

Построение скоринговой модели

Кирщин Иван, БЭАД223

1. Задача

Основной целью работы было построение логистической регрессии для бинарной классификации событий вида дефолт/недефолт поверх WOE-трансформированных признаков.

WOE (weight-of-evidence) трансформация – это дискретизация исходных признаков путем разбиения исходных значений на диапазоны, каждому из которых ставится в соответствие WOE-значение. Категориальные фичи регрессируются (либо остаются неизменными) и группе значений также ставится в соответствие WOE.

Итогом работы стала модель предсказания вероятности дефолта заемщика – физического лица по данным, доступным в заявке.

2. Данные

Модель обучалась на данных площадки онлайн-кредитования Lending Club. Обучающая выборка содержала в себе 61169 записи о результатах выдачи кредитов с апреля 2010 года по май 2014, которые описывались при помощи следующих признаков:

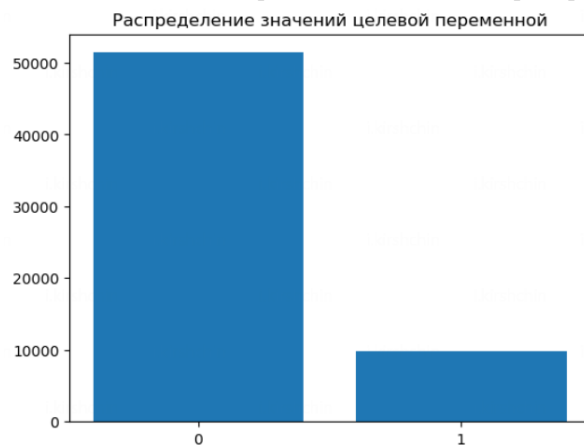
1. `issue_d` – месяц/год, в котором был выдан кредит
2. `purpose` – категория, указанная заемщиком для запроса кредита
3. `addr_state` – штат заемщика
4. `sub_grade` – внешне назначенная подкатегория кредита
5. `home_ownership` – статус владения жильем, указанный заемщиком при регистрации или полученный из кредитного отчета.
6. `emp_title` – должность, указанная заемщиком при подаче заявки на кредит
7. `installment` – ежемесячный платеж по кредиту
8. `dti` – коэффициент, рассчитываемый на основе общей ежемесячной суммы долговых платежей заемщика по отношению к его/ее заявленному ежемесячному доходу, исключая ипотеку и запрашиваемый кредит LC
9. `funded_amnt` – сумма кредита
10. `annual_inc` – заявленный годовой доход заемщика, указанный при регистрации
11. `emp_length` – продолжительность занятости в годах. Возможные значения от 0 до 10, где 0 означает менее года, а 10 - десять или более лет
12. `term` – количество платежей по кредиту: либо 36, либо 60
13. `inq_last_6mths` – количество запросов за последние 6 месяцев (исключая авто и ипотечные запросы)
14. `mths_since_recent_inq` – количество месяцев с момента последнего запроса
15. `delinq_2yrs` – количество инцидентов просрочки более чем на 30 дней в кредитном файле заемщика за последние 2 года
16. `chargeoff_within_12_mths` – количество списаний в течение 12 месяцев

17. num_accts_ever_120_pd – количество счетов, просроченных на 120 или более дней
18. num_tl_90g_dpd_24m – количество счетов, просроченных на 90 или более дней за последние 24 месяца
19. acc_open_past_24mths – количество открытых сделок за последние 24 месяца
20. avg_cur_bal – средний текущий баланс всех счетов
21. tot_hi_cred_lim – общий лимит по высококачественным кредитам
22. delinq_amnt – сумма просроченной задолженности по счетам, по которым заемщик сейчас просрочен

3. EDA

В качестве EDA я решил посмотреть на распределение значений целевой переменной в датасете, распределение значений целевой переменной по годам и по месяцам, default rate по годам и по месяцам. Также решил посмотреть на скоррелированные признаки и постарался объяснить причины их корреляции.

Значения целевой переменной в датасете распределились следующим образом:

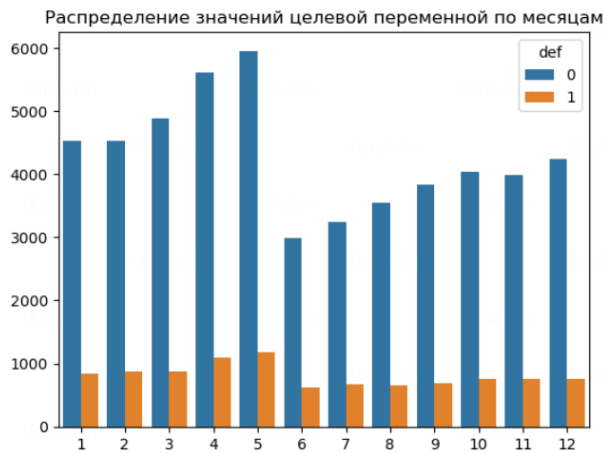


В датасете примеров клиентов, вернувших кредит, значительно больше чем примеров тех, кто кредит не вернул, что, конечно же, объяснимо. Однако дефолтнувшие клиенты, коих чуть меньше 10000, также представлены достаточно репрезентативно.

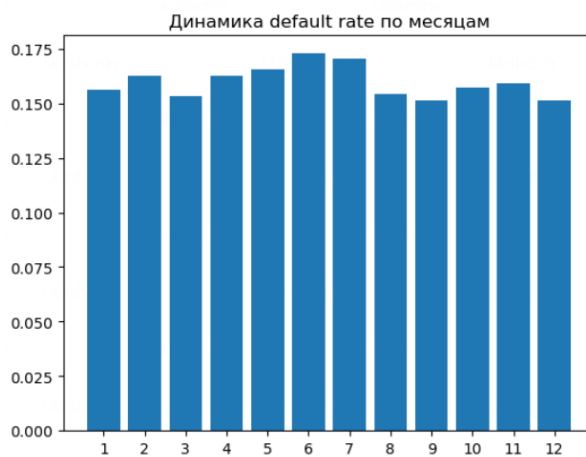
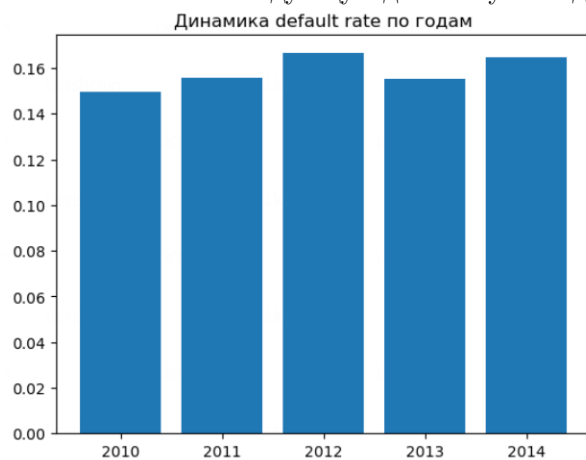
Значения целевой переменной по годам и по месяцам распределились следующим образом:



Пусть с течением времени число наблюдений для каждого года стремительно растет, каждый год представлен достаточно репрезентативно.



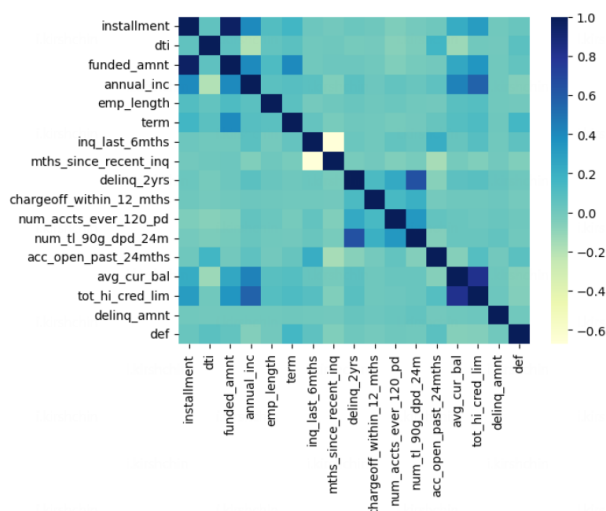
По месяцам наблюдения распределились скорее равномерно.
Default rate имеет следующую динамику по годам и месяцам:



От месяца к месяцу и от года к году default rate имеет примерно одно и то же значение.

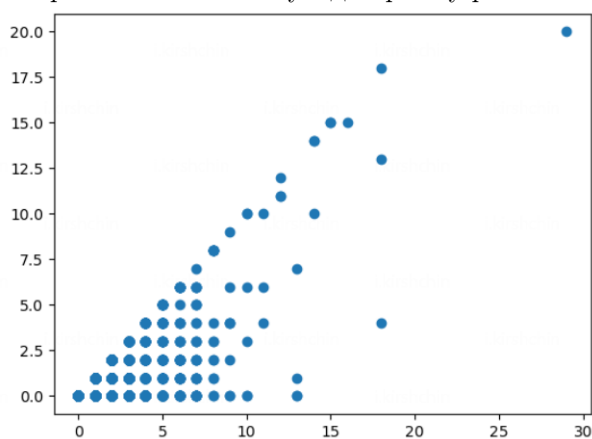
Так как чаще всего ставка кредита, которая входит в переменную `installment`, устанавливается исходя из `PD`, признак `installment` не должен быть известен на этапе скоринга, поэтому мы не будем использовать данный признак.

Посмотрим на корреляцию между признаками:



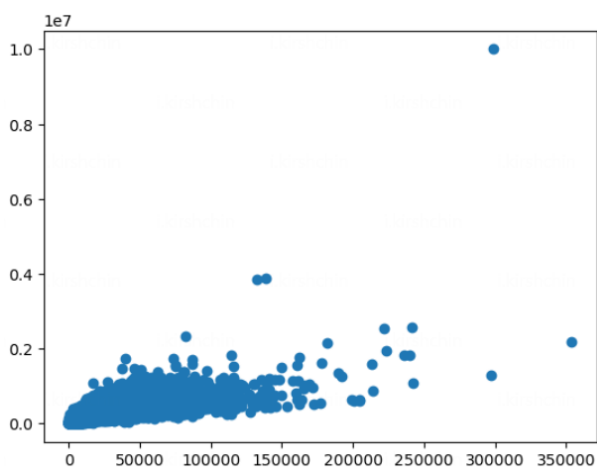
Сильно скоррелированными являются признаки `inq_last_6mths` и `mths_since_recent_inq`. Это объясняется тем, что `inq_last_6mths` – количество обращений за займом в последние 6 месяцев, а `mths_since_recent_inq` – количество месяцев с последнего обращения. Соответственно, при всех значениях `mths_since_recent_inq` > 6 значения признака `inq_last_6mths` будут равны 0 и наоборот.

Также сильно скоррелированными признаками являются `delinq_2yrs` и `num_tl_90g_dpd_24m`. Посмотрим на их совместную диаграмму рассеивания:



Данная зависимость объясняется тем, что `delinq_2yrs` – количество случаев просрочки платежа более чем на 30 дней в кредитной истории заемщика за последние 2 года и `num_tl_90g_dpd_24m` – количество счетов с просрочкой платежа на 90 или более дней за последние 24 месяца обозначают достаточно смежные вещи и для всех значений признака выполняется $\text{delinq_2yrs} \geq \text{num_tl_90g_dpd_24m}$.

Еще одна скоррелированная пара признаков - `avg_cur_bal` и `tot_hi_cred_lim`. Посмотрим на их совместную диаграмму рассеивания:



Наблюдаемая зависимость объясняется тем, что кредитный лимит для клиента устанавливается в том числе исходя из среднего текущего баланса на счетах клиента.

Ближе к концу, по итогам препроцессинга признаков, их WOE-преобразования и исключения неинформативных признаков, из каждой пары скореллированных признаков мы оставим более информативный в соответствии с IV.

4. Обработка признаков

Далее поочередно будем проходить по каждому из признаков, заполнять пропуски, дискретизировать его, применять WOE и отбрасывать в случае, когда признак оказывается неинформативным. Неинформативными будем считать те признаки, для которых по итогам WOE-преобразования получается $IV < 0.02$.

Создадим вспомогательный датафрейм «test».

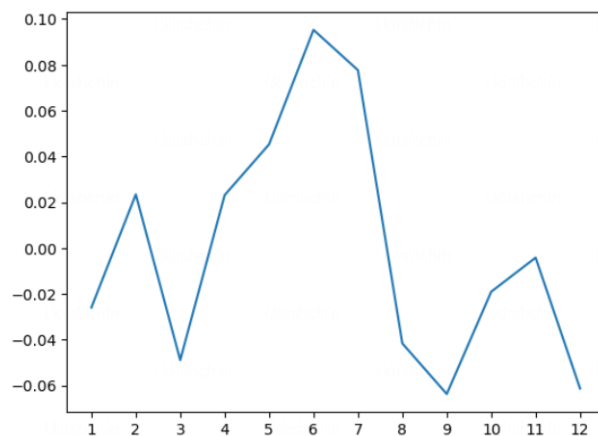
Для начала реализуем функцию для WOE-преобразования, принимающую в качестве аргументов датасет, название признака и названия таргета и возвращающую датасет с промежуточными вычислениями, датасет с посчитанным WOE для каждого значения признака и итоговый IV.

```
In [12]: def WOE(data, feature, target):
    woe_df = pd.DataFrame()
    s1 = data[target].sum()
    s0 = data[target].count() - s1
    woe_df['DistributionGood'] = data.groupby(feature).agg({'target': 'sum'}) / s1
    woe_df['DistributionBad'] = (data.groupby(feature).agg({'target': 'count'}) - data.groupby(feature).agg({'target': 'sum'})) / s0
    woe_df = woe_df[woe_df['DistributionBad'] != 0]
    woe_df = woe_df[woe_df['DistributionGood'] != 0]
    woe_df['WOE'] = np.log(woe_df['DistributionGood'] / woe_df['DistributionBad'])
    woe_df['IV'] = (woe_df['DistributionGood'] - woe_df['DistributionBad']) * woe_df['WOE']
    return woe_df, woe_df['WOE'], woe_df['IV'].sum()
```

- `issue_d`

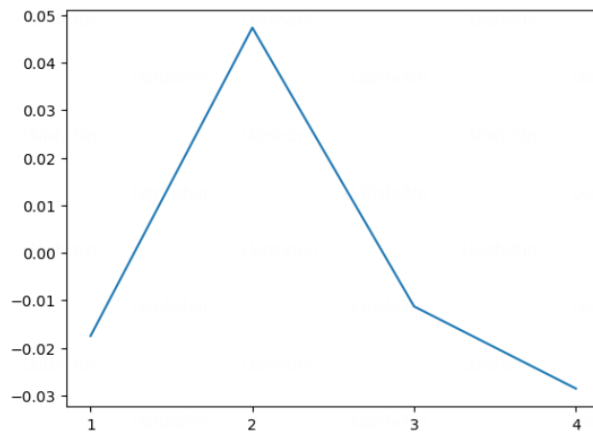
Пропущенные значения отсутствуют.

Оставим только месяц выдачи и применим к нему WOE-преобразование, после чего построим график зависимости WOE от значения месяца.



Никакой монотонной зависимости не наблюдается.

Попробуем сгруппировать по кварталам и повторить те же действия.



Никакой монотонности так и не появилось, так что не берем признак для дальнейшей классификации.

- **purpose**

Пропущенные значения отсутствуют.

Посмотрим на количество элементов для каждого значения признака.

```
debt_consolidation    35587
credit_card            13722
home_improvement       3549
other                  3111
major_purchase         1378
small_business         969
car                    758
medical                587
moving                 400
wedding                390
house                  356
vacation               306
renewable_energy       43
educational             13
Name: purpose, dtype: int64
```

Отнесем редко встречающиеся категории «renewable_energy» и «educational» к «other», после чего применим WOE-преобразование и посмотрим на получившийся IV.

Получившийся $IV = 0.029$, в связи с чем мы берем признак для использования в классификации.

- **addr_state**

Пропущенные значения отсутствуют.

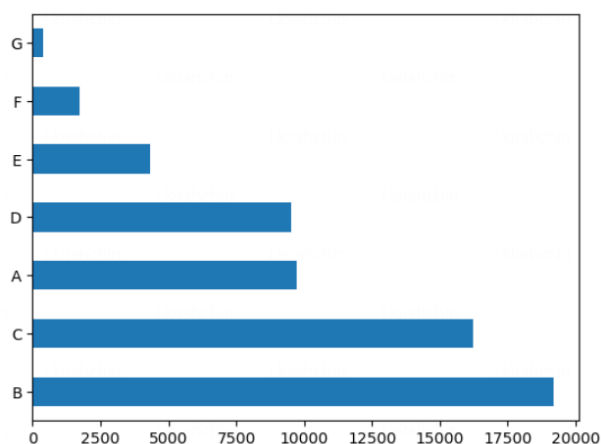
Применим WOE-преобразование для штата клиента. Получившийся $IV = 0.178$. Это меньше чем 0.2, поэтому попробуем преобразовать признак. Сгруппируем штаты по субрегионам, после чего вновь применим WOE-преобразование и посмотрим на получившийся IV.

В итоге имеем $IV = 0.0065$, что еще меньше порогового значения, поэтому не берем признак в дальнейшую классификацию.

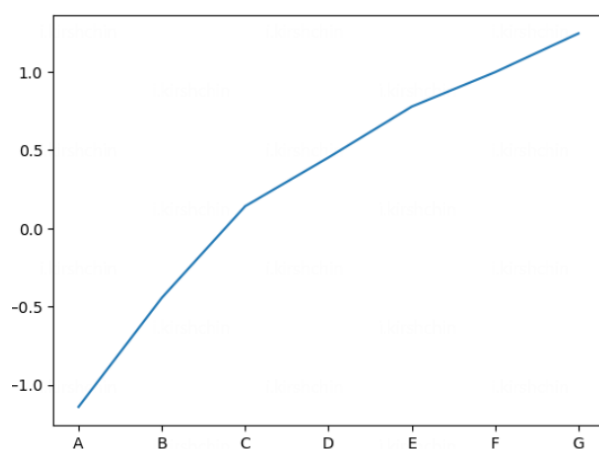
- **sub_grade**

Пропущенные значения отсутствуют.

Признак имеет большое количество различных значений, для многих из которых в датасете находится довольно небольшое число наблюдений. Объединим внешне назначенную подкатегорию кредита по первой букве подкатегории, после чего получим множество значений признака A, B, C, D, E, F, G, каждое из которых теперь представлено достаточным количеством наблюдений.



Применим WOE-преобразование и построим график зависимости IV от подкатегории (здесь значения признака вполне можно упорядочить).



Зависимость получается монотонной и $IV = 0.34$, поэтому берем этот признак для дальнейшей классификации.

- **home_ownership**

MORTGAGE	31414
RENT	24581
OWN	5158
OTHER	10
NONE	6

Пропущенные значения присутствуют, отнесем значение «NONE» к значению «OTHER».

Применим WOE.

Получившийся $IV = 0.013$, что меньше порогового значения, поэтому не берем признак для дальнейшей классификации.

- **emp_title**

```

NaN                3865
Teacher            505
Manager            420
Registered Nurse   239
RN                 237
...
Chesapeake energy    1
Commercial Banking Officer  1
US Navy (Civil Service)  1
Gemini Sales & Services  1
receptionoist        1
Name: emp_title, Length: 38577, dtype: int64

```

Пропущенные значения присутствуют, будем считать их отдельным значением признака.

Признак имеет 38+ тысяч разных значений, поэтому попробуем выделить конкретные названия профессий из данных значений при помощи суффиксов, характерных для названий профессий в английском языке.

```

In [59]: def profession(desc, top_prof):
          words = str(desc).split()
          for w in words:
              if w.lower()[2:] in ('er', 'or', 'ee') or w.lower()[3:] in ('ant', 'ian', 'ist', 'ent') or w.lower() in top_prof:
                  return w.lower()
          if str(desc).lower() in top_prof:
              return str(desc).lower()
          return 'other'

```

```

In [60]: top_prof = list(data['emp_title'].map(lambda x: str(x).lower()).value_counts(dropna=False)[:50].index)
          test['emp_title'] = data['emp_title'].map(lambda x: profession(x, top_prof))
          test['emp_title'].value_counts()

```

```

Out[60]: other          31231
         nan            3867
         manager        3237
         director        908
         sales           878
         ...
         larimer         1
         mechinist       1
         tenant          1
         hollyfrontier    1
         receptionoist    1
Name: emp_title, Length: 1952, dtype: int64

```

При помощи данной обработки удалось сократить количество различных значений в 20 раз. Осталось убрать особенно редкие категории.

```

In [61]: top_prof = list(test['emp_title'].value_counts(dropna=False)[:60].index)
          test['emp_title'] = test['emp_title'].map(lambda x: x if x in top_prof else 'other')
          test['emp_title'].value_counts()[:20]

```

```

Out[61]: other          38536
         nan            3867
         manager        3237
         director        908
         sales           878
         center          843
         assistant       822
         teacher         718
         senior          715
         supervisor      652
         nurse           600
         analyst         587
         engineer        586
         specialist      570
         officer         544
         driver          536
         department      510
         technician      460
         rn              323
         coordinator     320
Name: emp_title, dtype: int64

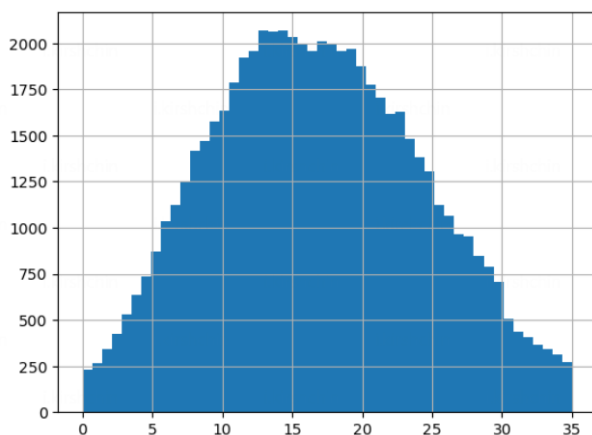
```

Применяем WOE-преобразование к получившимся категориям и получаем $IV = 0.021$, что позволяет нам взять признак для дальнейшей классификации.

• dti

Пропущенные значения отсутствуют.

Посмотрим на гистограмму распределения признака:



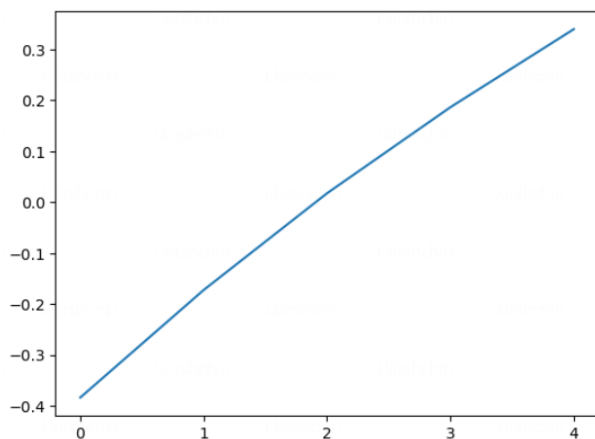
Настало время заняться дискретизацией числовых признаков. Напишем алгоритм, группирующий наблюдения таким образом, чтобы в каждый из бакетов попадал хотя бы 1% наблюдений, бакетов было минимум 4 и максимум 22 и $IV \geq 0.02$.

Функция `bucketing(data, feature, min_size, max_size, step)` работает следующим образом:

1. Перебираем размеры диапазона разбиения от `min_size` до `max_size` с шагом `step`
2. Делим значения признака нацело на размер диапазона разбиения и формируем бакеты
3. Идем по бакетам от меньших значений к БОльшим и, в случае, если в бакете содержится менее 1% всех наблюдений, присоединяем к нему все бакеты с БОльшими значениями признака, пока в получившемся бакете не окажется не менее 1% всех наблюдений, после чего переключаемся на следующий за ним из оставшихся.
4. После того, как мы дошли до последнего бакета, разворачиваемся и проделываем те же действия в обратном направлении
5. Если получившееся разбиение удовлетворяет всем условиям (хотя бы 1% наблюдений в каждом бакете, бакетов минимум 4 и максимум 22 и $IV \geq 0.02$), сохраняем получившееся разбиение как одно из возможных.

После нахождения всех возможных разбиений, для каждого из них строим график зависимости значений WOE от дискретизованных значений признака и выбираем бакет, для которого данная зависимость получается наиболее монотонной. Выбранный бакет и будет оптимальным разбиением.

Вернемся к «dti». Способом разбиения, дающим самую большую монотонную зависимость, оказался 0-7, 7-14, 14-21, 21-28, 28-34.99.

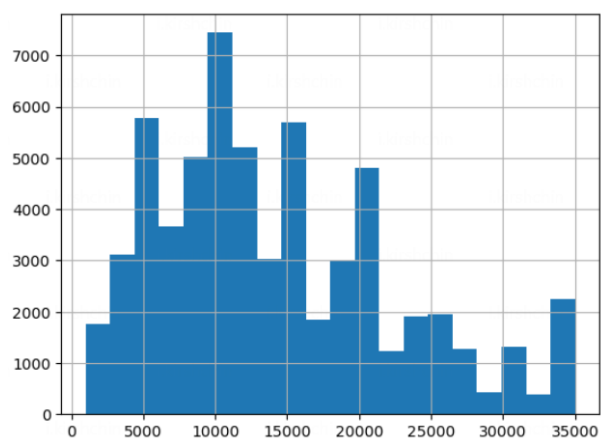


По самому условию отбора, получившийся признак обладает $IV \geq 0.02$, поэтому мы оставляем преобразованный признак для использования в классификации.

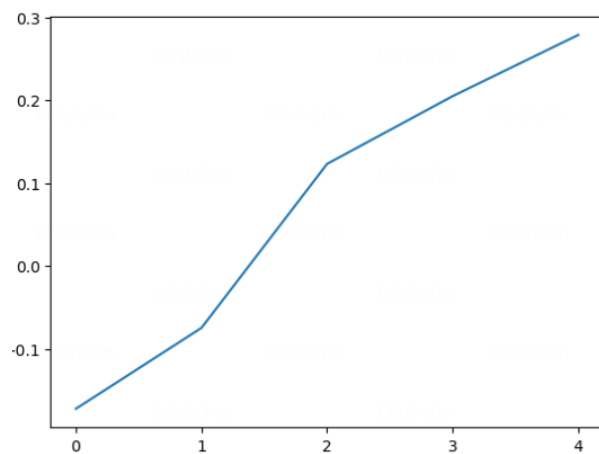
- **funded_amnt**

Пропущенные значения отсутствуют.

Посмотрим на гистограмму распределения признака:



Попробуем подобрать способ разбиения. Самым оптимальным разбиением оказалось 0-7600, 7600-15200, 15200-22800, 22800-30400, 30400-35000.



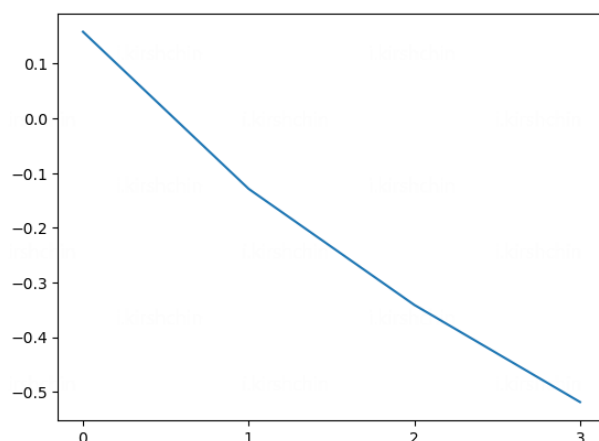
- **annual_inc**

Пропущенные значения отсутствуют.

Посмотрим на характеристики распределения признака:

```
count    6.116900e+04
mean     7.271755e+04
std      4.954698e+04
min      6.695000e+03
25%      4.500000e+04
50%      6.250000e+04
75%      8.800000e+04
max      3.900000e+06
```

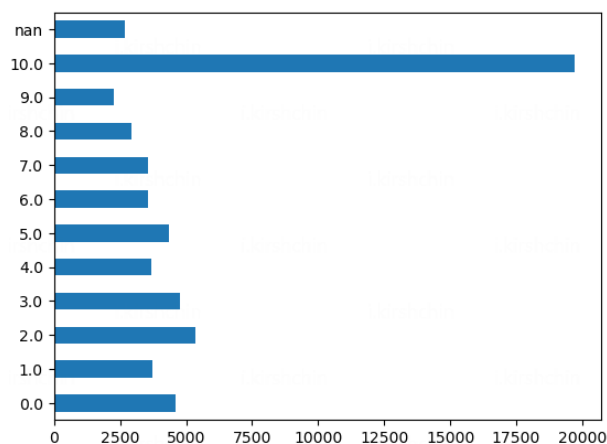
Попробуем подобрать способ разбиения. Самым оптимальным разбиением оказалось 0-62000, 62000-124000, 124000-186000, 186000-3900000.



- **emp_length**

Пропущенные значения присутствуют. Поочередно закодируем пропуски нулем и медианой, для каждого варианта попробуем подобрать оптимальное разбиение, и затем выберем самый оптимальный вариант из всех.

Посмотрим на распределение признака:



Ни для варианта с нулем, ни для варианта с медианой подходящих разбиений не нашлось в принципе, поэтому не берем признак в дальнейшую классификацию.

- **term**

Пусть изначально, в соответствии с IV, признак и оказался сильным, в дальнейшем валидация показала, что именно на данном признаке модель сильно переобучалась, в связи с чем мы не будем использовать данный признак в классификации.

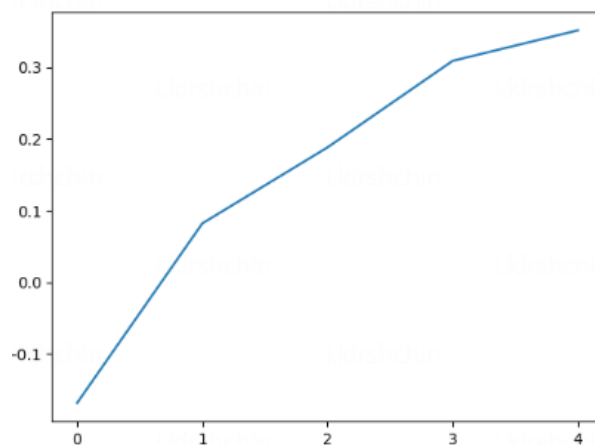
- **mths_since_recent_inq**

Пропущенные значения присутствуют. Поочередно закодируем пропуски нулем и медианой, для каждого варианта попробуем подобрать оптимальное разбиение, и затем выберем самый оптимальный вариант из всех.

Посмотрим на распределение признака:

NaN	13529
1.0	5106
0.0	4248
2.0	4023
3.0	3607
4.0	3430
5.0	3064
7.0	2860
6.0	2841
8.0	2574
9.0	2180
10.0	1959
11.0	1662
12.0	1542
13.0	1340
14.0	1206
15.0	986
16.0	869
17.0	751
18.0	671
19.0	634
20.0	558
21.0	474
22.0	426
23.0	416
24.0	213

Попробуем подобрать способ разбиения. Разбиения подобрались только для варианта с медианой и самым оптимальным из них оказалось 0-5, 5-10, 10-15, 15-20, 20-24.



- **delinq_2yrs**

Пропущенные значения отсутствуют.

Посмотрим на распределение признака:

0	50997
1	6987
2	1981
3	657
4	259
5	140
6	60
7	33
8	16
9	11
10	7
11	4
13	4
18	3
14	3
12	3
15	2
16	1
29	1

Подходящих способов разбиения не нашлось

- **chargeoff_within_12_mths**

Пропущенные значения отсутствуют.

Посмотрим на распределение признака:

0	0.994294
1	0.005019
2	0.000621
3	0.000065

Так как в 99.5% случаев признак принимает одно и то же значение, будет иметь смысл его исключить.

- **num_accts_ever_120_pd**

Пропущенные значения присутствуют. Поочередно закодируем пропуски нулем и медианой, для каждого варианта попробуем подобрать оптимальное разбиение, и затем выберем самый оптимальный вариант из всех.

Посмотрим на распределение признака:

0.0	39427
NaN	11941
1.0	5561
2.0	2099
3.0	898
4.0	518
5.0	291
6.0	197
7.0	99
8.0	45
9.0	37
10.0	24
11.0	14
12.0	10
14.0	4
13.0	1
20.0	1
15.0	1
29.0	1

Ни для варианта с нулем, ни для варианта с медианой подходящих разбиений не нашлось, поэтому не берем признак в дальнейшую классификацию.

- **num_tl_90g_dpd_24m**

Пропущенные значения присутствуют. Поочередно закодируем пропуски нулем и медианой, для каждого варианта попробуем подобрать оптимальное разбиение, и затем выберем самый оптимальный вариант из всех.

Посмотрим на распределение признака:

0.0	46585
NaN	11941
1.0	2015
2.0	434
3.0	87
4.0	52
5.0	16
6.0	15
8.0	6
10.0	4
15.0	3
14.0	2
7.0	2
11.0	2
13.0	1
9.0	1
18.0	1
12.0	1
20.0	1

Ни для варианта с нулем, ни для варианта с медианой подходящих разбиений не нашлось, поэтому не берем признак в дальнейшую классификацию.

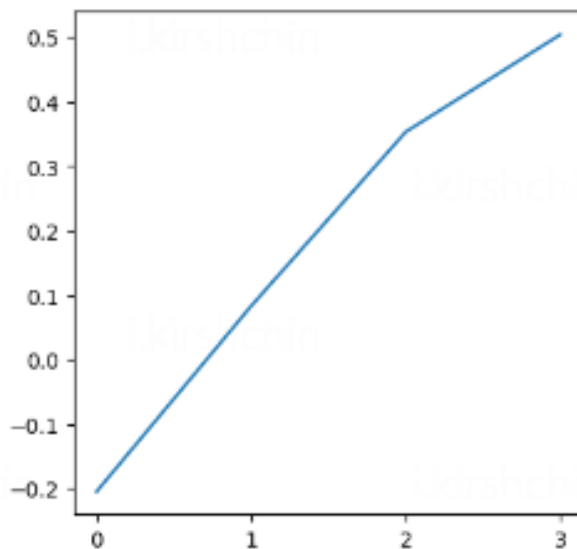
- **acc_open_past_24mths**

Пропущенные значения присутствуют. Поочередно закодируем пропуски нулем и медианой, для каждого варианта попробуем подобрать оптимальное разбиение, и затем выберем самый оптимальный вариант из всех.

Посмотрим на распределение признака:

3.0	8829
2.0	8133
4.0	7990
NaN	7886
5.0	6470
1.0	5857
6.0	4718
7.0	3269
0.0	2689
8.0	2104
9.0	1299
10.0	751
11.0	455
12.0	273
13.0	172
14.0	115
15.0	56
16.0	39
17.0	25
18.0	14
19.0	11
20.0	6
21.0	4
23.0	1
33.0	1
28.0	1
22.0	1

Попробуем подобрать способы разбиения. Разбиения подошлись и для варианта с медианой и для заполнения нулем, и оптимальным из них оказалось разбиение для варианта с нулем по бакегам 0-4, 4-8, 8-12, 12-33.



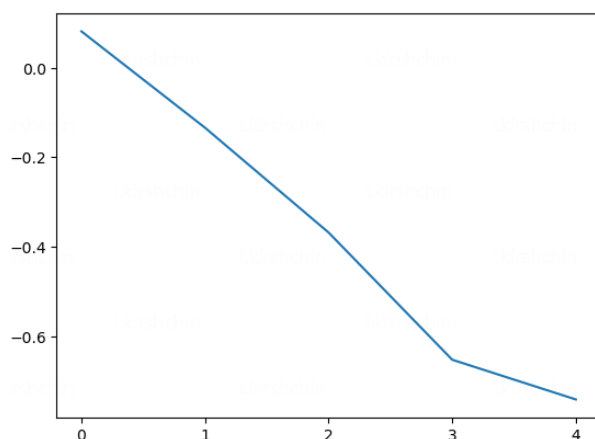
- `avg_cur_bal`

Пропущенные значения присутствуют. Поочередно закодируем пропуски нулем и медианой, для каждого варианта попробуем подобрать оптимальное разбиение, и затем выберем самый оптимальный вариант из всех.

Посмотрим на характеристики распределения признака:

count	49224.000000
mean	13734.641049
std	16054.221586
min	0.000000
25%	3025.000000
50%	7847.000000
75%	19527.000000
max	354015.000000

Попробуем подобрать способы разбиения. Разбиения подошлись и для варианта с медианой и для заполнения нулем, и оптимальным из них оказалось разбиение для варианта с нулем по бакегам 0-15000, 15000-30000, 30000-45000, 45000-60000, 60000-354015.



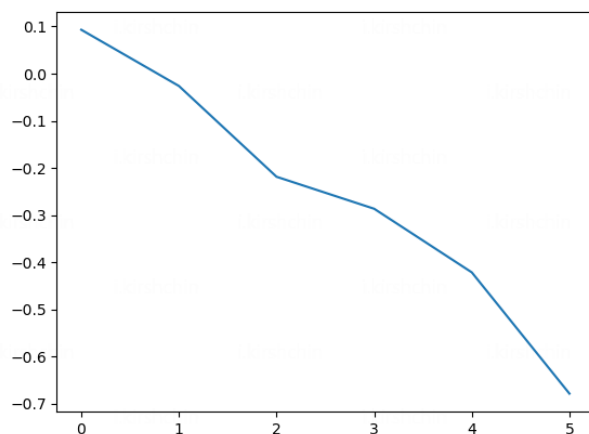
- **tot_hi_cred_lim**

Пропущенные значения присутствуют. Поочередно закодируем пропуски нулем и медианой, для каждого варианта попробуем подобрать оптимальное разбиение, и затем выберем самый оптимальный вариант из всех.

Посмотрим на характеристики распределения признака:

```
count    4.922800e+04
mean     1.671557e+05
std      1.699201e+05
min       0.000000e+00
25%      4.550000e+04
50%      1.110695e+05
75%      2.458785e+05
max       9.999999e+06
```

Попробуем подобрать способы разбиения. Разбиения подобрались и для варианта с медианой и для заполнения нулем, и оптимальным из них оказалось разбиение для варианта с медианой по бакедам 0-120000, 120000-240000, 240000-360000, 360000-480000, 480000-600000, 600000-999999.



- **delinq_amnt**

Пропущенные значения отсутствуют.

Посмотрим на распределение признака:

```

0      0.997924
25     0.000098
30     0.000098
65000  0.000049
254    0.000033
...
1420   0.000016
1921   0.000016
56     0.000016
94     0.000016
10     0.000016

```

Так как в 99.5% случаев признак принимает одно и то же значение, будет иметь смысл его исключить.

5. Feature engineering

Создадим признак `funded_amnt / tot_hi_cred_lim`, отражающий, насколько много денег банк выдал клиенту относительно его лимита по высококачественным кредитам.

Заполним пропуски в `tot_hi_cred_lim` средним значением по выборке.

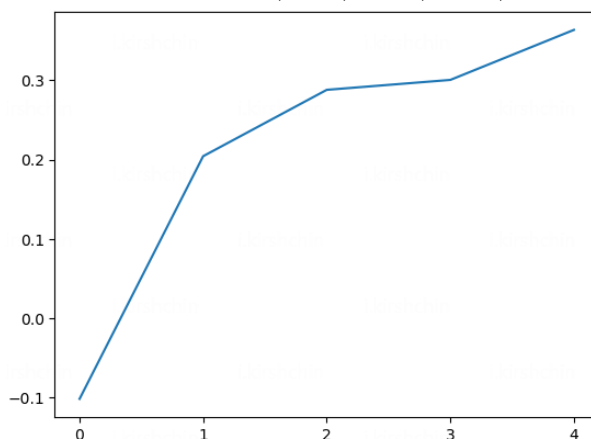
Посмотрим на характеристики распределения нового признака.

```

count    61169.000000
mean      17.450394
std       25.112705
min       0.201284
25%       5.248289
50%      9.622302
75%      21.342895
max      2250.000000

```

Попробуем подобрать оптимальное разбиение по бакетам для признака. Самым оптимальным разбиением оказалось 0-4, 4-20, 20-40, 40-60, 60-80.



Так как новый признак теперь показывает, насколько много банк дал клиенту относительно того, сколько мог дать, мы можем исключить `funded_amnt`, показывающий в целом ту же информацию, но в менее информативных абсолютных значениях.

6. Финальное преобразование

Вспомним скоррелированные пары признаков из начала.

В паре `tot_hi_cred_lim, avg_cur_bal` менее информативным оказался `tot_hi_cred_lim`, а в паре `inq_last_6mths, mths_since_recent_inq` — `mths_since_recent_inq`, данные признаки мы исключаем.

Также стандартизуем все столбцы получившегося датасета для более корректного использования регуляризации.

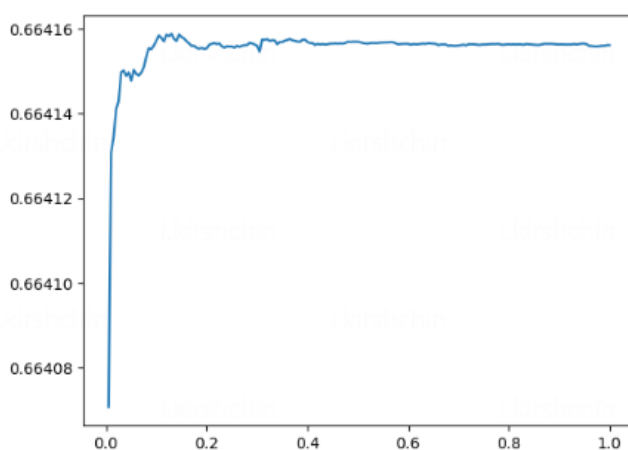
7. Подбор гиперпараметров и обучение модели

Для подбора обратного коэффициента регуляризации разобьем тренировочную выборку на новую тренировочную и валидационную. После этого при помощи перебора значений обратного коэффициента регуляризации по сетке от 0.005 до 1 с шагом 0.005 найдем оптимальный исходя из качества модели, обученной на новой тренировочной выборке, на валидационной выборке.

```
In [381]: X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, test_size=0.2, random_state=3)
val_scores = []
max_score = -1
coef = -1
for i in tqdm(np.arange(0.005, 1.005, 0.005)):
    mod = LogisticRegression(penalty='l2', C=i)
    mod.fit(X_train1, y_train1)
    score = roc_auc_score(y_val1, mod.predict_proba(X_val1)[:, 1])
    val_scores.append(score)
    if score > max_score:
        max_score = score
        coef = i
plt.plot(np.arange(0.005, 1.005, 0.005), val_scores);
print(f'Максимальный ROC-AUC на валидационной выборке: {max_score}')
print(f'Оптимальный коэффициент регуляризации C: {coef}')
```

100% 200/200 [00:08<00:00, 28.21it/s]

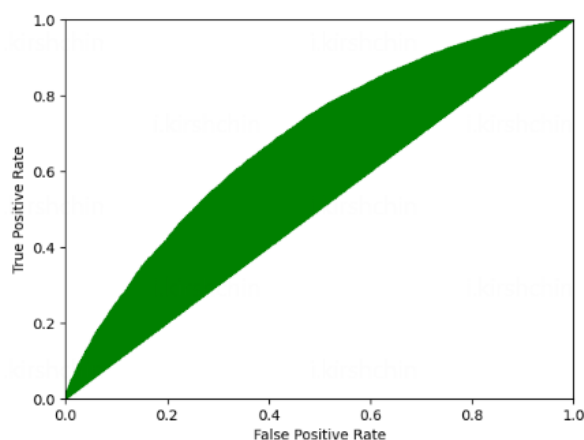
Максимальный ROC-AUC на валидационной выборке: 0.664158842076152
Оптимальный коэффициент регуляризации C: 0.13



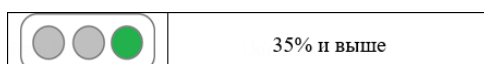
Оптимальным значением оказалось $C = 0.13$. Обучим модель с данным гиперпараметром и перейдем к валидации.

8. Валидация

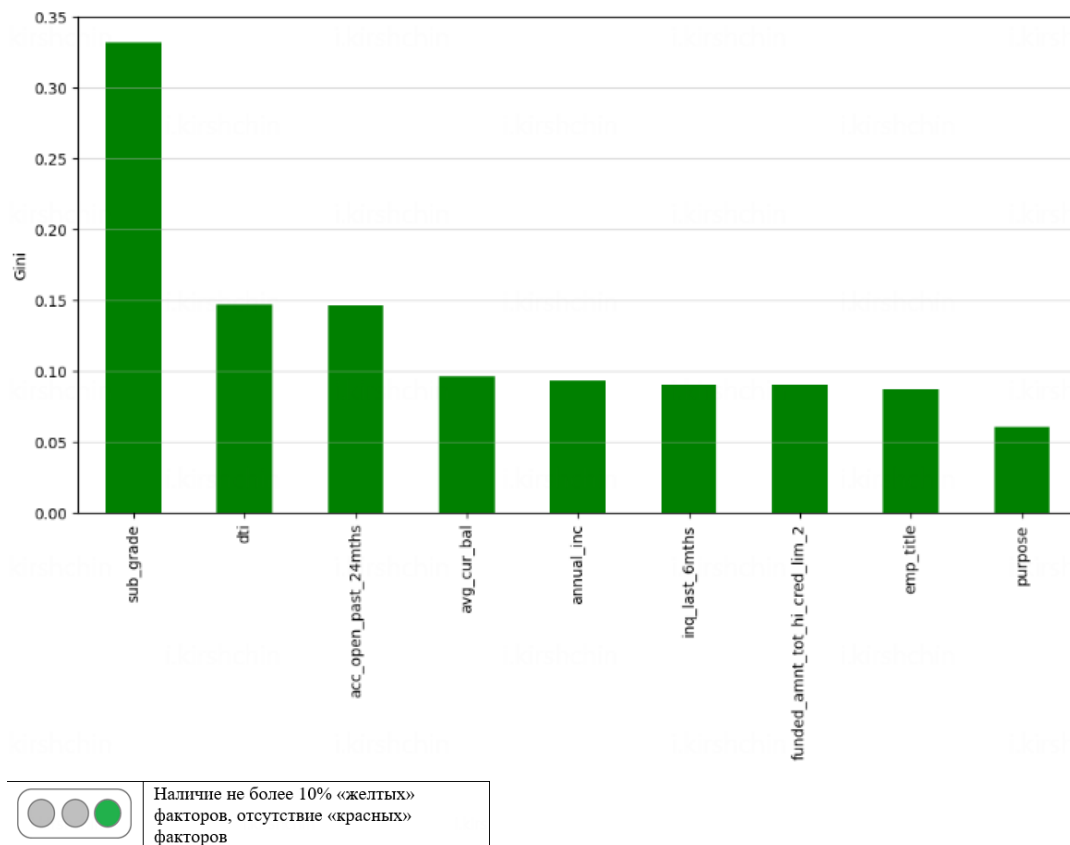
- Тест M2.1: Эффективность ранжирования всей модели



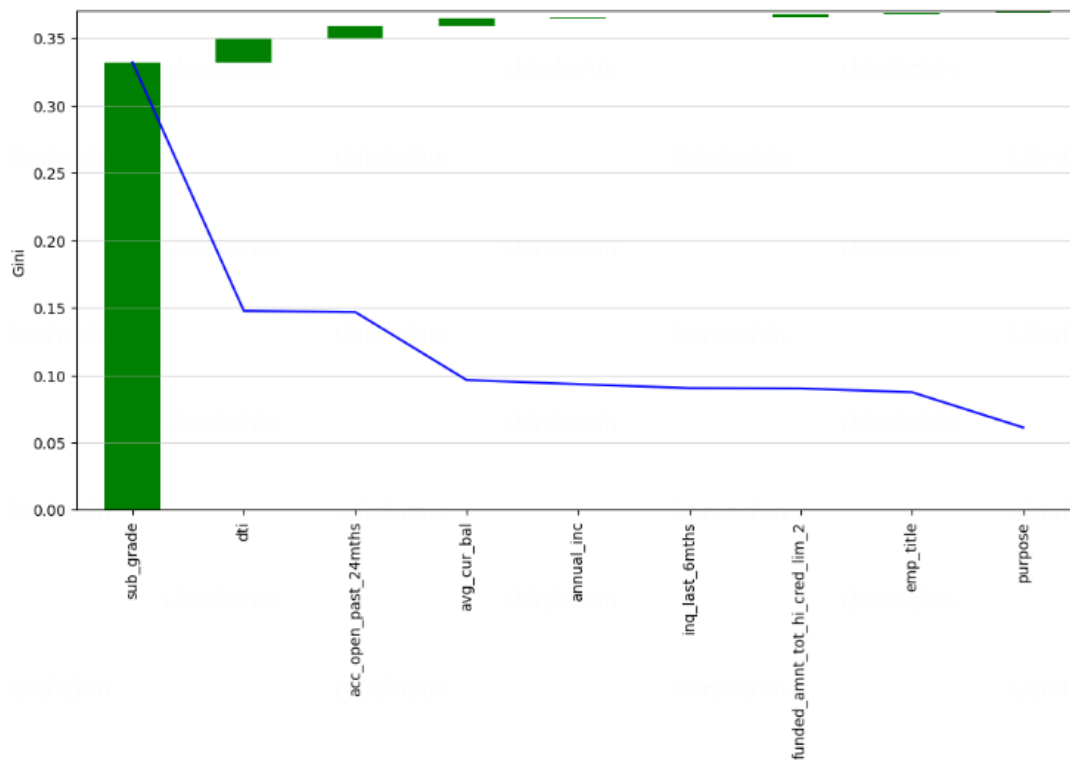
Gini = 36,82%



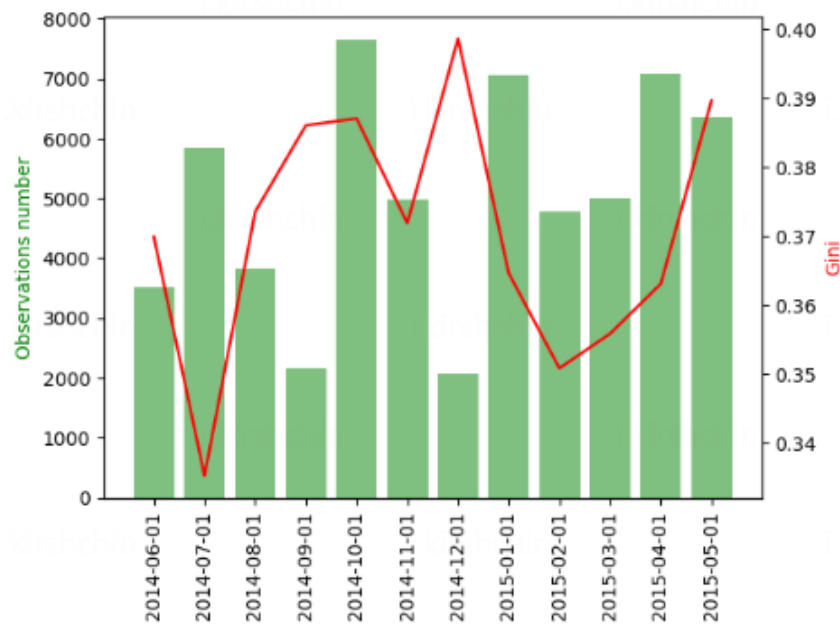
- Тест М2.2: Эффективность ранжирования отдельных факторов



- Тест М2.5: Анализ вкладов факторов в формирование Джини модели



- Тест М2.4: Динамика коэффициента Джини



- Тест М4.1: Сравнение прогнозного и фактического TR (Target Rate) на уровне выборки

```
print(f'Среднее арифметическое значение вероятности наступления целевого события: {lr.predict_proba(X_val)[: , 1].mean()}')
```

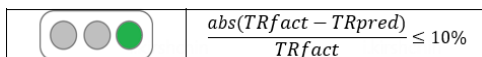
Среднее арифметическое значение вероятности наступления целевого события: 0.1614381049493969

```
print(f'Средний уровень целевого события: {y_val.mean()}')
```

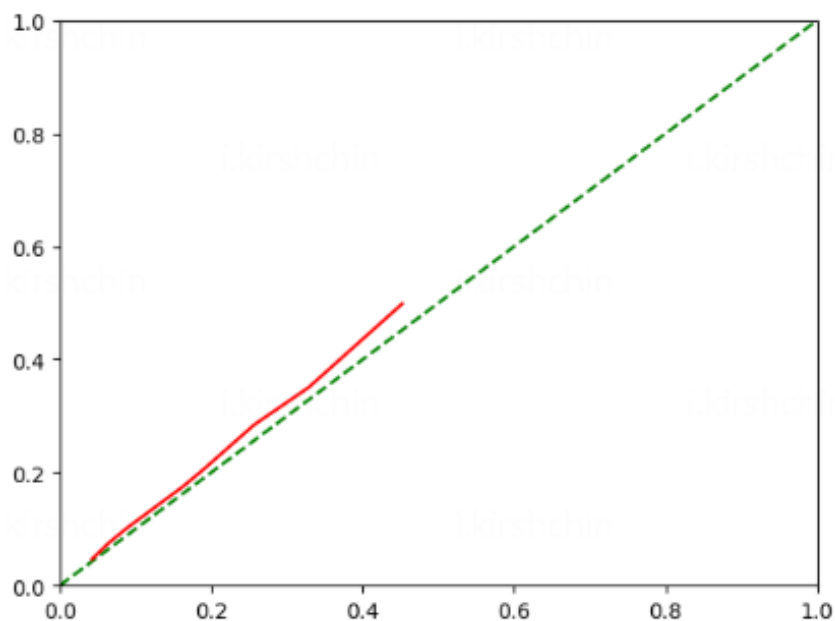
Средний уровень целевого события: 0.1767357830706247

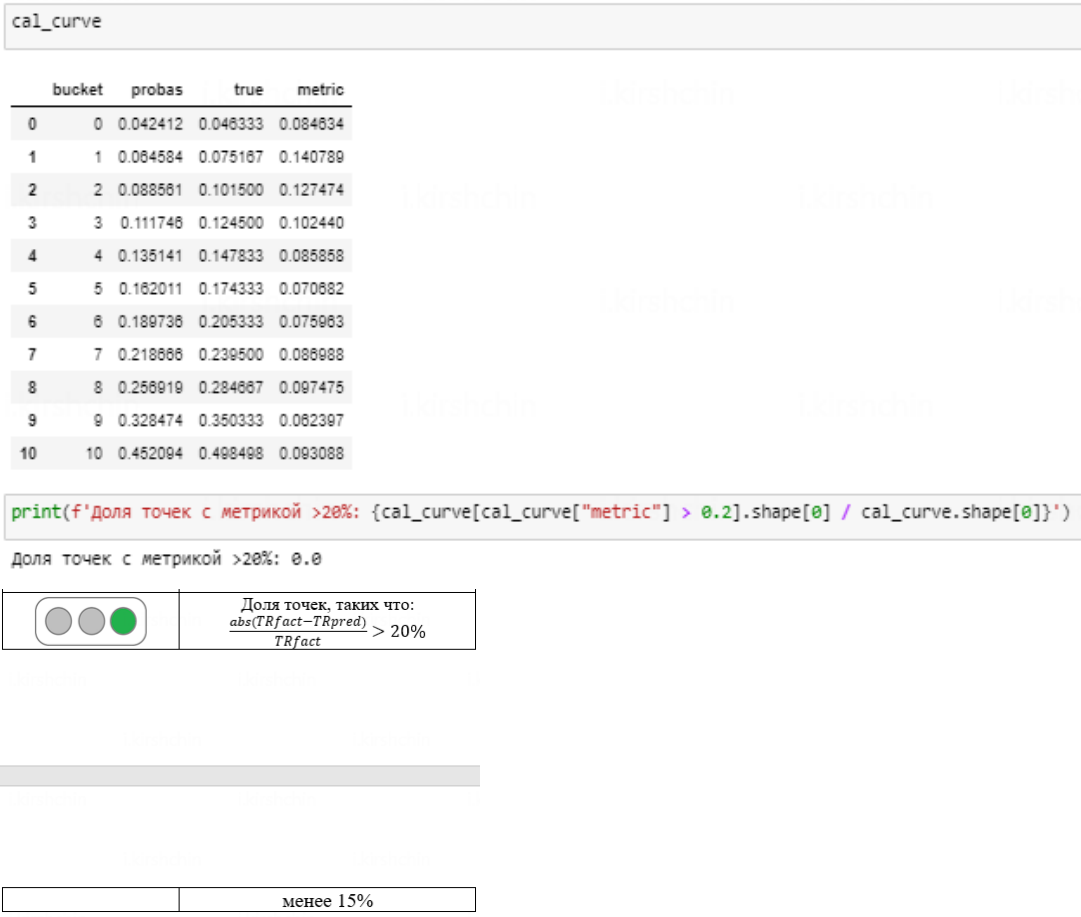
```
abs(y_val.mean() - lr.predict_proba(X_val)[: , 1].mean()) / y_val.mean()
```

0.08655676770965373




- Тест М4.2: Тест формы калибровочной кривой



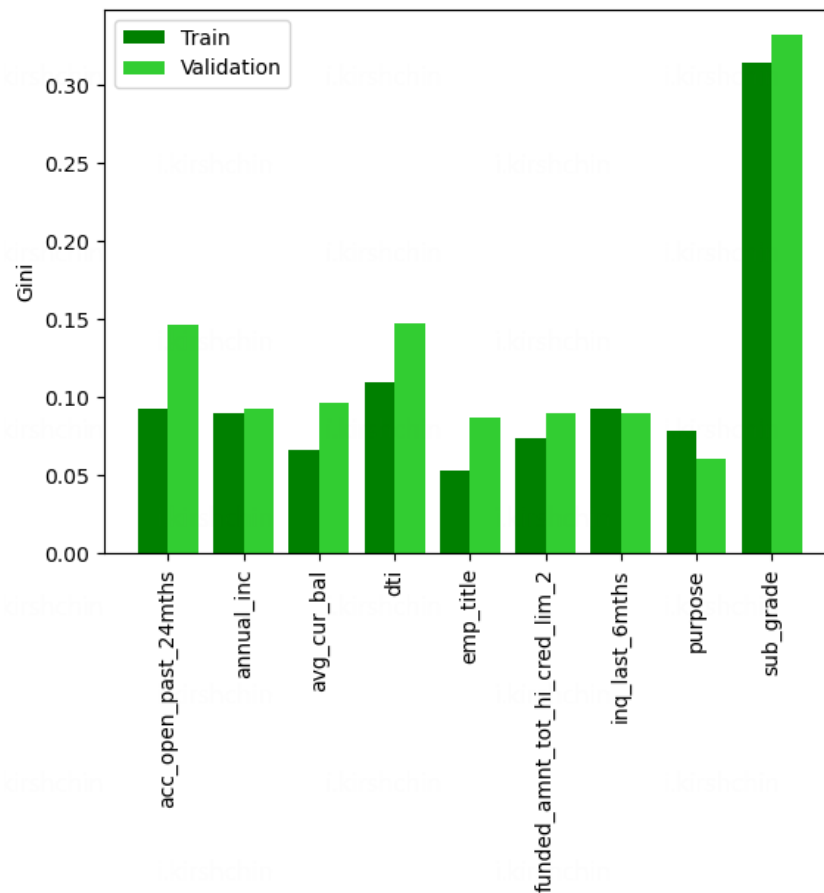


Тест М5.1: Сравнение эффективности ранжирования модели на разработке и валидации




	Абсолютное снижение менее 3 п. п. ЦЛП относительное снижение менее 15%
---	--

Тест М5.2: Сравнение эффективности ранжирования отдельных факторов модели на разработке и валидации

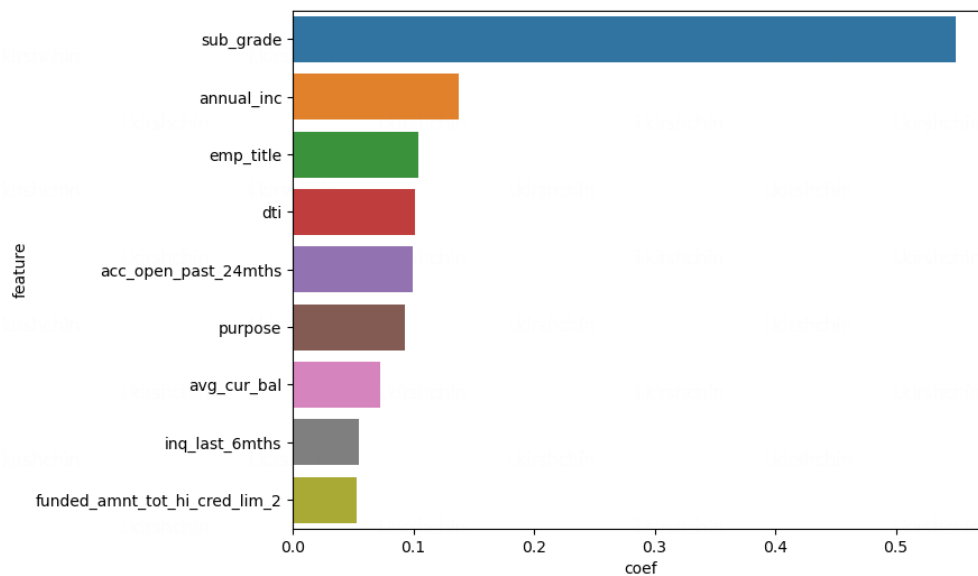


	gini_train	gini_val	abs_change	rel_change	feature_flg
acc_open_past_24mths	0.092746	0.146763	0.054017	0.582416	green
annual_inc	0.090019	0.093239	0.003220	0.035772	green
avg_cur_bal	0.066240	0.096341	0.030101	0.454420	green
dti	0.109655	0.147593	0.037938	0.345981	green
emp_title	0.053139	0.087293	0.034154	0.642733	green
funded_amnt_tot_hi_cred_lim_2	0.073670	0.090090	0.016420	0.222891	green
inq_last_6mths	0.092526	0.090339	-0.002187	-0.023638	green
purpose	0.078703	0.061069	-0.017634	-0.224061	green
sub_grade	0.314731	0.332032	0.017300	0.054968	green

	Наличие не более 10% «желтых» факторов
---	--

9. Скоринговая карта

В соответствии с весами в модели линейной регрессии, по своему вкладу в решение о выдаче кредита признаки упорядочены следующим образом:



10. Расчет ожидаемой прибыли от кредитования

Прибыль для каждого кредита рассчитывается по формуле $\text{installment} * \text{term} - \text{funded_amnt}$.

Если банк решает выдать кредит и клиент возвращает кредит, банк получает прибыль в размере $\text{installment} * \text{term} - \text{funded_amnt}$, если клиент не возвращает – убытки в размере funded_amnt .

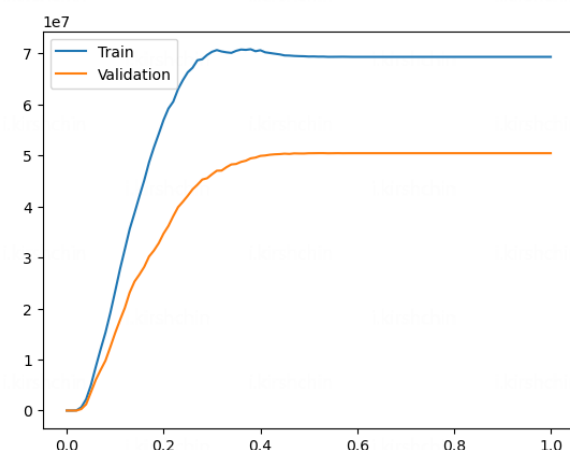
Если банк решает не выдавать кредит, его прибыль равна 0.

Таким образом, ожидаемая прибыль банка от кредитования – сумма по всем клиентам, которым решили выдать кредит, выражения $(1 - \text{вер-ть дефолта}) * (\text{installment} * \text{term} - \text{funded_amnt}) - \text{вер-ть дефолта} * \text{funded_amnt}$.

Задача заключается в нахождении оптимального правила для решения о выдаче кредита с целью максимизации ожидаемой прибыли от кредитования. Оптимальным правилом является порог предсказанной вероятности дефолта, выше которого банк отказывается выдавать кредит клиенту.

- Случай 1: в случае дефолта теряем весь кредит

```
plt.plot(np.arange(0, 1.01, 0.01), profit_train, label='Train')
plt.plot(np.arange(0, 1.01, 0.01), profit_val, label='Validation')
plt.legend();
```



```
print(f'Оптимальный порог отсеечения: {np.array(profit_train).argmax() * 0.01}')
```

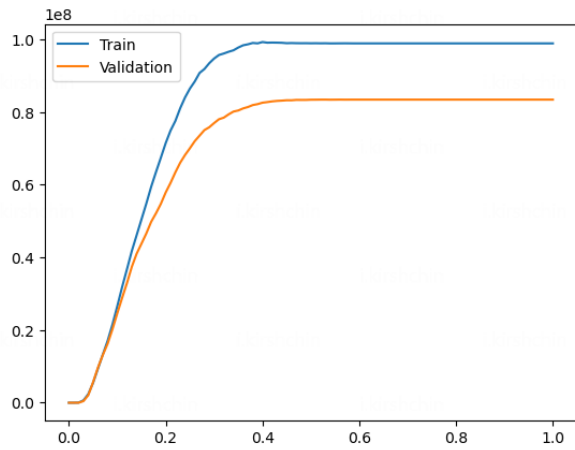
Оптимальный порог отсеечения: 0.38

```
print(f'Прибыль на валидационной выборке для оптимального порога отсеечения: {profit_val[np.array(profit_train).argmax()]})')
```

Прибыль на валидационной выборке для оптимального порога отсеечения: 49425537.16

- Случай 2: в случае дефолта теряем 80% кредита

```
plt.plot(np.arange(0, 1.01, 0.01), profit_train, label='Train')
plt.plot(np.arange(0, 1.01, 0.01), profit_val, label='Validation')
plt.legend();
```



```
print(f'Оптимальный порог отсеечения: {np.array(profit_train).argmax() * 0.01}')
```

Оптимальный порог отсеечения: 0.4

```
print(f'Прибыль на валидационной выборке для оптимального порога отсеечения: {profit_val[np.array(profit_train).argmax()]}')
```

Прибыль на валидационной выборке для оптимального порога отсеечения: 82553060.76000002