').join(internet.set_index('customerID'))
```

Не все пользуются услугами интернета, некоторые используют только телефонию, из-за этого при объединении образуются прпуски.

```
df = df.join(phone.set_index('customerID'))
```

Объединенная таблица будет выглядеть следующим образом:

```
df.head()
```

Пропуски

Исследуем получившуюся таблицу на наличие пропусков и всех видов дубликатов. Но перед этим изменим типы данных там, где это необходимо.

```
df['TotalCharges'].replace(to_replace= {' ':np.nan},inplace = True)
df['TotalCharges'] = df['TotalCharges'].astype('float')
df['EndDate'].replace(to_replace= {'No':np.datetime64("NaT")},inplace
= True)
df['EndDate'] = pd.to_datetime(df['EndDate'])
df['BeginDate'] = pd.to_datetime(df['BeginDate'])
df.info()
```

Типы данных приведены к нужному формату, посмотрим на пропуски.

```
df.isna().sum()
```

Пропуски в столбце EndDate это пользователи контракт которых еще действует, этот столбец делит пользователей на ушедших и оставшихся, на его основании будет сформирован целевой признак. В столбце TotalCharges должны быть указаны суммарные траты пользователя за все время. Пропуски в остальных столбцах означают что услуга не подключена, а если точнее то интернета вообще нет. Учитывая это заполним пропуски.

```
df[df['TotalCharges'].isna()]
```

Понятно в чем дело, это свежие договоры оформленые в день выгрузки данных 1 февраля 2020 года. Общая стоимость будет равна месячной плате (предположим что подключение бесплатное).

```
df.loc[df['TotalCharges'].isna(),'TotalCharges'] =
df.loc[df['TotalCharges'].isna(),'MonthlyCharges']
```

Заполним данные об интернет услугах новым значением обозочаеющим отсутвие подключенного интернета.

```
df.loc[df['InternetService'].isna(),'InternetService':'StreamingMovies
'] = 'not connected'
```

Отуствие телефонии обозначим также.

```
df.loc[df['MultipleLines'].isna(),'MultipleLines'] = 'not connected'
```

Столбец EndDate мы не сможем использовать в исходном виде при обучении модели из-за утечки данных, поэтому расформируем столбец и на его основе сделаем два новых. Первый будет показывать время прошедшее с заключения договора с клиентом, а второй целевой признак.

```
df['Departed'] = (~df['EndDate'].isna()).astype(int) # Столбец с
целевым признаком

df.loc[df['EndDate'].isna(), 'EndDate'] = '2020-02-01'# Дата выгрузки
датафрейма

df['Tenor'] = (df['EndDate'] - df['BeginDate']).dt.days #Время,
прошедшее с момента подписания договора

df.drop(['EndDate','BeginDate'], axis=1, inplace=True) #Удалим
неинофрмативные столбцы

df.info()
```

Переходим к дубликатам.

Дубликаты

```
for column in df:
    if df[column].dtype==object:
        display(column, df[column].unique())
```

Явных дубликаов нет, так как все ID уникальные, неявных дубликатов тоже нет.

Данные предварительно обработаны и готовы к дальнешему анализу

Исследовательский анализ данных

Исследовательский анализ данных играет ключевую роль в понимании поведения пользователей. Анализ групп пользователей позволяет сравнить поведение двух групп и выявить паттерны в их поведении. Полученные инсайты не только помогают бизнесу принимать обоснованные решения, но и служат основой для обучения моделей машинного обучения, способных предсказывать вероятность оттока.

Статистический анализ признаков

```
df.drop(['SeniorCitizen','Departed'],axis=1).describe()
```

Средние ежемесячные расходы составляют около 65, с довольно высоким стандартным отклонением (30.09), что указывает на значительное разнообразие в расходах пользователей. Показатели TotalCharges также указывают на различия в тратах пользователей, однако во многом высокие значения связаны с большей длительностью контракта. Tenor, то есть длительность контракта, показывает что в среднем пользователи пользуются услугами чуть больше двух лет, это может быть связано с наличием подписки на 2 года.

```
for column in df:
    if df[column].dtype==object or column in
['Departed','SeniorCitizen']:
        display(pd.DataFrame(df[column].value_counts()))
```

Четверть пользователей имеют подписку на два года и чуть меньше на 1 год, в целом у пользователй наблюдается заинтересованость в дополнительных сервисах и электронным методам оплаты. Посмотрим подбробно на распределение пользователей различных групп.

```
for column in ['Tenor','MonthlyCharges','TotalCharges']:
    plt.figure(figsize=(10, 6))
    plt.subplot(1, 2, 1)
    sns.histplot(data=df[df['Departed']==0], x=column, stat='density',
common_norm=False, label='Остались')
    plt.title(column)
    plt.subplot(1, 2, 1)
    sns.histplot(data=df[df['Departed']==1], x=column, stat='density',
common_norm=False, label='Ушли')
    plt.legend()
    plt.show();
```

Ушедшие клиенты чаще всего расторгают контракт через примерно 3 года от его заключения, также с увеличение роста месячной платы растет и число ушедших клиентов.

```
for column in ['Tenor','MonthlyCharges','TotalCharges']:
    plt.figure(figsize=(10, 6))
    plt.subplot(1, 2, 1)
    plt.title(column, loc='center', pad=20)
    df[df['Departed']==0][column].plot.box(color='blue',
label='Остались')
    plt.subplot(1, 2, 2)
    plt.title(column, loc='center', pad=20)
    df[df['Departed']==1][column].plot.box(color='red', label='Ушли')
    plt.show();
```

```
for column in df:
    if df[column].dtype==object or column=='SeniorCitizen':
        plt.subplot(1, 2, 1)
        df[df['Departed']==0]
[column].value_counts().plot.pie(figsize=(12,12),autopct='%1.f%
%',shadow=True)
        plt.xlabel('Остались')
        plt.subplot(1, 2, 2)
        df[df['Departed']==1]
[column].value_counts().plot.pie(figsize=(12,12),autopct='%1.f%%',
shadow=True)
        plt.xlabel('Ушли')
        plt.show();
```

Явно прослеживаются различия между группами клиентов по типу подписки. Хотя распределение остальных признаков в этих группах не кажется особенно значимым, они все же могут оказывать определенное влияние на поведение пользователей.

Корреляционный анализ

Перед обучением модели необходимо изучить их взаимосвязи и проверить их корреляцию, чтобы обеспечить более качественный анализ данных и улучшить производительность модели.

```
corr_matrix = df.phik_matrix(interval_cols=['MonthlyCharges',
'TotalCharges', 'Tenor'])

plt.figure(figsize=(12, 12))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Матрица корреляций')
plt.show()
```

При использовании различных моделей важно учитывать влияние мультиколлинеарности на их производительность и интерпретируемость. Логистическая регрессия подвержена негативным эффектам, тогда как деревья решений, KNN и CatBoost могут более эффективно справляться с этой проблемой благодаря своей структуре и алгоритмам. Поэтому при обучении линейной модели будем использовать, поскольку она позволяет минимизировать негативные последствия мультиколлинеарности и улучшить общую производительность модели.

Подготовка данных и создание моделей

Данные, предоставленные для обучения модели, имеют дисбаланс классов. Существует множество методов борьбы с этой проблемой с разной эффективностью, но мы сохраним исходный баланс классов, потому что это позволит нам сохранить полное представление о данных и избежать потери информации, связанной с изменением распределения классов. Единственное, что мы сделаем, это обеспечим одинаковое распределение классов при разделении на тестовую и тренировочную выборки, используя стратифицированное

разбиение. Это гарантирует, что пропорции классов будут сохранены в обеих выборках, что позволит модели лучше обучаться и оцениваться.

```
RANDOM_STATE = 141024
TEST_SIZE = 0.25

X_train, X_test, y_train, y_test = train_test_split(
    df.drop('Departed',axis=1),
    df['Departed'],
    stratify = df['Departed'],
    random_state=RANDOM_STATE,
    test_size = TEST_SIZE)

#Создаем списки признаков
ohe_col = list(df.select_dtypes(include=['object']).columns)
num_col = ['MonthlyCharges','TotalCharges','Tenor']

cat_pipe = Pipeline(
    [
         ('simpleImputer_cat', SimpleImputer(missing_values=np.nan,
strategy='most_frequent')),
         ('cat', OneHotEncoder(drop='first', handle_unknown='ignore'))
    ]
)

#Создаём пайплайн для подготовки признаков
data_preprocessor = ColumnTransformer(
    [
        ('cat', cat_pipe, ohe_col),
        ('num', StandardScaler(), num_col)
    ],
    remainder='passthrough'
)


#Финашльный пайплайн
pipe_final = Pipeline(
    [
        ('preprocessor', data_preprocessor),
        ('models', LogisticRegression(random_state=RANDOM_STATE))
    ]
)

#Параметры пайплайна
param_grid = [

     # словарь для модели DecisionTreeClassifier()
    {
        'models': [DecisionTreeClassifier(random_state=RANDOM_STATE)],
        'models__max_depth': range(2,16),
```

```
        'models__min_samples_leaf': range(1,16),
        'preprocessor__cat__cat': [OrdinalEncoder()]
    },


    # словарь для модели LogisticRegression()
    {
        'models': [LogisticRegression(random_state=RANDOM_STATE)],
        'models__C': range(1,5),
        'models__penalty':('l2','l1'),
        'models__solver': ['liblinear'],
    },

    # словарь для модели KNeighborsClassifier()
    {
        'models': [KNeighborsClassifier()],
        'models__n_neighbors': range(2,10),
        'preprocessor__num': [StandardScaler(), MinMaxScaler(),
'passthrough'],
        'preprocessor__cat__cat':
[OrdinalEncoder(),OneHotEncoder(drop='first',
handle_unknown='ignore')]
    },

    {
        'models': [CatBoostClassifier(random_state=RANDOM_STATE,
verbose=0, class_weights={0:1,1:7})],
        'models__learning_rate': [0.01, 0.03],
        'models__depth': range(2, 8, 1),
        'preprocessor__cat__cat': [OrdinalEncoder()]
    }
]



GridSearch = GridSearchCV(
    pipe_final,
    param_grid,
    cv=3,
    scoring=['roc_auc','accuracy'],
    refit='roc_auc',
    n_jobs=-1)

GridSearch.fit(X_train, y_train)

print('Лучшая модель и её параметры:\n\n', GridSearch.best_estimator_)
print ('Метрика лучшей модели на кросс-валидации',
GridSearch.best_score_)
```

```
# Получаем результаты кросс-валидации
results = GridSearch.cv_results_

# Создаем DataFrame из результатов
results_df = pd.DataFrame(results)

# Извлекаем названия моделей из столбца 'params'
results_df['model'] = results_df['params'].apply(lambda x:
str(x['models']).split('(')[0].strip())

# Группируем по модели и находим индекс строки с максимальным
значением AUC-ROC
best_auc_indices = results_df.groupby('model')
['mean_test_roc_auc'].idxmax()

# Извлекаем лучшие результаты
best_results = results_df.loc[best_auc_indices, ['model',
'mean_test_roc_auc','mean_test_accuracy']]

# Переименовываем столбцы для удобства
best_results.columns = ['Model', 'Best AUC-ROC', 'Best accuracy']

# Выводим таблицу с результатами
display(best_results)
```

Модели catboost удалось добиться необходимой метрики на кросс-валидации.

Значение ROC-AUC 0.877 указывает на то, что модель имеет отличную способность различать положительные и отрицательные классы. Это означает, что модель хорошо справляется с задачей классификации, и вероятность того, что она правильно идентифицирует положительный класс, высока.

Значение Accuracy 0.83 означает, что 83% предсказаний модели были правильными. Это также является хорошим результатом, однако стоит учитывать, что точность может быть не самым лучшим показателем, особенно в случаях несбалансированных классов. Например, если есть много отрицательных примеров и немного положительных, модель может показывать высокую точность, просто предсказывая отрицательный класс.

Анализ модели

Проанализируем модель, возможно мы сможем улучшить метрики модели.

```
X_train_new =
GridSearch.best_estimator_.named_steps['preprocessor'].transform(X_tra
in)
explainer =
shap.TreeExplainer(GridSearch.best_estimator_.named_steps['models'],
X_train_new)
feature_names
=GridSearch.best_estimator_.named_steps['preprocessor'].get_feature_na
```

```
mes_out()
X_train_new = pd.DataFrame(X_train_new, columns=feature_names)
shap_values = explainer(X_train_new)

shap.summary_plot(shap_values, X_train_new, show=False)

# Get the current figure and axes objects. from @GarrettCGraham code
fig, ax = plt.gcf(), plt.gca()

# Modifying main plot parameters
ax.tick_params(labelsize=14)
ax.set_xlabel("Влияние на модель", fontsize=14)
ax.set_title('Вклад признаков в предсказание модели', fontsize=16)

# Get colorbar
cb_ax = fig.axes[1]

# Modifying color bar parameters
cb_ax.tick_params(labelsize=15)
cb_ax.set_ylabel("Значение признака", fontsize=20)

plt.show()
```

Большинство признаков не важны для модели, удалим неинформативные признаки. Это поможет улучшить производительность модели и упростить ее интерпретацию.

Настройка модели

Оставим только наиболее информативные признаки для модели, при этом выберем только те которые не введут модель в заблуждение.

```
X_train, X_test, y_train, y_test = train_test_split(
    df.loc[:,['Tenor','Type','MonthlyCharges','Partner']],
    df['Departed'],
    stratify = df['Departed'],
    random_state=RANDOM_STATE,
    test_size = TEST_SIZE)

#Создаем списки признаков
ohe_col = ['Type','Partner']
num_col = ['Tenor','MonthlyCharges']

#Создаём пайплайн для подготовки признаков
data_preprocessor = ColumnTransformer(
    [
        ('cat', cat_pipe, ohe_col),
        ('num', StandardScaler(), num_col)
    ],
    remainder='passthrough'
)
```

```
#Финашльный пайплайн
pipe_final = Pipeline(
    [
        ('preprocessor', data_preprocessor),
        ('models', CatBoostClassifier(random_state=RANDOM_STATE,
verbose=0, class_weights={0:1,1:7}))
    ]
)

#Параметры пайплайна
param_grid = [
        {
        'models': [CatBoostClassifier(random_state=RANDOM_STATE,
verbose=0, class_weights={0:1,1:7})],
        'models__learning_rate': [0.01, 0.03],
        'models__depth': range(2, 8, 1),
        'preprocessor__cat__cat': [OrdinalEncoder()]

    }
]


GridSearch = GridSearchCV(
    pipe_final,
    param_grid,
    cv=3,
    scoring=['roc_auc','accuracy'],
    refit='roc_auc',
    n_jobs=-1)

GridSearch.fit(X_train, y_train)

print('Лучшая модель и её параметры:\n\n', GridSearch.best_estimator_)
print ('Метрика лучшей модели на кросс-валидации:',
GridSearch.best_score_)
```

После отбора признаков модели удалось улучшить показатели модели, посомотрим как поведет себя модель на тестовой выборке.

```
y_test_pred = GridSearch.predict_proba(X_test)
print(f'ROC-AUC лучшей модели на тестовой выборке:
{roc_auc_score(y_test, y_test_pred[:, 1])}')
print(f'Accuracy лучшей модели на тестовой выборке:
{accuracy_score(y_test,(y_test_pred[:, 1] > 0.5).astype(int))}')
```

Модель показывает стабильно хороший результат как на кросс валидации так и на тестовой выборке. Это указывает на то, что модель хорошо обобщает данные и не переобучена.

```
cm = confusion_matrix(y_test, (y_test_pred[:, 1] > 0.5).astype(int))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues_r');
plt.ylabel('True label')
plt.xlabel('Predicted');
```

Модель правильно классифицировала 237 объектов как положительные из 281 действительно положительных и 1263 объекта как отрицательные из 1408 действительно отрицательных. Модель сбалансированно предсказывает разные классы, несмотря на изначальный дисбаланс. Это связано с параметром весов классов в CatBoost, что позволяет модели уделять больше внимания меньшинству классов и улучшать их предсказания.

Вывод

В ходе анализа модели предсказания ухода клиентов мы достигли значительных успехов в понимании поведения клиентов и в разработке стратегии по минимизации рисков ухода. Модель демонстрирует хорошую способность предсказывать как уходящих, так и остающихся клиентов, что позволяет бизнесу более эффективно управлять своими ресурсами и усилиями по удержанию клиентов.

Однако, учитывая важность точности предсказаний, мы выделили необходимость настройки порогов предсказаний для оптимизации модели в зависимости от конкретных бизнес-целей. Это позволит минимизировать ложноположительные и ложоотрицательные ошибки, что, в свою очередь, повысит эффективность маркетинговых и сервисных стратегий.

Рекомендации для бизнеса

- Настройка порогов предсказаний:

Провести анализ затрат и выгод для определения оптимального порога предсказаний. Если бизнес-фокус заключается в удержании клиентов, следует рассмотреть возможность снижения порога, чтобы минимизировать ложоотрицательные ошибки. Если целью является минимизация затрат на удержание, целесообразно установить более высокий порог, чтобы сократить ложноположительные предсказания.

- Регулярный мониторинг и обновление модели:

Периодически пересматривать и обновлять модель, чтобы учитывать изменения в поведении клиентов и внешних факторах. Это обеспечит актуальность предсказаний и поможет адаптироваться к новым условиям рынка.

- Интеграция модели в бизнес-процессы:

Внедрить модель в существующие бизнес-процессы для автоматизации действий по удержанию клиентов. Например, если модель предсказывает, что клиент может уйти, можно автоматически инициировать специальные предложения или дополнительные услуги для этого клиента.

- Рекомендации по типам подписки:

Стоит рассмотреть возможность предложения скидок или дополнительных привилегий для клиентов, выбирающих 2-летнюю подписку, чтобы стимулировать долгосрочные обязательства и предотвратить уход после окончания действия договора.

- Обратная связь от клиентов:

Собирать и анализировать обратную связь от клиентов, чтобы лучше понять причины их ухода и удовлетворенности. Это поможет в дальнейшем улучшении модели и стратегий удержания.

Эти рекомендации помогут бизнесу не только минимизировать риски ухода клиентов, но и улучшить общую клиентскую удовлетворенность и лояльность. Настройка и интеграция модели предсказания ухода в бизнес-процессы создаст дополнительные возможности для роста и повышения конкурентоспособности компании.