Содержание

- Вступление
- Мержим исходные датасеты
- Метод разбиения на train и test
- EDA
- FeatureGeneration
- Исследование корреляции признаков
- Разбиение эстиматора на 8 моделей по vas_id
- Настройка моделей
- Создание пайплайнов из моделей
- Отбор фичей
- Возможные ошибки
- TO DO
- Создание рекомендательной системы

Вступление

- Извиняюсь за грязный код в проекте. Но времени на создание проекта катастрофически не хватало. Поэтому шаблоны с кодом были выдернуты из разных ноутбуков и вставлены в проект. Прикрепил ноутбук с EDA (Step2_EDA), 8 ноутбуков по созданию моделей предсказания по разным vas_id, ноутбук с проверкой общего пайплайна (Step_1_2_END).
- Но мой проект решением поставленной задачи я не назову. Скорее всего это только разведка боем, которая показала, что с такими датасетами я еще не сталкивался и как правильно эту задачу решить я не знаю. В первую очередь нет понимания что делать с выбросом который происходит где-то с 317 дня по всем vas_id? Если б было хотя бы по 1 выбросу на тренировочный и валидационный датасет, можно было б поделить датасет по времени на тренировочный, валидационный и тестовый. А так получается я и выбросить его не могу, т.к. по многим vas_id в нем основной объем продаж и такой же выброс (скорее всего акция) м.б. и на data_train.scv. И разделить по времени его не могу, т.к. мне нужно добавить в модель хотя бы 1 фичу ‘is_action’ с информацией об этом выбросе, а наиболее надежный способ сделать это: обучить модель на предсказание юзеров, покупающих по акции. И если разделить датасет по времени, разделив выброс, модель недообучится. К тому же за выбросом по многим vas_id наблюдений очень мало для валидации. Если разделить датасет по ‘target’, нарушается метод разделения data_train.scv и data_test.scv плюс нарушается правильная работа созданных мною фичей логирования предложений юзеру ‘vas_id_0X’ и их производная по времени ‘vas_id_day’. Но другого выхода я не увидел, разделил по ‘target’. И в результате я не могу выделить тестовый датасет к выделенным тренировочному и валидационному, поскольку при разбиении по ‘target’ он бессмысленен, т.к. наблюдений, оставшихся за выбросом для него не хватит. И теперь в моей системе принятия решений нет обратной связи. А работа с положительной обратной связью неизбежно ведет к ошибке.
- Другой проблемой я вижу аномалию, когда высокая метрика практически у всех 8 моделей на кроссвалидации сопровождается низкой метрикой на валидации. В результате при настройке модели мне нужно настраивать не увеличение метрики кроссвалидации а избавление от «переобучения». И как избавиться от этой аномалии я не понимаю. М.б. Сделать тренировочный и валидационный датасет ы одинаковой величины или попытаться отрегулировать другие входные параметры?

Мержим исходные датасеты

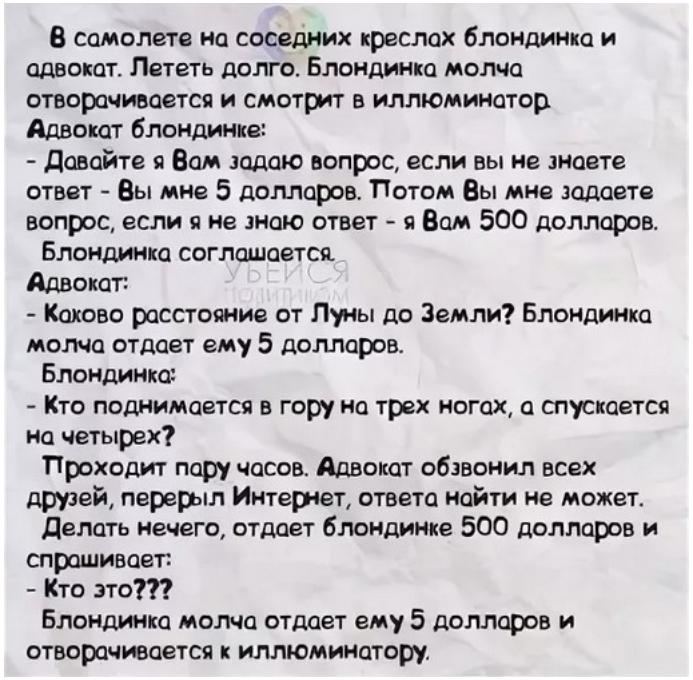
```
def transform(self, data_train, features_train):
    """Трансформация данных"""

    dublicated = features_train[features_train['id'].duplicated(keep=False)].sort_values(by='id')['id']
    data_merged_train = pd.merge(data_train, features_train, on='id')
    tmp = data_merged_train.loc[data_merged_train['id'].isin(dublicated.index)]
    tmp['delta'] = abs(tmp['buy_time_x'] - tmp['buy_time_y'])
    tmp.sort_values(by = ['Unnamed: 0_x', 'delta'], inplace=True)
    duplicates = tmp['Unnamed: 0_x'].duplicated()
    duplicates_to_delete = duplicates[duplicates.values == True]
    data_merged_train.drop(duplicates_to_delete.index, axis=0, inplace=True)
    data_merged_train['time_delta'] = data_merged_train['buy_time_x'] - data_merged_train['buy_time_y']
    data_merged_train.drop(['Unnamed: 0_x', 'Unnamed: 0_y', 'buy_time_y'], axis=1, inplace=True)

    return data_merged_train
```

Я решил сохранить разницу между временем создания фичей юзера и временем предложения юзеру в подключении услуги в отдельную фичу. И поэтому метод merge_asof мне не подошел. Пришлось потанцевать с бубнами, чтобы сохранить разницу между buy_time из обоих датасетов в смерженный датасет. И, как показал метод определения важности фичей SHAP, фича оказалась одной из самых полезных.

Метод разбиения на train и test



Поскольку датасеты data_train и data_test разбиты по времени, то и data_ttrain на обучающую и валидационную выборку логично разбивать по времени. Но... Оказывается в data_train заложена мина: основной объем подключения услуг находится в выбросе между примерно 317 и 350 днем. Таким образом исключать сей факт из ведения модели ну никак нельзя! Пришлось создавать фичу is_action, в которой отмечать единицей подключения по акции. И обучать CatBoostClassificier на нахождение таких подключений. Так вот. Если разбивать по времени, то либо на трейн мы включаем датасет до окончания акции, и на тесте почти не остается подключений вообще. Либо включаем до половины акции и CatBoostClassificier у нас недообучается. Поэтому пришлось делить data_train со стратификацией по фиче ‘target’.

EDA

<class 'category_encoders.basen.BaseNEncoder'> 0.6344820159287033

<class 'category_encoders.binary.BinaryEncoder'> 0.6344820159287033

<class 'category_encoders.cat_boost.CatBoostEncoder'> 0.6344820159287033

<class 'category_encoders.hashing.HashingEncoder'> 0.6344820159287033

<class 'category_encoders.helmert.HelmertEncoder'> 0.6344820159287033

<class 'category_encoders.james_stein.JamesSteinEncoder'> 0.6344820159287033

<class 'category_encoders.one_hot.OneHotEncoder'> 0.6344820159287033

<class 'category_encoders.leave_one_out.LeaveOneOutEncoder'> 0.4288610038610039

<class 'category_encoders.m_estimate.MEstimateEncoder'> 0.6344820159287033

<class 'category_encoders.ordinal.OrdinalEncoder'> 0.6344820159287033

<class 'category_encoders.polynomial.PolynomialEncoder'> 0.6344820159287033

<class 'category_encoders.sum_coding.SumEncoder'> 0.6344820159287033

<class 'category_encoders.target_encoder.TargetEncoder'> 0.6344820159287033

<class 'category_encoders.woe.WOEEncoder'> 0.6344820159287033

Разбиение на вещественные, категориальные и бинарные признаки в основном ноутбуке я не стал делать, поскольку выгоды это никакой не принесло. Пытался провести кодировку категориальных признаков различными методами, но почти все они приводили к одинаковой метрике, более низкой, чем без кодировки. Вообще групповая обработка признаков это как групповой секс: суеты много а толку никакого. Нужна углубленная проработка каждого признака. Но представляете сколько на это требуется времени? И что самое обидное, это не принесет какого-то значительного увеличения метрики. Поэтому я сосредоточился на генерации фичей.

EDA2

[114...

```
for i in big_nunique_features:
  print(f"График {i}")
  stats.probplot(df_selected[i], dist='norm', plot=pylab)
  pylab.show()
```

График 238

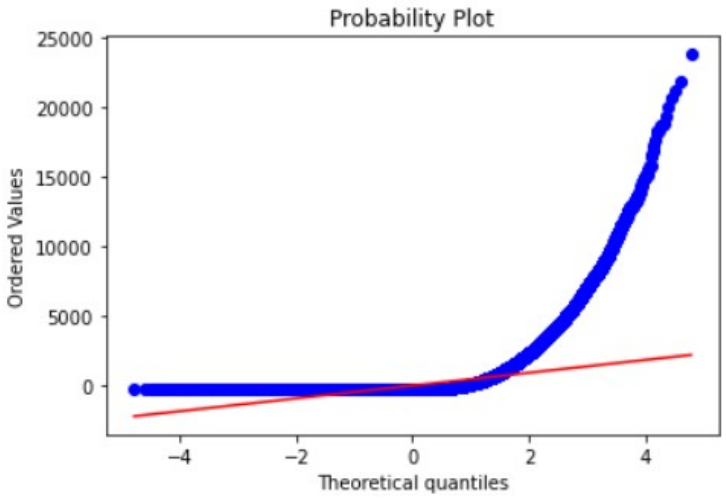
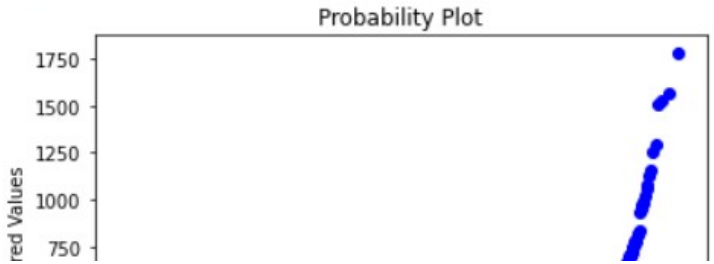


График 167



Не принесла пользы и групповая обработка выбросов, которые есть практически у всех вещественных признаков справа. Правда обработка была самая простая через SimpleImputer.

EDA3

График 134

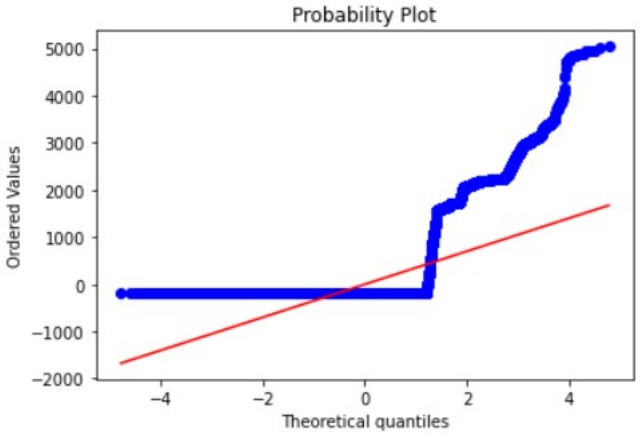
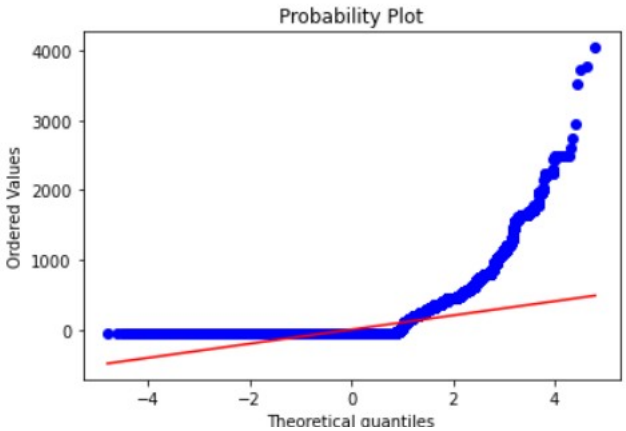


График 223



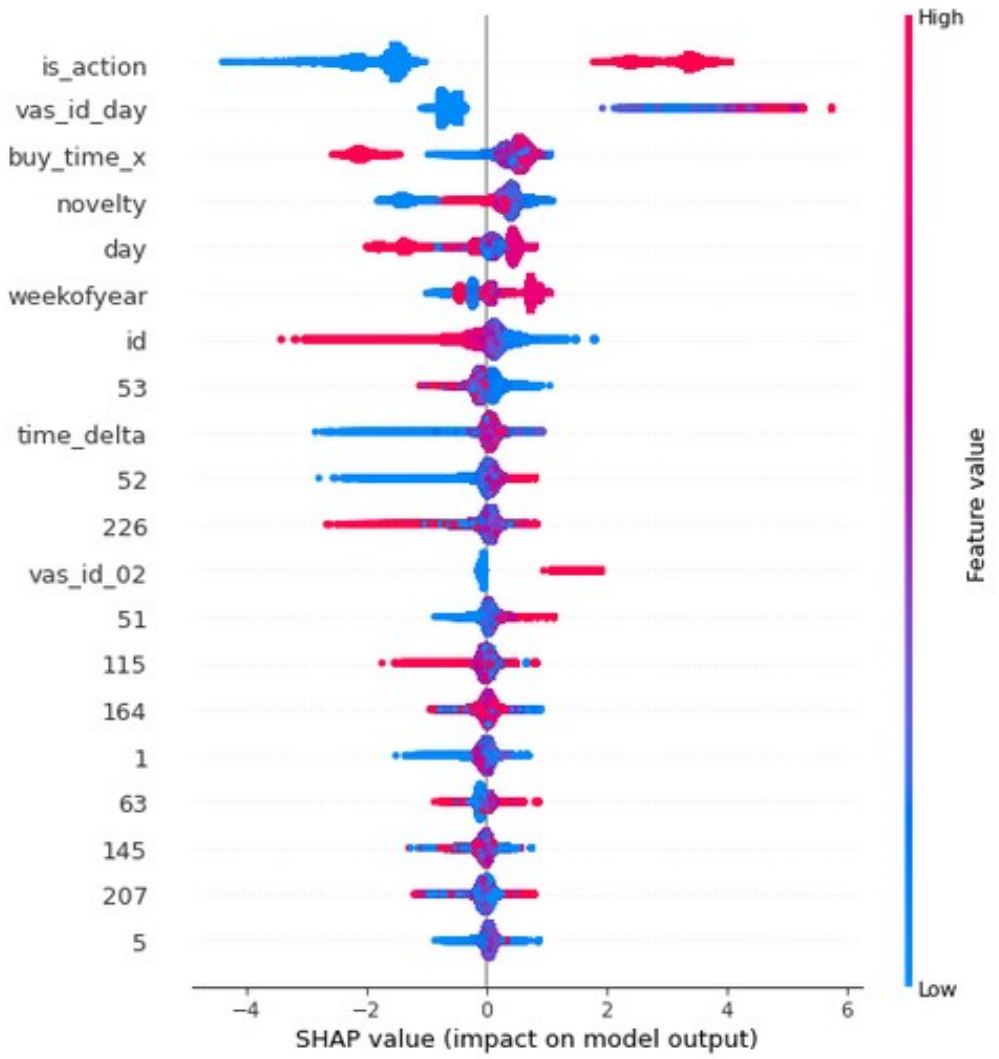
Не принесло пользы и групповое логарифмировани е вещественных признаков, хотя по отдельным признакам логарифмировани е просто плачет.

FeatureGeneration

Первым делом из buy_time вытащил все возможные признаки. Но не все признаки «зашли», и их я не стал включать в датасет. Также хорошо «зашла» фича ‘novelty’. Это разница по времени между временем предложения подключения услуги и текущим моментом.

```
X['time_max'] = X.buy_time_x.max()
X['novelty'] = X['time_max'] - X['buy_time_x']
```

FeatureGeneration2



Очень хорошо, согласно SHAP, зашла фича логирования количества предложений юзеоу по разным vaz_id (что естественно, учитывая тот факт: если юзеру посылается уже десятое предложение на подключение услуги, вероятность ее подключения юзером будет меньше, нежели это первое предложение ее подключения, правда следует еще учитывать фактор времени, но для этого требуются новые фичи) и ее производная. Они же, по моим представлениям должны участвовать наряду со временными фичами в создании рекомендательной системы.

Исследование корреляции признаков

Корреляция признаков была исследована по двум моделям: Pearson Correlation и Spearman Correlation. Ничего критического для «деревянных» моделей не выявлено.

Pearson Correlation

Ввод [ ]:
```
plt.figure(figsize = (100,100))

sns.set(font_scale=10)

pearson_matrix = train_df1_scaled.corr(method='pearson')

pearson_matrix = np.round(pearson_matrix, 2)  # округлим до 2 знаков
pearson_matrix[np.abs(pearson_matrix) < 0.7] = 0  # если корреляция меньше 0,3, то мы заменим на 0

sns.heatmap(pearson_matrix, annot=True, linewidths=.5, cmap='coolwarm')

plt.title('Correlation pearson matrix')
plt.show()
```

Spearman Correlations

Ввод [ ]:
```
plt.figure(figsize = (15,10))

sns.set(font_scale=1.4)

spearman_matrix = train_df1_scaled.corr(method='spearman')

spearman_matrix = np.round(spearman_matrix, 2)  # округлим до 2 знаков
spearman_matrix[np.abs(spearman_matrix) < 0.7] = 0  # если корреляция меньше 0,3, то мы заменим на 0

sns.heatmap(spearman_matrix, annot=True, linewidths=.5, cmap='coolwarm')

plt.title('Correlation spearman matrix')
plt.show()
```

Ну в целом ничего страшного, тем более "деревянные" модели умеют работать с коррелированными признаками.

Разбиение эстиматора на 8 моделей по vas_id

```
predicted = estimator.predict(test_df)
```

	precision	recall	f1-score	support
0.0	0.96	0.97	0.97	308588
1.0	0.56	0.50	0.53	24074
accuracy			0.94	332662
macro avg	0.76	0.73	0.75	332662
weighted avg	0.93	0.94	0.93	332662

Прогнал расчет метрики через общий датасет и 8 датасетов разбитых по vaz_id. 8 датасетов показали общую метрику на 1% выше на тестовой выборке. Но распределение по ‘target’ по разным vaz_id разное, разные распределения и по другим фичам. Кроме того, разбивая на 8 датасетов по vaz_id мы облегчаем работу модели, а недостаток информации по взаимодействию юзера с другими vaz_id можно компенсировать фичами логирования. И на базе этих 8 моделей уже можно строить рекомендательную систему. Настройка отдельных 8 моделей привела к увеличению метрики на валидации на 7% при стабильной метрике на кроссвалидации.

Настройка моделей

	precision	recall	f1-score	support
0.0	0.89	0.98	0.93	182747
1.0	0.98	0.87	0.92	180147
accuracy			0.93	362894
macro avg	0.93	0.93	0.93	362894
weighted avg	0.93	0.93	0.93	362894

TEST

	precision	recall	f1-score	support
0.0	0.99	0.98	0.99	121764
1.0	0.39	0.66	0.49	2265
accuracy			0.97	124029
macro avg	0.69	0.82	0.74	124029
weighted avg	0.98	0.97	0.98	124029

CONFUSION MATRIX

col_0	0.0	1.0
target		
0.0	119390	2374
1.0	776	1489

macro avg 0.78 0.87 0.82 28204

71]:
```
run_cv(model_lgbm, kfold_cv, X_train_balanced, y_train_ba
```

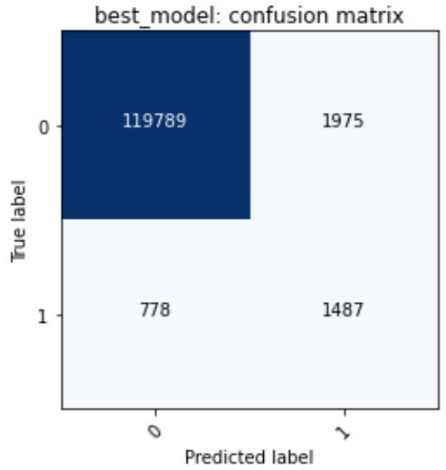
model lgbm: f1 macro = 0.93 (+/- 0.00)

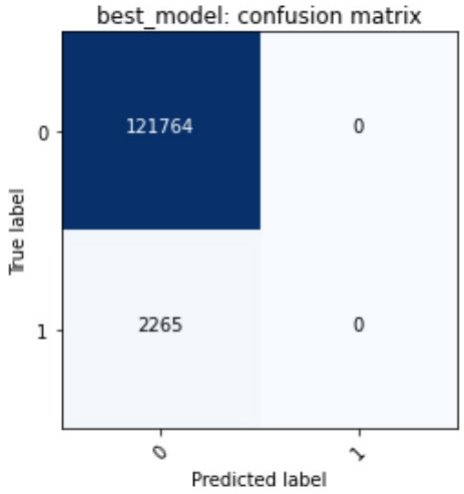
В связи с нехваткой времени провел выбор и настройку моделей и фичей в автоматическом режиме одним и тем же кодом. Практически у всех моделей очень высокая метрика кроссвалидации на тренировочном датасете и низкая на валидационном. Поэтому, из-за нехватки времени, провел выбор настройки модели только по двум фичам: глубине и количестве деревьев. И отдельно настраивался по уменьшению переобучения.

Предварительно по графикам выбирал значения фичей, показавшие себя хорошо при их изолированном использовании.

Практически по всем моделям разными методами удалось убрать переобучение при сохранении стабильной метрики на кроссвалидации. Избегал использование трешхолда, чтобы не пришлось для модели с трешхолдом писать отдельный дочерний класс.

Создание пайплайнов из моделей





При создании пайплайнов с предсказательными моделями, столкнулся с тем что предсказания без пайплайна и с пайплайном значительно различаются, поэтому сохранял для предсказаний не пайплайн а саму модель.

Отбор фичей

[93]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
```

[94]:
```
lr = LinearRegression()
#select 10 the most informative features
rfe = RFE(lr)
selector = rfe.fit(X_train, y_train)
```

[95]:
```
selector.ranking_
```

Out[95]:
```
array([ 94,  95,   1,  67,   1,  59,   1,  60,   1,  62,   1,  51,   1,
         1,   1,  47,   1,   1,   1,   1,  46,   6,   1,   1,   1,  61,
         1,  83,   1,   1,   1,  33,   1,   1,   1,   9,  77,  80,  91,
         1,   1,  39,  27,   1,  49,  42,   1,  76,  10,   1,   1,   1,
         1,   1,   1,   1,   1,   7,   1,   1,   5,   2,  25,  11,  66,
        34,   1,  23,  24,  92,  69,   1,   1,  20,   1,   1,  22,   3,
        21,  35,   1,  17,   1,   1,  13,   1,   1,  90,  48,   1,   1,
         1,   1,   1,   1,   4,  16,  30,   1,   1,  75,   1,   1,   1,
         1,  40,   1,   1,   1,  43,  70,  12,   1,   1,   1,   1,   1,
         1,   1,   1,   1,   1,   1,   1,   1,   1,  37,  38,  71,  58,
        36,   1,  18,  50,  32,   1,   1,   1,  28,   1,   8,  57,   1,
        26,   1,   1,   1,  72,   1,   1,   1,   1,   1,   1,   1,   1,
         1,   1,  93,  96,  29,  68, 100,   1,  99,  98,  41,   1, 101,
        31, 103, 102,  14,  56,  82,  84,  44,  74,  64,  45,  87,  54,
        15,  55,  53,  52,  88,  63,  89,  86,  81,  73,  85,  79,  65,
        78,  19,   1,  97,   1,   1,   1,   1])
```

Отбор фичей для модели тоже произведен в автоматическом режиме. Первым делом произведена попытка отсечь фичи, которым модель дает featuer_importance == 0. И второй ступенью произведена попытка отсечь ненужные фичи с помощью метода RFE из sklearn.feature_selection. Но для ускорения использовал не выбранную предсказательную модель, а линейную регрессию.

Возможные ошибки

- Фича ‘is_action’ занимает 1-е место по важности для всех моделей. Но возможно она недообучена из-за разделения датасета по фиче ‘target’. И это приведет к большой ошибке.
- Возможно в тестовых датасетах не следовало дополнять фичи ‘vas_id_0X’ данными из тестовых датасетов. Это может привести к искажению ее смысла и искажению фичи ‘vas_id_day’. То же самое для фичи ‘novelty’.
- Поскольку не было разбиения датасета по времени, модель могла неправильно применить фичи связанные со временем и требуется их проверка при таком разбиении без фичи ‘is_action’.
- Скорее всего фичу ‘is_action’ нужно было обучать на отдельных датасетах, разбитых по ‘vas_id’, т.к. временные графики по фиче ‘target’ значительно различаются.
- Возможно ошибкой было бороться с переобучением путем штрафования нулей или единиц с помощью class_weight, хотя я проверял после этого модель на кроссвалидации, лучше воспользоваться изменением трешхолда. Это более проверенный метод.
- Можно было попробовать общий датасет для создания временных фичей и фичей логирования разбить по времени, а отдельные датафреймы по ‘vas_id’ для обучения моделей и создания фичи ‘is_action’ разбить по ‘target’.

TO DO:

1. Выборочно обработать редкие значения в категориальных признаках.
2. Выборочно прокодировать категориальные признаки.
3. С категориальными признаками классификаторы работают лучше чем с вещественными. Выборочно перевести вещественные фичи в категориальные. Часть вещественных перейдет в категориальные при помощи RareLabelEncoder .
4. Выборочно прологарифмировать вещественные признаки.
5. Выборочно обработать выбросы по вещественным признакам, после этого часть вещественных перейдут в категориальные.
6. Создать таргет энкодинг по фиче ‘vas_id’. Но после этого нужно будет модель к датасету подключать через лямбда-функцию (если таргет энкодинг по фиче ‘vas_id’==1 выдавать ноль, а если таргет энкодинг по фиче ‘vas_id’==0, то выдавать предсказание. То есть если услуга уже подключена, то выдавать предсказание ноль).
6. Нагенерировать фичей логирования и их производных, в том числе с временными фичами.
7. Нагенерировать фичи с помощью скриптов для автоматического генерирования фичей.
8. Настроить отбор фичей с помощью RFE где эстиматором будет выступать не логарифмическая регрессия а выбранная модель. Попробовать другие методы для этого.
9. Выборочно сгенерировать фичи с помощью таргет энкодинга.
10. Произвести более глубокую настройку моделей на большем количестве параметров. Настроить трешхолд для моделей.
11. Поработать с разделением на кластеры.
12. Настроить более точный predict_proba с помощью

Создание рекомендательной системы

- Ну сразу же надо определиться с vas_id==6. Тут надо либо все таки получить либо высокую метрику. Либо, поскольку она достаточно популярна, попробовать uplift-модель. Для остальных вполне нормально зайдет response-модель.
- В связи с почти бесплатной для Мегафона стоимостью СМС, в качестве метрики предлагаю использовать recall, с разрешения клиента на рассылки конечно (чего смски на складе будут пылиться?) либо Fb_score в случае убыточности recall. Коэффициент b следует подобрать в зависимости от окупаемости акции.
- Для решения о рассылке одной вероятности подключения будет маловато. Ну в первую очередь конечно бы узнать не подключена ли уже услуга. И представьте что мы клиенту у которого высокая вероятность подключения услуги будем по 10 раз на дню слать приглашение? Для принятия решения о рассылке нужна еще фича или несколько фичей, учитывающих количество посланных ему приглашений (наша „vas_id_0X“), давность последнего приглашения (наша „novelty“), частота посылаемых приглашений, «старость» клиента, разница по времени между первым и последним приглашением. Другие услуги он покупает по акциям или нет и вообще покупал ли? На таком как у нас бедном датасете веса этих признаков придется подбирать опытным путем. А если датасет богатый на взаимодействия, можно с помощью модели, например логистической регрессии.