[Отбор признаков, часть 2 (на примере задачи Porto Seguro’s Safe Driver Prediction с Kaggle, 11-е место) — ИЦ "ГЕВИССТА" (wordpress.com)](https://gewissta673156975.wordpress.com/2022/02/26/%D0%BE%D1%82%D0%B1%D0%BE%D1%80-%D0%BF%D1%80%D0%B8%D0%B7%D0%BD%D0%B0%D0%BA%D0%BE%D0%B2-%D1%87%D0%B0%D1%81%D1%82%D1%8C-2-%D0%BD%D0%B0-%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D1%80%D0%B5-%D0%B7/)

**Отбор признаков, часть 2 (на примере задачи Porto Seguro’s Safe Driver Prediction с Kaggle, 11-е место)**

[RC Gewissta](https://gewissta673156975.wordpress.com/author/rcgewissta/)  [Отбор признаков](https://gewissta673156975.wordpress.com/category/%d0%be%d1%82%d0%b1%d0%be%d1%80-%d0%bf%d1%80%d0%b8%d0%b7%d0%bd%d0%b0%d0%ba%d0%be%d0%b2/)  26 февраля 2022 6 Minutes

*Ничто так не испортит острые ощущения от покупки нового автомобиля, как стоимость страховки. Переживания становятся еще более болезненными, когда ты знаешь, что являешься хорошим водителем. Кажется несправедливым, что вам нужно платить так много, если вы годами осторожничали на дороге. С этим полностью согласна одна из крупнейших в Бразилии компаний по страхованию автомобилей Porto Seguro. Неточности в прогнозах страховых компаний по страхованию автомобилей увеличивают стоимость страховки для хороших водителей и снижают стоимость для плохих.*

*В этом соревновании вам предстоит построить модель, которая прогнозирует вероятность того, что водитель продлит автостраховку в следующем году. Хотя Porto Seguro использует машинное обучение в течение последних 20 лет, они надеются, что сообщество Kaggle по машинному обучению предложит новые, более эффективные методы. Более точный прогноз позволит им еще больше адаптировать свои цены и, надеюсь, сделает автострахование более доступным для большего числа водителей.*

С этой преамбулы начинается соревнование Porto Seguro’s Safe Driver Prediction [https://www.kaggle.com/c/porto-seguro-safe-driver-prediction](http://me.dm/r-Cue7bIckyE?source=email-3bae8303d77e-1634379352451-newsletter.subscribeToProfile-------------------------4d3b9525_614e_4d7a_ab0d_2834f266db36--------cb5a35dcd36) c Kaggle, на нем мы и проиллюстрируем, как отбор признаков позволяет попасть в топ-20. Оптимизируемой метрикой будет нормализованный коэффициент Джини. Для ее оптимизации вполне подойдет максимизация AUC-ROC.

Импортируем необходимые библиотеки и выведем версии библиотек.

*# загружаем необходимые библиотеки, классы и функции*

import pandas as pd

import numpy as np

from sklearn.model\_selection import (train\_test\_split,

cross\_val\_score)

from sklearn.metrics import roc\_auc\_score

from sklearn.model\_selection import GridSearchCV

from catboost import CatBoostClassifier, Pool

from lightgbm import LGBMClassifier

from xgboost import XGBClassifier

from rfpimp import (feature\_dependence\_matrix,

plot\_dependence\_heatmap,

plot\_corr\_heatmap)

from sklearn.model\_selection import cross\_validate

import shap

import matplotlib.pyplot as plt

%matplotlib inline

%config InlineBackend.figure\_format = 'retina'

*# отключаем экспоненциальное представление и увеличиваем*

*# максимальное количество отображаемых столбцов*

pd.set\_option('display.float\_format', lambda x: '%.8f' % x)

pd.set\_option('display.max\_columns', 60)

Изображение выглядит как текст

Автоматически созданное описание

Теперь загружаем исторический набор.

*# загружаем данные*

data = pd.read\_csv('Data/porto\_seguro\_train.csv')

data.head()

**Output:**

Изображение выглядит как стол

Автоматически созданное описание

Удалим идентификатор, убедимся в отсутствии пропусков и посмотрим типы данных.

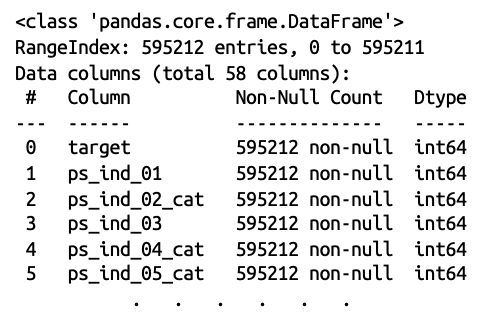
*# удаляем id*

data.drop('id', axis=1, inplace=True)

*# смотрим пропуски и типы данных*

data.info()

**Output:**



Формально пропусков нет, однако некоторые участники соревнования рассматривали значения -1 в переменных как пропуски и применяли разные стратегии импутации.

Разбиваем набор на обучающую и тестовую выборки.

*# разбиваем данные на обучающие и тестовые:*

*# получаем обучающий массив признаков,*

*# тестовый массив признаков, обучающий*

*# массив меток, тестовый массив меток*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

data.drop('target', axis=1),

data['target'],

test\_size=0.3,

stratify=data['target'],

random\_state=42)

Теперь нужно вычислить важности признаков на основе информационного выигрыша с помощью деревьев градиентного бустинга LightGBM.

Чтобы оценки важностей были надежными, нужно настроить гиперпараметры градиентного бустинга и постараться максимизировать метрику AUC-ROC.

Сначала ищем оптимальный темп обучения для зафиксированного количества деревьев с помощью обычного поиска по сетке.

*# создаем экземляр класса LGBMClassifier*

lgbm\_model = LGBMClassifier(random\_state=42,

n\_estimators=300)

*# задаем сетку гиперпараметров*

param\_grid = {'learning\_rate': [0.01, 0.05, 0.1]}

*# создаем экземпляр класса GridSearchCV,*

*# передав конвейер, сетку гиперпараметров*

*# и указав количество блоков перекрестной*

*# проверки, отключив запись метрик для*

*# обучающих блоков перекрестной проверки*

*# в атрибут cv\_results\_*

gs = GridSearchCV(lgbm\_model,

param\_grid,

scoring='roc\_auc',

cv=5,

return\_train\_score=False)

*# выполняем поиск по всем значениям сетки*

gs.fit(X\_train, y\_train);

Взглянем на результаты оптимизации.

*# смотрим наилучшие значения гиперпараметров*

print('Наилучшие значения гиперпараметров: {}'.format(

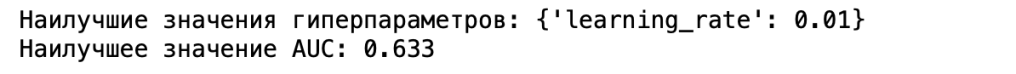
gs.best\_params\_))

*# смотрим наилучшее значение AUC*

print('Наилучшее значение AUC: {:.3f}'.format(

gs.best\_score\_))

**Output:**



Теперь с найденным оптимальным темпом обучения для зафиксированного количества деревьев ищем с помощью обычного поиска по сетке оптимальные значения гиперпараметров lambda\_l1, bagging\_fraction, feature\_fraction. Особенность этого соревнования заключалась в том, что большее значение lambda\_l1 (по умолчанию 0) улучшало результат.

*# создаем экземляр класса LGBMClassifier*

lgbm\_model2 = LGBMClassifier(random\_state=42,

n\_estimators=300,

learning\_rate=0.01)

*# задаем сетку гиперпараметров*

param\_grid2 = {

'lambda\_l1': [0, 10],

'bagging\_fraction': [0.5, 1],

'feature\_fraction': [0.5, 1]

}

*# создаем экземпляр класса GridSearchCV,*

*# передав конвейер, сетку гиперпараметров*

*# и указав количество блоков перекрестной*

*# проверки, отключив запись метрик для*

*# обучающих блоков перекрестной проверки*

*# в атрибут cv\_results\_*

gs2 = GridSearchCV(lgbm\_model2,

param\_grid2,

scoring='roc\_auc',

cv=5,

return\_train\_score=False)

*# выполняем поиск по всем значениям сетки*

gs2.fit(X\_train, y\_train);

Взглянем на результаты оптимизации.

*# смотрим наилучшие значения гиперпараметров*

print('Наилучшие значения гиперпараметров: {}'.format(

gs2.best\_params\_))

*# смотрим наилучшее значение AUC*

print('Наилучшее значение AUC: {:.3f}'.format(

gs2.best\_score\_))

**Output:**



Теперь вычислим важности признаков на основе информационного выигрыша. Для надежности воспользуемся перекрестной проверкой: построим 5 моделей LightGBM с найденными значениями гиперпараметров, вычислим на основе каждой модели важности признаков и усредним.

*# создаем экземляр класса LGBMClassifier*

model\_all\_features = LGBMClassifier(

random\_state=42,

learning\_rate=0.01,

n\_estimators=300,

bagging\_fraction=0.5,

feature\_fraction=1,

lambda\_l1=10,

importance\_type='gain')

*# выполняем перекрестную проверку и сохраняем*

*# результат с помощью функции cross\_validate()*

output = cross\_validate(

model\_all\_features,

X\_train,

y\_train,

cv=5,

scoring='roc\_auc',

return\_estimator=True)

*# создаем список fi, в который будем сохранять*

*# важности признаков, и сохраняем в него важности,*

*# рассчитанные для каждой из моделей*

fi = []

for estimator in output['estimator']:

fi.append(estimator.feature\_importances\_)

*# преобразовываем список в датафрейм, индексы*

*# в котором будут именами наших переменных*

fi = pd.DataFrame(

np.array(fi).T,

columns=['importance ' + str(idx)

for idx in range(len(fi))],

index=X\_train.columns)

*# вычисляем усредненные важности*

*# и добавляем столбец с ними*

fi['mean\_importance'] = fi.mean(axis=1)

*# смотрим полученный датафрейм*

fi

**Output:**

Изображение выглядит как стол

Автоматически созданное описание

Здесь мы видим, что некоторые переменные имеют нулевые или очень низкие важности. Речь идет о переменных *ps\_calc\_16\_bin*, *ps\_car\_10\_cat*, *ps\_ind\_13\_bin*, *ps\_ind\_12\_bin*, *ps\_ind\_10\_bin*, *ps\_ind\_11\_bin*, *ps\_ind\_14*. Эти переменные можно рассматривать для удаления первыми.

Теперь записываем серию, в которой индексные метки — признаки, значения — усредненные важности. Затем сортируем индексные метки (признаки) по возрастанию усредненных важностей.

*# записываем серию, в которой индексные*

*# метки - признаки, значения - важности*

features = fi['mean\_importance']

*# сортируем индексные метки*

*# по возрастанию важностей*

features = features.sort\_values(ascending=True)

features

**Output:**

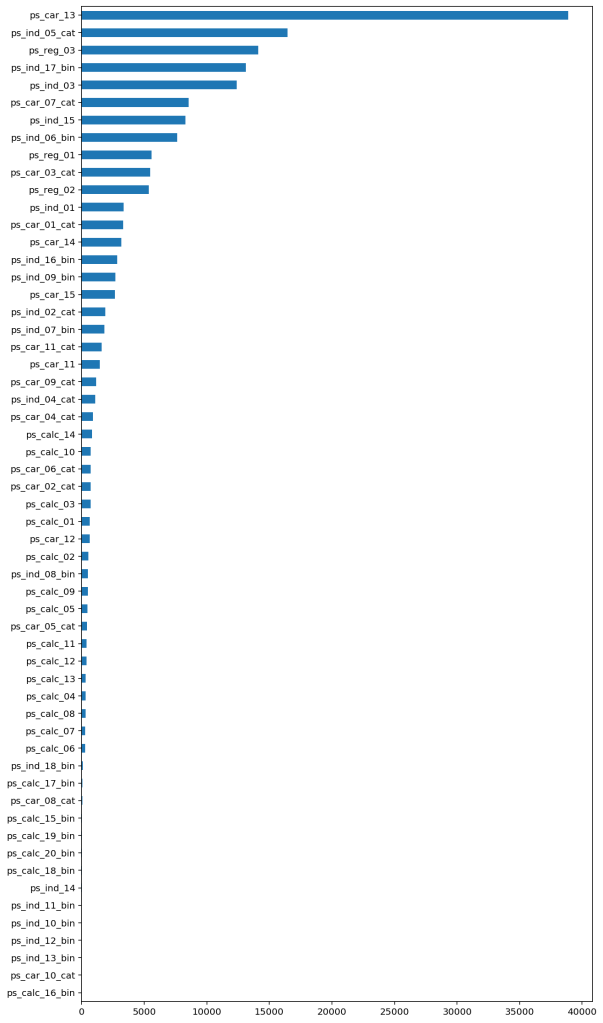


Визуализируем график усредненных важностей.

*# выводим график усредненных важностей*

features.plot.barh(figsize=(10, 20));

**Output:**



Давайте создадим список из индекса серии.

*# создаем список признаков*

features = list(features.index)

features

**Output:**

Изображение выглядит как текст

Автоматически созданное описание

Вычисляем усредненную метрику качества для модели со всеми переменными.

*# вычисляем AUC, усредненный по проверочным блокам*

*# всех моделей перекрестной проверки*

auc\_score\_all = output['test\_score'].mean()

auc\_score\_all

**Output:**



Теперь попробуем уже знакомый гибридный подход к отбору признаков. В нем мы сочетаем встроенный метод (вычисление важностей) с методом-оберткой (рекурсивным жадным удалением признаков). Мы отсортируем признаки по мере возрастания важности (воспользуемся оценками важностей с т.з. информационного выигрыша, усредненными по итогам перекрестной проверки), сформируем список переменных, отсортированных по мере возрастания усредненной важности, и пойдем по нему, строя модель, каждый раз исключая признак из списка. Предварительно мы зададим пороговое значение разницы между AUC-ROC модели со всеми признаками (только что вычислили его) и AUC-ROC модели с удаленным признаком. Если значение разницы является отрицательным, значит удаление признака повысило AUC-ROC. Например, оценка AUC-ROC модели со всеми признаками равна 0,80, а оценка AUC-ROC модели с конкретным удаленным признаком равна 0,81, тогда разница 0,80–0,81 = –0,01. Если значение разницы меньше порогового значения, то удаляем признак. Например, мы задали пороговое значение 0,01. Разница AUC-ROC составляет –0,01, что меньше порогового значения, тогда удаляем признак. Если значение разницы больше или равно пороговому значению, то сохраняем признак. Допустим, разница AUC-ROC равна 0,04, что больше порогового значения, сохраняем признак. Нас, конечно, будут интересовать признаки, дающие отрицательную разницу, такие переменные будут первыми кандидатами на удаление.

Помним, что для надежности лучше использовать оценку не одной модели, а нескольких моделей в рамках процедуры перекрестной проверки. Кроме того, помните, что пороговое значение разницы между AUC-ROC модели со всеми признаками и AUC-ROC модели с удаленным признаком — это гиперпараметр, который настраивается либо на проверочной выборке, либо на проверочных блоках перекрестной проверки, итоговую оценку качества модели, построенную на наборе с отобранными признаками (а пространство этих отобранных признаков будет зависеть от порогового значения разницы) нужно получать на независимой тестовой выборке. Здесь мы перебирали разные пороговые значения (для экономии этап пропущен), если используется одно пороговое значение, исходя из каких-то предварительных данных, можно обойтись без независимой тестовой выборки.

*# задаем пороговое значение разницы AUC*

tol = 0.0001

print('выполнение последовательного' +

'удаления признаков')

*# создаем список, в который будем*

*# записывать удаляемые признаки*

features\_to\_remove = []

*# создаем список, в который будем*

*# записывать значение AUC*

auc\_score\_mean\_list = []

*# создаем список, в который будем*

*# записывать разницу AUC*

diff\_auc\_list = []

*# задаем счетчик для оценки прогресса*

count = 1

*# итерируем по всем признакам, признаки*

*# упорядочены по важности - мере возрастания*

*# информационного выигрыша*

for feature in features:

print()

print('проверяемый признак: ', feature,

' признак ', count,

' из ', len(features))

count = count + 1

*# создаем экземляр класса LGBMClassifier*

model = LGBMClassifier(random\_state=42,

learning\_rate=0.01,

n\_estimators=300,

bagging\_fraction=0.5,

feature\_fraction=1,

lambda\_l1=10)

*# обучаем модели со всеми признаками минус*

*# уже удаленные признаки (берем их из списка*

*# удаляемых признаков) и оцениваемый признак*

auc\_scores = cross\_val\_score(

model,

X\_train.drop(

features\_to\_remove + [feature], axis=1),

y\_train,

scoring='roc\_auc',

cv=5)

*# вычисляем AUC, усредненный по проверочным блокам*

*# всех моделей перекрестной проверки*

auc\_score\_mean = auc\_scores.mean()

*# печатаем усредненное значение AUC*

print('AUC модели после удаления={}'.format(

auc\_score\_mean))

*# добавляем усредненное значение AUC в список*

auc\_score\_mean\_list.append(auc\_score\_mean)

*# печатаем AUC модели со всеми признаками*

*# (опорное значение AUC)*

print('AUC модели со всеми признаками={}'.format(

auc\_score\_all))

*# определяем разницу AUC (если отрицательное*

*# значение - удаление признака улучшило AUC)*

diff\_auc = auc\_score\_all - auc\_score\_mean

*# записываем разницу AUC в список*

diff\_auc\_list.append(diff\_auc)

*# сравниваем разницу AUC с порогом, заданным*

*# заранее, если разница AUC больше или равна*

*# порогу, сохраняем*

if diff\_auc >= tol:

print('Разница AUC={}'.format(diff\_auc))

print('сохраняем: ', feature)

print

*# если разница AUC меньше порога, удаляем*

else:

print('Разница AUC={}'.format(diff\_auc))

print('удаляем: ', feature)

print

*# если разница AUC меньше порога и мы удаляем*

*# признак, мы в качестве нового опорного значения*

*# AUC задаем значение AUC для модели с оставшимися*

*# признаками*

auc\_score\_all = auc\_score\_mean

*# добавляем удаляемый признак в список*

features\_to\_remove.append(feature)

*# формируем датафрейм*

df = pd.DataFrame(

{'feature': features,

'auc\_score\_mean': auc\_score\_mean\_list,

'diff\_auc\_score': diff\_auc\_list})

*# цикл завершен, вычисляем количество*

*# удаленных признаков*

print('ВЫПОЛНЕНО!!')

print('общее количество признаков для удаления: ',

len(features\_to\_remove))

*# определяем признаки, которые мы*

*# хотим сохранить (не удаляем)*

features\_to\_keep = [x for x in features

if x not in features\_to\_remove]

print('общее количество признаков для сохранения: ',

len(features\_to\_keep))

**Output:**

Изображение выглядит как текст

Автоматически созданное описание

Большая часть признаков, в названии которых упоминается calc, имеют отрицательные важности. Обязательно нужно попробовать построить модель без этих признаков.

*# выводим получившийся датафрейм*

df.sort\_values(by='diff\_auc\_score', ascending=False)

**Output:**

Изображение выглядит как стол

Автоматически созданное описание

Еще можно попробовать подход, в ходе которого строим модели LightGBM, увеличивая глубину, и смотрим важности признаков. Здесь нам важно понять, как быстро признаки «включаются в работу», т. е. начинают использоваться в качестве предикторов расщепления. Чем раньше, т. е. чем меньше глубина использования признака, тем важнее признак.

*# еще один подход - смотрим, как меняются важности*

*# признаков по мере увеличения глубины: наиболее*

*# важные признаки - те, которые начинают*

*# использоваться раньше остальных*

*# задаем сетку значений глубины*

max\_depth\_grid = [1, 2, 3, 4, 5]

*# создаем список fi, в который будем сохранять*

*# важности признаков, и сохраняем в него важности,*

*# рассчитанные для каждой из моделей*

fi = []

*# обучаем модели с разными значениями глубины и*

*# получаем важности, записываем важности в список*

for max\_depth in max\_depth\_grid:

model\_all\_features = LGBMClassifier(

random\_state=42,

learning\_rate=0.01,

n\_estimators=300,

bagging\_fraction=0.5,

feature\_fraction=1,

lambda\_l1=10,

max\_depth=max\_depth,

importance\_type='gain')

model\_all\_features.fit(X\_train, y\_train)

fi.append(model\_all\_features.feature\_importances\_)

*# преобразовываем список в датафрейм, индексы*

*# в котором будут именами наших переменных*

fi = pd.DataFrame(

np.array(fi).T,

columns=['importance ' + str(idx)

for idx in range(len(fi))],

index=X\_train.columns)

*# вычисляем усредненные важности*

*# и добавляем столбец с ними*

fi['mean\_importance'] = fi.mean(axis=1)

*# сортируем по убыванию усредненных важностей*

fi = fi.sort\_values(by='mean\_importance',

ascending=False)

*# смотрим полученный датафрейм*

fi

**Output:**

Изображение выглядит как стол

Автоматически созданное описание

Опять видим, что признаки с префиксом calc, а также признаки *ps\_car\_08\_cat*, *ps\_car\_10\_cat*, *ps\_ind\_13\_bin*, *ps\_ind\_12\_bin*, *ps\_ind\_10\_bin*, *ps\_ind\_11\_bin*, *ps\_ind\_14*начинают использоваться деревьями позже остальных (нулевые значения). Потенциально это также может говорить о низкой важности этих признаков. Примечательно, что почти все эти признаки, кроме *ps\_car\_08\_cat*и некоторых признаков c префиксом calc, давали отрицательные разности AUC-ROC.

Наконец, попробуем еще один подход, в ходе мы вычисляем матрицу взаимозависимостей признаков, значениями в этой матрице будут пермутированные важности признаков, с помощью которых мы пытаемся предсказать интересующий признак. Если признак хорошо предсказывается остальными, он будет менее важен, в то же время, если признак не предсказывается вообще остальными признаками, высока вероятность того, что он создан искусственно и скорее всего будет снижать качество модели. Нам потребуются функции feature\_dependence\_matrix(),plot\_dependence\_heatmap() и plot\_corr\_heatmap() из пакета rfimp.

*# вычисляем матрицу взаимозависимостей признаков,*

*# значения - это пермутированные важности*

*# признаков, с помощью которых мы пытаемся*

*# предсказать интересующий признак*

D = feature\_dependence\_matrix(

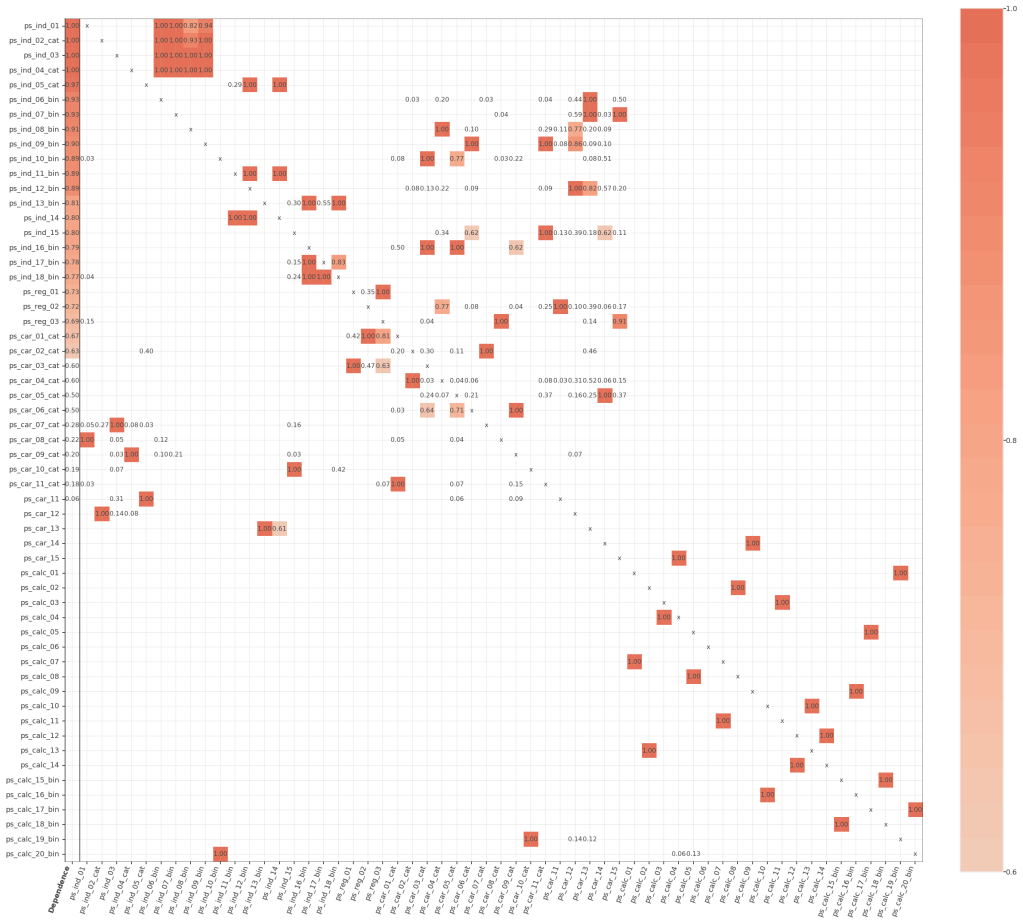
X\_train, sort\_by\_dependence=True)

viz = plot\_dependence\_heatmap(

D, figsize=(18, 18))

viz.view()

**Output:**



Видим, что в предсказании признаков совершенно не участвуют признаки с префиксом calc, а сами признаки с префиксом calc совершенно не предсказываются остальными признаками.

Давайте взглянем на матрицу корреляций на основе ранговой корреляции Спирмена, которая не предполагает линейной взаимосвязи между переменными.

*# выводим матрицу корреляций (на основе*

*# ранговой корреляции Спирмена)*

viz2 = plot\_corr\_heatmap(X\_train,

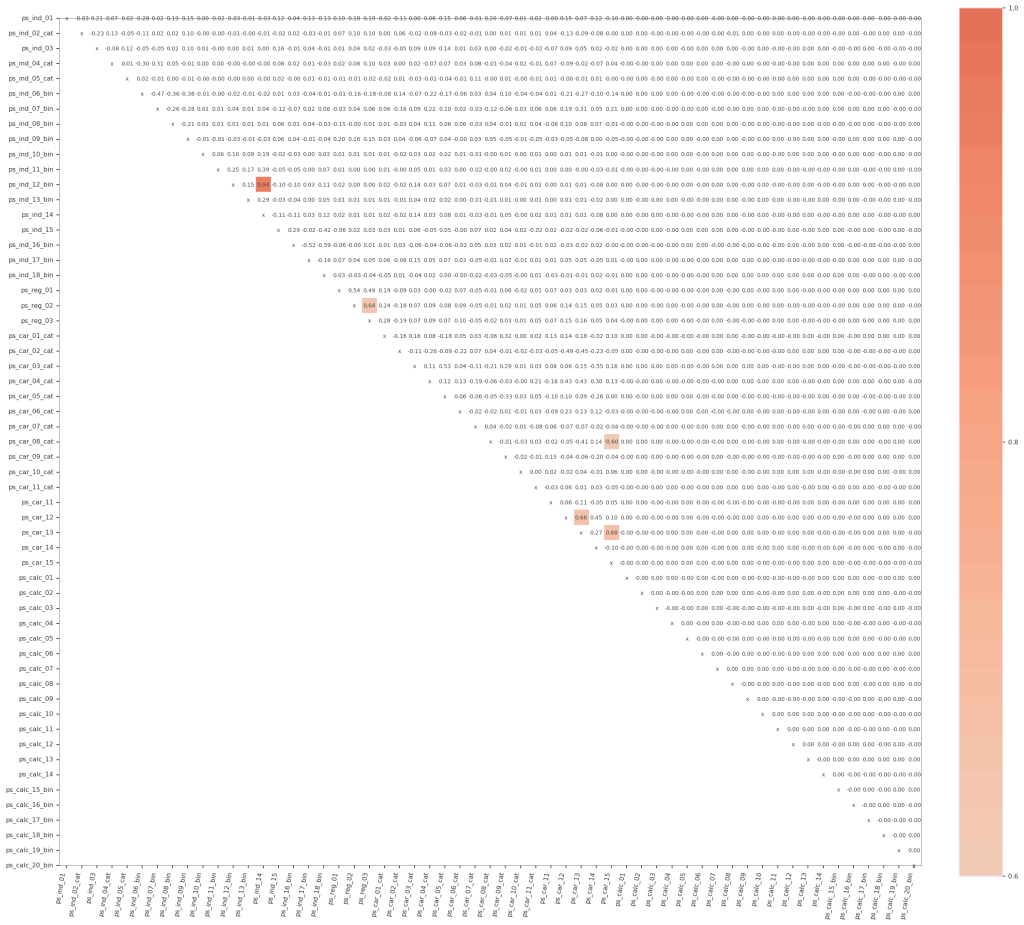
figsize=(18, 18),

label\_fontsize=8,

value\_fontsize=7)

viz2.view()

**Output:**



Видим, что признаки с префиксом calc, имеют нулевые корреляции с остальными признаками, не имеющими префикса calc. Принимаем решение, что не будем использовать признаки с упоминанием calc и следующие признаки:

*ps\_ind\_14*;

*ps\_car\_10\_cat*;

*ps\_ind\_10\_bin*;

*ps\_ind\_11\_bin*;

*ps\_ind\_12\_bin*;

*ps\_ind\_13\_bin*;

*ps\_ind\_18\_bin*.

Начинаем готовить финальное решение на базе трех моделей — LightGBM, CatBoost и XGBoost.

Пишем функцию предварительной подготовки для исторического набора и набора новых данных.

*# пишем функцию предварительной подготовки*

def preprocessing(df, lightgbm=True, newdata=False):

*# удаляем столбцы с calc в названии*

calc\_pat = df.columns.str.contains('calc')

calc\_columns = df.columns[calc\_pat]

df.drop(calc\_columns, axis=1, inplace=True)

*# для новых данных*

if newdata:

*# записываем id набора*

ident = df['id']

*# удаляем id из набора*

df.drop('id', axis=1, inplace=True)

*# для обучающих данных*

else:

*# удаляем id из набора*

df.drop('id', axis=1, inplace=True)

*# формируем массив меток и массив признаков*

labels = df.pop('target')

*# если готовим данные для LightGBM*

if lightgbm:

*# удаляем наименее важные переменные*

some\_columns = ['ps\_ind\_14', 'ps\_car\_10\_cat',

'ps\_ind\_10\_bin', 'ps\_ind\_11\_bin',

'ps\_ind\_12\_bin', 'ps\_ind\_13\_bin',

'ps\_ind\_18\_bin']

df.drop(some\_columns, axis=1, inplace=True)

*# записываем список столбцов с cat в названии*

cat\_pat = df.columns.str.contains('cat')

cat\_columns = df.columns[cat\_pat]

*# формируем массив с ним*

df\_ = df[cat\_columns]

*# столбцам с cat в названии присваиваем тип object*

*# и выполняем дамми-кодирование*

for col in cat\_columns:

df[col] = df[col].astype('object')

df = pd.get\_dummies(df)

*# конкатенируем массив с исходными столбцами с cat*

*# в названии и массив, к которому было применено*

*# дамми-кодирование*

df = pd.concat([df\_, df], axis=1)

*# в противном случае (если готовим данные*

*# для CatBoost и XGBoost)*

else:

*# удаляем наименее важные переменные*

some\_columns = ['ps\_ind\_14', 'ps\_car\_10\_cat',

'ps\_car\_14', 'ps\_ind\_10\_bin',

'ps\_ind\_11\_bin', 'ps\_ind\_12\_bin',

'ps\_ind\_13\_bin', 'ps\_car\_11',

'ps\_car\_12']

df.drop(some\_columns, axis=1, inplace=True)

*# для новых данных*

if newdata:

*# возвращаем преобразованный массив*

*# признаков, идентификатор*

return df, ident

*# для обучающих данных*

else:

*# возвращаем преобразованный массив*

*# признаков, массив меток*

return df, labels

Итак, что мы делаем с помощью нашей функции?

Удаляем переменные с префиксом calc.

Если мы работаем с набором новых данных (здесь под новыми данными подразумевается данные без зависимой переменной), то записываем *id* в переменную *ident* и удаляем *id* из набора. Если же мы работаем с историческими данными, то удаляем *id* из набора и с помощью метода .pop() формируем массив меток и массив признаков.

Если мы делаем предварительную подготовку данных для LightGBM, то удаляем переменные:

*ps\_ind\_14*;

*ps\_car\_10\_cat*;

*ps\_ind\_10\_bin*;

*ps\_ind\_11\_bin*;

*ps\_ind\_12\_bin*;

*ps\_ind\_13\_bin*;

*ps\_ind\_18\_bin*.

Записываем список столбцов с cat в названии и формируем массив с ним. Столбцам с cat в названии присваиваем тип object и выполняем дамми-кодирование. Затем конкатенируем массив с исходными столбцами с cat в названии и массив, к которому было применено дамми-кодирование.

Если мы делаем предварительную подготовку данных для CatBoost и XGBoost, то удаляем переменные:

*ps\_ind\_14*;

*ps\_car\_10\_cat*;

*ps\_car\_14*;

*ps\_ind\_10\_bin*;

*ps\_ind\_11\_bin*;

*ps\_ind\_12\_bin*;

*ps\_ind\_13\_bin*;

*ps\_car\_11*;

*ps\_car\_12*.

Кроме того, вычисление важностей по SHAP с постепенным увеличением глубины для CatBoost и вычисление важностей по информационному выигрышу с постепенным увеличением глубины для XGBoost (для экономии места этап пропущен) показало, что можно удалить *ps\_car\_11*и *ps\_car\_12*.

Если мы работаем с новыми данными, возвращаем преобразованный массив признаков и идентификатор. Если же мы работаем с обучающими данными, возвращаем преобразованный массив признаков и массив меток (его, разумеется, оставляем без изменений).

Загружаем исторический набор и набор новых данных и выполняем предварительную подготовку данных для LightGBM.

*# загружаем наборы*

train = pd.read\_csv('Data/porto\_seguro\_train.csv')

test = pd.read\_csv('Data/porto\_seguro\_test.csv')

*# выполняем предварительную подготовку*

*# данных для LightGBM*

train, labels = preprocessing(

train, lightgbm=True, newdata=False)

test, ident = preprocessing(

test, lightgbm=True, newdata=True)

Теперь строим модель LightGBM на историческом наборе и вычисляем вероятности для набора новых данных. Гиперпараметры предварительно настраивались.

*# создаем экземпляр класса LGBMClassifier*

lightgbm\_model = LGBMClassifier(

random\_state=42,

feature\_fraction=0.4,

lambda\_l1=8,

bagging\_fraction=0.1,

learning\_rate=0.012,

n\_estimators=1600)

*# строим модель на всей обучающей выборке*

lightgbm\_model.fit(train, labels)

*# вычисляем вероятности для тестовой выборки*

lgbm\_preds\_prob = lightgbm\_model.predict\_proba(test)[:, 1]

Загружаем исторический набор и набор новых данных, выполняем предварительную подготовку данных для CatBoost и XGBoost.

*# загружаем наборы*

train = pd.read\_csv('Data/porto\_seguro\_train.csv')

test = pd.read\_csv('Data/porto\_seguro\_test.csv')

*# выполняем предварительную подготовку*

*# данных для CatBoost и XGBoost*

train, labels = preprocessing(

train, lightgbm=False, newdata=False)

test, ident = preprocessing(

test, lightgbm=False, newdata=True)

Формируем массив индексов категориальных признаков и обучающий пул. Затем строим модель CatBoost на историческом наборе и вычисляем вероятности для набора новых данных. Гиперпараметры предварительно настраивались.

*# формируем массив индексов*

*# категориальных признаков*

cat\_features\_ids = np.where(

train.dtypes != np.float)[0]

*# формируем обучающий пул*

train\_pool = Pool(

train, labels,

cat\_features=cat\_features\_ids)

*# создаем экземпляр класса CatBoostClassifier*

catbst\_model = CatBoostClassifier(

iterations=1200,

learning\_rate=0.1,

random\_strength=0.15,

simple\_ctr='Counter:CtrBorderCount=50',

model\_size\_reg=0.1,

max\_depth=4,

random\_seed=0,

logging\_level='Silent')

*# обучаем модель*

catbst\_model.fit(train\_pool)

*# вычисляем вероятности*

catbst\_preds\_prob = catbst\_model.predict\_proba(

test)[:, 1]

Теперь строим модель XGBoost на историческом наборе и вычисляем вероятности для набора новых данных. Гиперпараметры предварительно настраивались.

*# создаем экземпляр класса XGBClassifier*

xgbst\_model = XGBClassifier(

learning\_rate=0.04,

subsample=0.7,

random\_state=42,

max\_depth=4,

n\_estimators=500)

*# обучаем модель*

xgbst\_model.fit(train, labels)

*# вычисляем вероятности*

xgbst\_preds\_prob = xgbst\_model.predict\_proba(

test)[:, 1]

Усредняем вероятности трех моделей, используя вес, при этом модель, дающее меньшее качество, берется с меньшим весом.

*# усредняем вероятности с весами,*

*# учитывающими качество модели*

average\_prob = (catbst\_preds\_prob \* 0.5 +

xgbst\_preds\_prob \* 0.3 +

lgbm\_preds\_prob \* 1) / 3

*# формируем посылку*

pd.DataFrame({'id': ident,

'target': average\_prob}).to\_csv(

'subm\_seguro.csv', index=False)

Отправляем решение. Наш результат соответствует 11-му месту на приватном лидерборде.