|  |
| --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное автономное образовательное учреждение  высшего образования |
| **Национальный исследовательский ядерный университет «МИФИ»** |



*ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ*

*КИБЕРНЕТИЧЕСКИХ СИСТЕМ*

**Кафедра №75 «Финансовый мониторинг»**

ОТЧЕТ

по курсу «Математические методы в задачах финансового мониторинга»

*Выполнили:*

студенты группы С15-501

Жосан А.В.

Забелина О.А.

Замятина М.Д.

Проверила: Домашова Д.В.

Москва 2019

Оглавление

[Постановка задачи 3](#_Toc27910425)

[Ход работы 4](#_Toc27910426)

[1. Первичный анализ данных 5](#_Toc27910427)

[1.1. Определение размера выборки 5](#_Toc27910428)

[1.2. Определение типов переменных 6](#_Toc27910429)

[1.3. Определение категориальных переменных 6](#_Toc27910430)

[1.4. Анализ целевой переменной 7](#_Toc27910431)

[2. Предобработка данных 7](#_Toc27910432)

[2.1. Удаление пустых значений 7](#_Toc27910433)

[2.2. Анализ пропущенных значений 7](#_Toc27910434)

[2.3. Заполнение пропущенных значений 8](#_Toc27910435)

[2.4. Введение фиктивных переменных 12](#_Toc27910436)

[2.5. Удаление лишних фиктивных переменных 13](#_Toc27910437)

[3. Формирование обучающей, тестовой и валидационной выборок 13](#_Toc27910438)

[3.1. Формирование валидационной выборки 13](#_Toc27910439)

[3.2. Разделение на тестовую и обучающую выборки 14](#_Toc27910440)

[4. Бэггинг 14](#_Toc27910441)

[4.1. Встроенный бэггинг 15](#_Toc27910442)

[4.2. Разработанный бэггинг 15](#_Toc27910443)

[5. Отбор признаков 19](#_Toc27910444)

[5.1. Отбор с помощью feature selector 19](#_Toc27910445)

[5.2. С помощью случайного леса 21](#_Toc27910446)

[5.3. С помощью логистической регрессии 23](#_Toc27910447)

[6. AdaBoost 24](#_Toc27910448)

[7. Логистическая регрессия 25](#_Toc27910449)

[8. Метрики качества 26](#_Toc27910450)

[8.1. Критерий средней ошибки на контрольных данных 26](#_Toc27910451)

[8.2. Критерий «Precision» 26](#_Toc27910452)

[8.3. Критерий «Recall» 26](#_Toc27910453)

[8.5 F-мера 26](#_Toc27910454)

[8.5. Критерий «ROC-curve» 27](#_Toc27910455)

[Выводы 29](#_Toc27910456)

[Приложение 30](#_Toc27910457)

# Постановка задачи

Для исследования была использована выборка банковских данных о выданных

кредитах. Данные были взяты из разных банковских учреждений США.

Общий объем выборки составил 100514 строк. Анализировались 19 переменных

(включая целевую).

Переменные для анализа:

1. Уникальный идентификатор кредита (Loan ID);
2. Уникальный идентификатор клиента (Customer ID);
3. Статус кредита - целевая переменная (Loan Status). Принимает значения:
   1. Fully Paid (кредит выплачен)
   2. Charged Off (кредит не выплачен);
4. Непогашенная часть кредита ( Current Loan Amount );
5. Срок кредитования (Term). Принимает значения:
   1. Short Term (краткосрочный кредит)
   2. Long Term (долгосрочный кредит);
6. Кредитный рейтинг банка ( Credit Score)
7. Годовой доход клиента (Annual Income );
8. Стаж (Years in current job). Принимает значения:
   1. < 1 year
   2. 1 year
   3. 2 years
   4. 3 years
   5. 4 years
   6. 5 years
   7. 6 years
   8. 7 years
   9. 8 years
   10. 9 years
   11. 10+ years
9. Недвижимость в собственности (Home Ownership). Принимает значения:
   1. Home Mortgage (залог недвижимости в качестве обеспечения)
   2. Home (владеет собственной недвижимостью)
   3. Rent (снимает недвижимость)Have Mortgage (имеет ипотеку)
10. Цель займа (Purpose )
    1. Home Improvements (улучшение жилищных условие)
    2. Debt Consolidation (закрытие долгов)
    3. Buy House (покупка недвижимости)
    4. Business Loan (кредит на развитие бизнеса)
    5. Buy a Car (покупка автомобиля)
    6. Major purchase (крупная покупка)
    7. Take a Trip (поездка)
    8. Small business (развитие малого бизнеса)
    9. Medical Bills (медицинские счета)
    10. Wedding (свадьба)
    11. Vacation (отдых)
    12. Educational Expenses (на образование)
    13. Moving (на переезд)
    14. Renewable energy
    15. Other (прочие)
11. Месячная выплата по кредиту (Monthly Debt );
12. Длительность кредитной истории (Years of Credit History);
13. Количество месяцев с последней задолженности (Months since last delinquent);
14. Количество открытых счетов (Number of Open Accounts );
15. Количество «проблемных» кредитов (Number of Credit Problems );Т
16. Текущая задолженность (Current Credit Balance );
17. Количество открытых кредитов ( Maximum Open Credit );
18. Наличие банкротства (Bankruptcies );
19. Налоговые затраты (Tax Liens)

Будем рассматривать множество объектов X размерности N, заданных на признаковом пространстве D, множество меток Y = {0, 1}, и целевую функцию y ∗ : X → Y, значения которой известны на конечном наборе Xtrain = X1, X2, .., Xn ∈ X и равны Ytrain = y1, y2, ..., yn ∈ Y соответственно.

Совокупность {Xtrain,Ytrain} называется обучающей выборкой. Остальные Ntest = N −Ntrain элементов, для которых неизвестно значение целевой функции, образуют тестовую выборку Xtest. Задача обучения с учителем состоит в том, чтобы по обучающей выборке научиться восстанавливать зависимость y ∗ , т.е. построить такое отображение X → Y, которое приближает целевую функцию y ∗ , как на обучающей выборке, так и на тестовой.

В задаче бинарной классификации имеется два класса и целевая переменная y принимает два значения 0 и 1: yk =1, если объект xk принадлежит классу 1 и yk=0, если объект xk принадлежит классу 0.

## Ход работы

Поэтапное описание работы приведено в таблице №1 «Этапы работы».

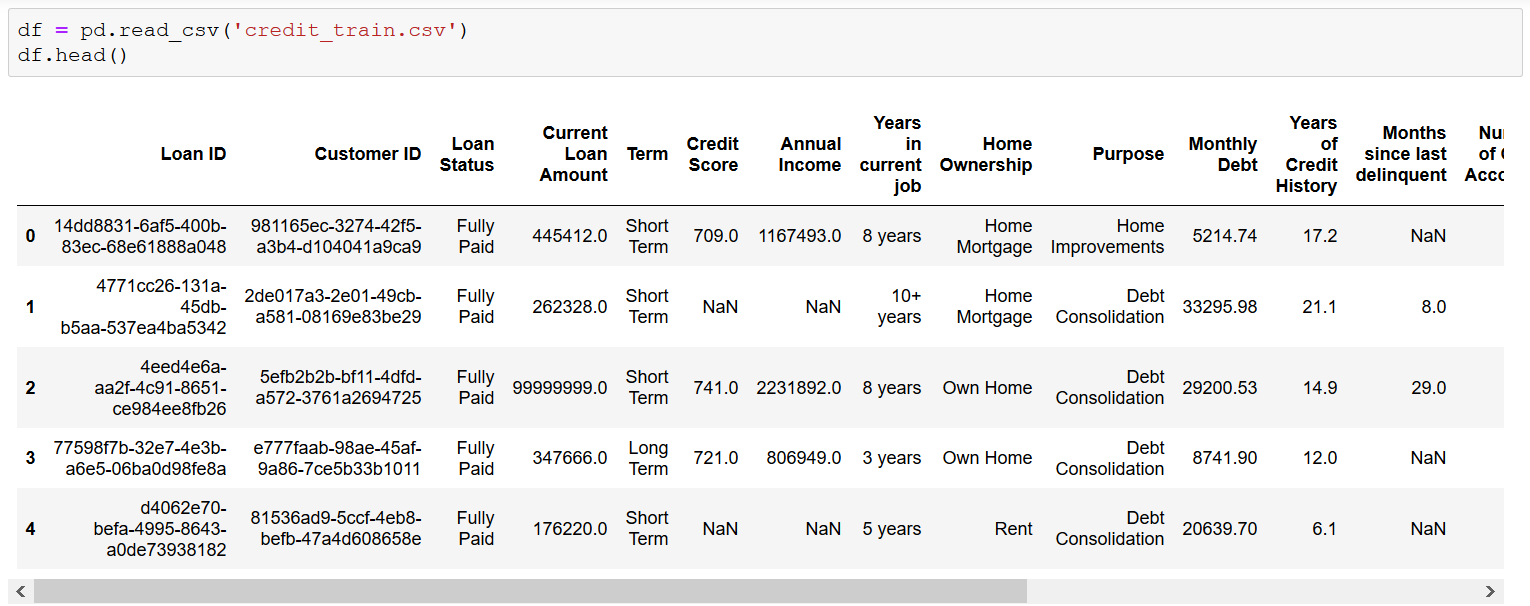
*Таблица №1 «Этапы работы»*

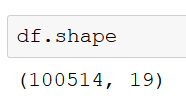
|  |  |
| --- | --- |
| Этап | Задачи |
| 1. Первичный анализ данных | 1.1. Определение размера выборки |
| 1.2. Определение типов переменных |
| 1.3. Определение категориальных переменных и принятие решения о введении фиктивных переменных |
| 1.4. Анализ целевой переменной |
| 2. Предобработка данных | 2.1. Удаление пустых значений |
| 2.2. Анализ пропущенных значений |
| 2.3. Заполнение пропущенных значений |
| 2.4. Введение фиктивных переменных |
| 2.5. Удаление лишних фиктивных переменных для избежания мультиколлинеарности |
| 3. Формирование обучающей, тестовой и валидационной выборок | 3.1. Формирование валидационной выборки |
| 3.2. Разбиение оставшейся части выборки на обучающую и тестовую |
| 4. Бэггинг | 4.1. Использование «встроенного» бэггинга |
| 4.2. Разработанный самостоятельно бэггинг |
| 5. Отбор признаков | 5.1. Отбор «feature selector» |
| 5.2. Отбор «случайный лес» |
| 5.3. Отбор логистической регрессией |
| 6. AdaBoost | 6. Алгоритм AdaBoost |
| 7. Логистическая регрессия | 7. Определение оптимальных параметров для регрессии |
| 8. Метрики качества | 8.1. Критерий accuracy |
| 8.2. Критерий precision |
| 8.3. Критерий recall |
| 8.4. F-мера |
| 8.5. Критерий roc-curve |

# 1. Первичный анализ данных

## 1.1. Определение размера выборки

Проведем первичное исследование данных.

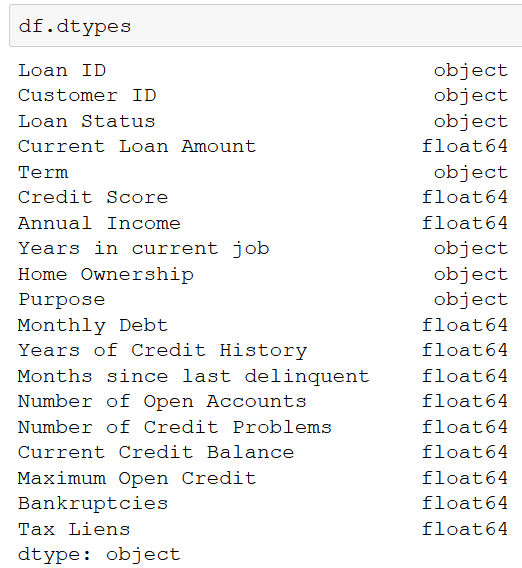




Исследование показало, что выборка содержит 100514 объектов, описанных 19 признаками.

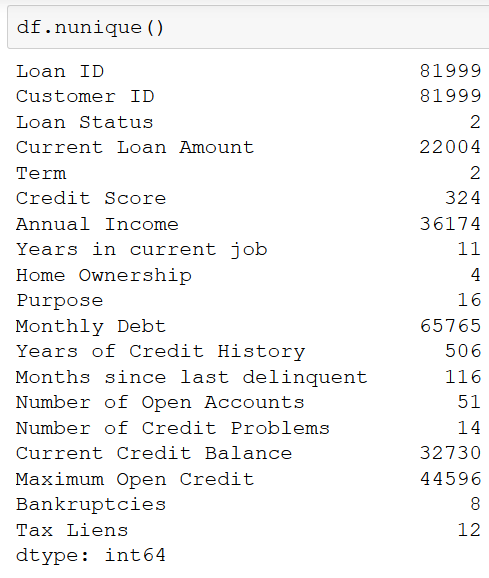
## 1.2. Определение типов переменных

Проанализируем типы данных:



## 1.3. Определение категориальных переменных

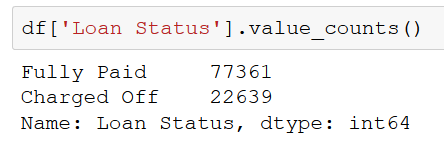
Данные содержат 5 категориальных переменных (переменные Loan ID и Customer неинформативны). Таким образом, можно сделать предположение о том, что для их интерпретации будет необходимо ввести фиктивные переменные. Посмотрим количество уникальных значений каждой переменной:



Те атрибуты, для которых было выдвинуто предположение о введении фиктивных переменные, имеют мало уникальных значений, что подтверждает предположение.

## 1.4. Анализ целевой переменной

Проведем анализ распределения значений целевой переменной:

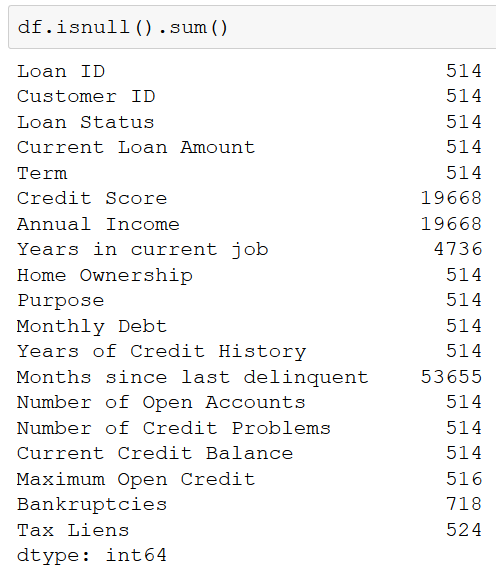


Анализ показал, что количество выплаченных кредитов составляет 77361, а невыплаченных - 22693. Из этого следует, что выборка несбалансирована.

# 2. Предобработка данных

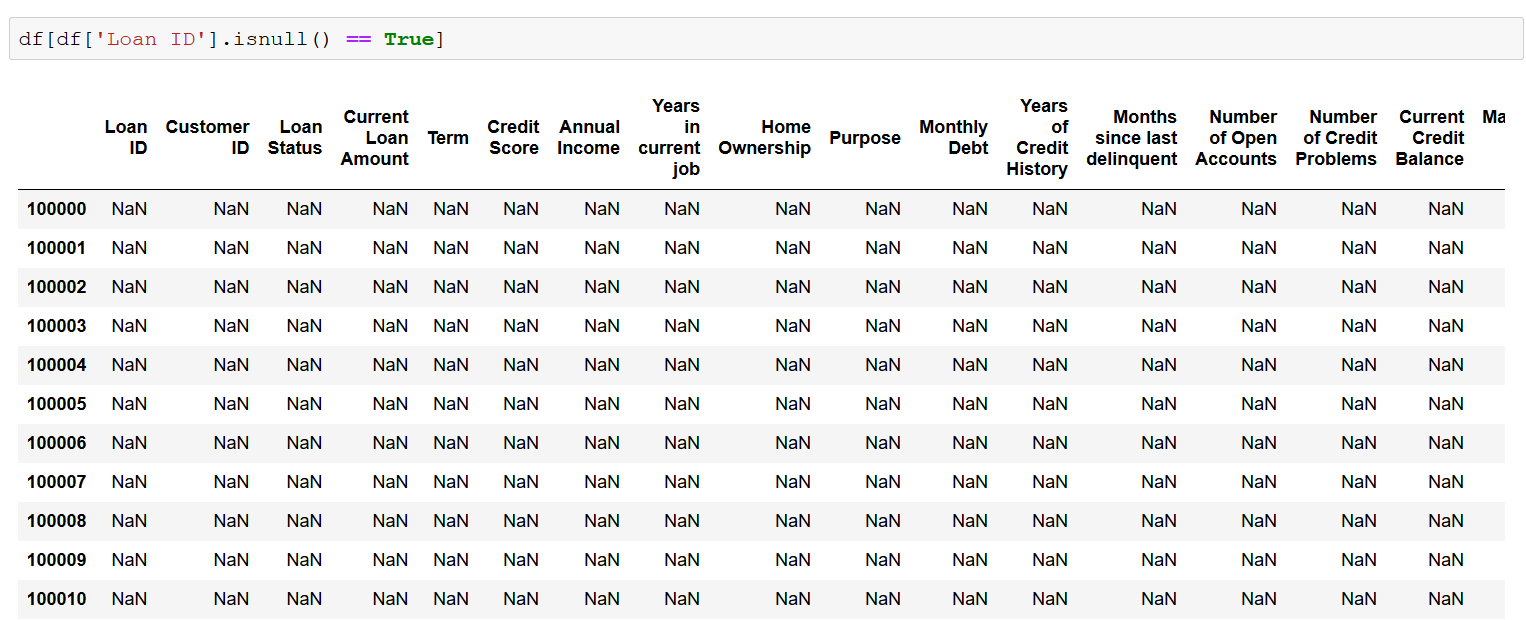
## 2.1. Удаление пустых значений

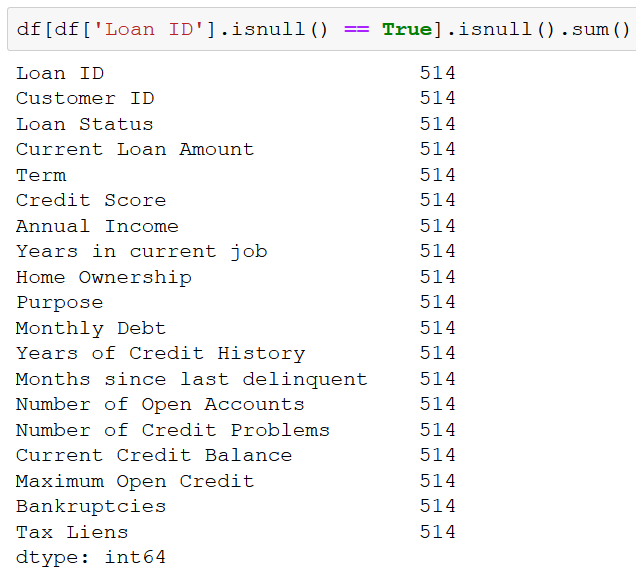
Определим количество пропущенных значений по каждой из переменных:



## 2.2. Анализ пропущенных значений

Результаты анализа позволяют выдвинуть предположения о том, что в выборке 514 строк не заполнены. Проверим наше предположение. Выведем эти строки:



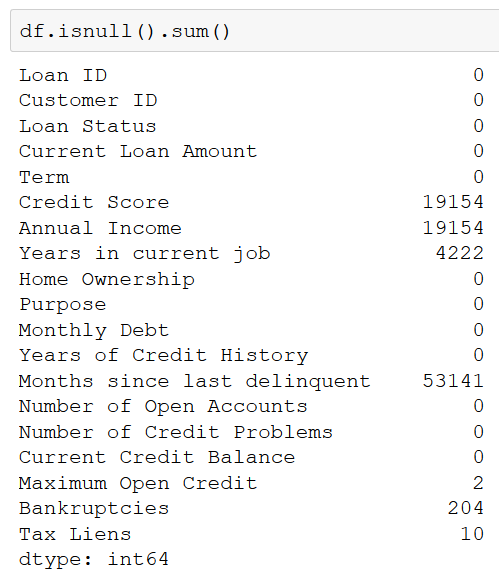


Наше предположение подтвердилось. Удалим объекты из выборки.

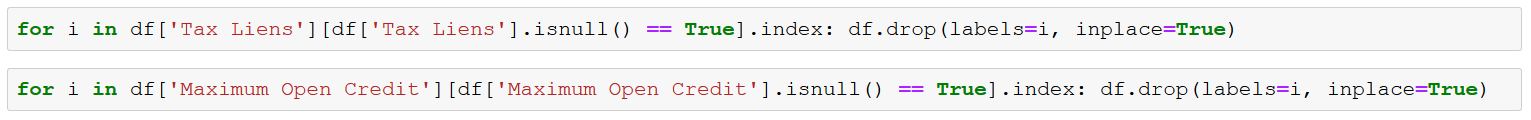
https://lh3.googleusercontent.com/nVIX3Lt_V4rEpbaU6Zxm0a3UpZ1ZWRuYNs3x1XEaprEtH9nIFHvHKXknmUDky_xmcNhtjvcTtbVCW6Bp-h61kdoBNvrZV3PpEFbgGxgCgvFo2fu85bIS1-fIt4kMcszIMspoicRv

## 2.3. Заполнение пропущенных значений

Проведем анализ оставшихся пропущенных значений:



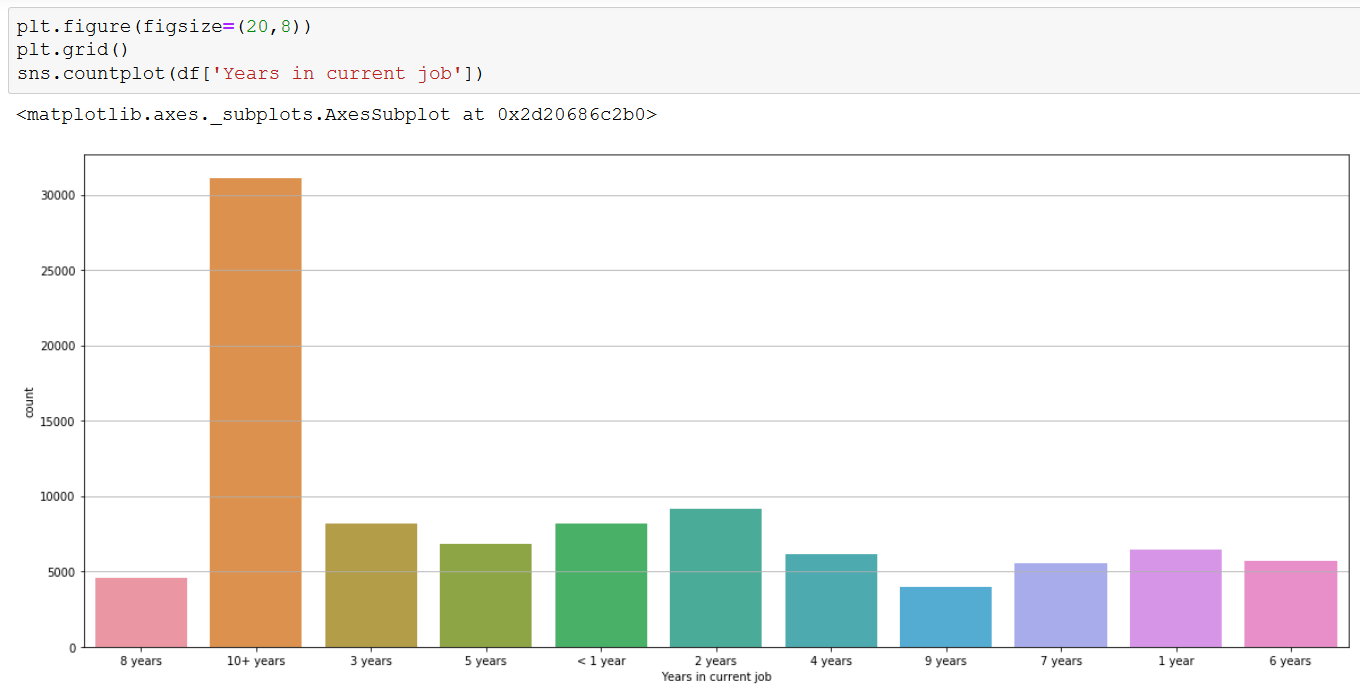
По переменным Maximum Open Credit и Tax Liens нулевые строки имеют маленький удельный вес относительно общего объема, удалим данные строки.



В переменной Months since last delinquent пропущенных значений гораздо больше и удалять их нецелесообразно. В этой переменной хранится время, прошедшее с момента последней задолженности выплаты по кредиту. Таким образом, можно предположить, что если информации в ней нет, то ежемесячные выплаты всегда платились вовремя. Условно заполняем пустые значения “-1”:

https://lh3.googleusercontent.com/dS6cyel_rMN5jlf2agfVsdrOJsB7izzkz2DdJbCJyJPWW9ITnVXeD_D1N1s7xlfUuC_n93PESHU0XgcERZ4BK8w0Y1OvD0zTP0NLhpA5uw0v7HAjUpe4lGIlfnFJFrZuU7F86mk3

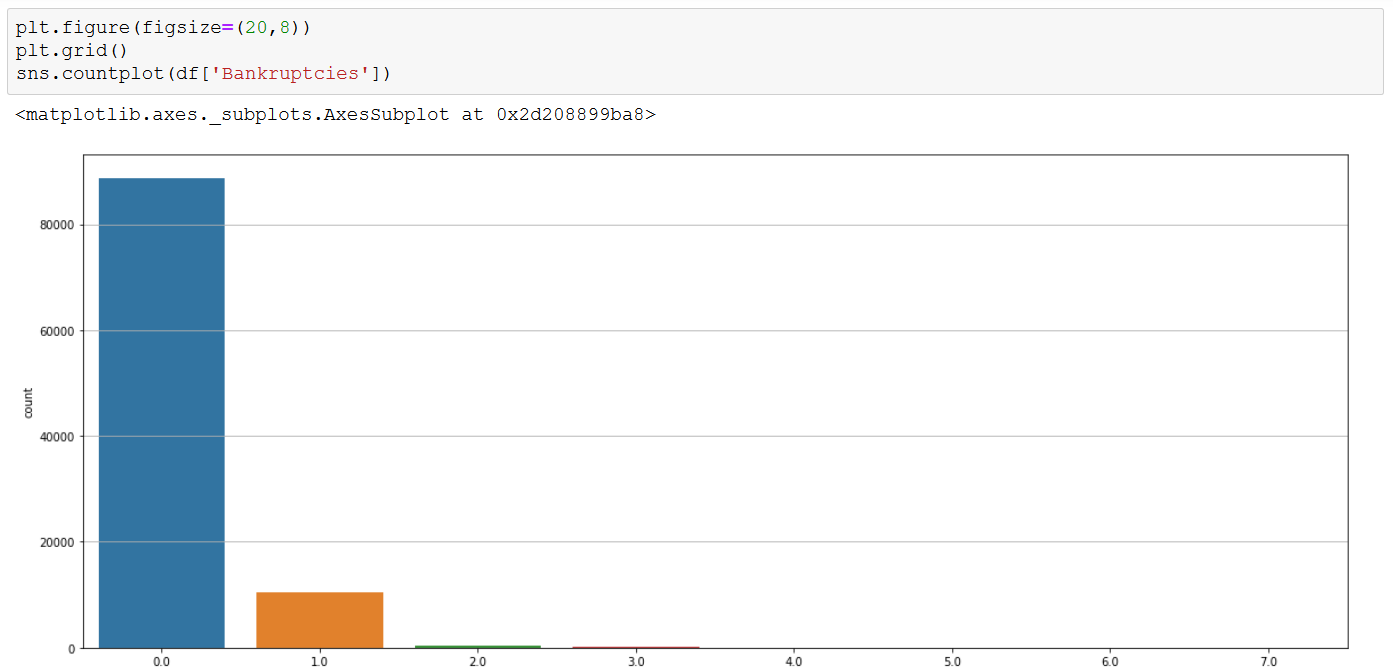
После проведенной замены осталось 4222 строки, в которых не заполнен стаж. Построим гистограмму распределения значений:



Анализ показал, что подавляющее большинство заемщиков имеет стаж больше 10 лет. Заполним пустые значения наиболее часто встречающимся:

https://lh3.googleusercontent.com/pQrOMnzPurNGcgJc9D7-IEkwLlxDCISwYHifwFtWTFqHmlqq2iV0Yst58j9tB0mkZ6DiiFaUBh6tkPHO1eE-9ytYAcp4NHdVvHi-nvDgU2zV7l-LZq0I-PTgm7gWNG3y_hsnu1pk

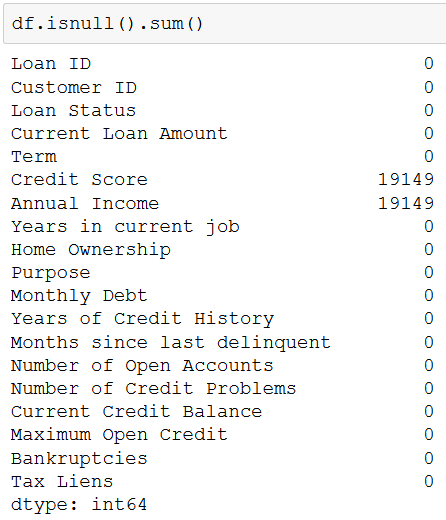
Однако, осталось 194 строки, в которых не указан факт банкротства заемщика. Проведем аналогичный анализ. Строим гистограмму:



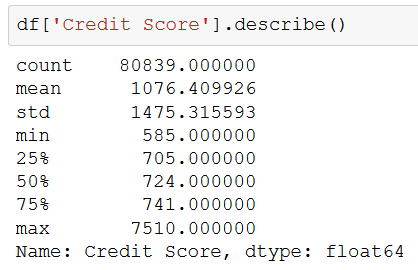
Заполняем пустые значения нулем, т.к. он встречается чаще всего.

https://lh5.googleusercontent.com/HPR0B72A5NCUIcn1gZQqfQGaO5DcKFwSCx_alX0qqxeERNdBI6H3ObGppSDwuJYhWJrFeYUObmYZX7xZ2rVN1_r2T1ODbQmws2MAOA2BjPHcg6AmpuS_zOGSW_g9Z9Rc02T2-OiK

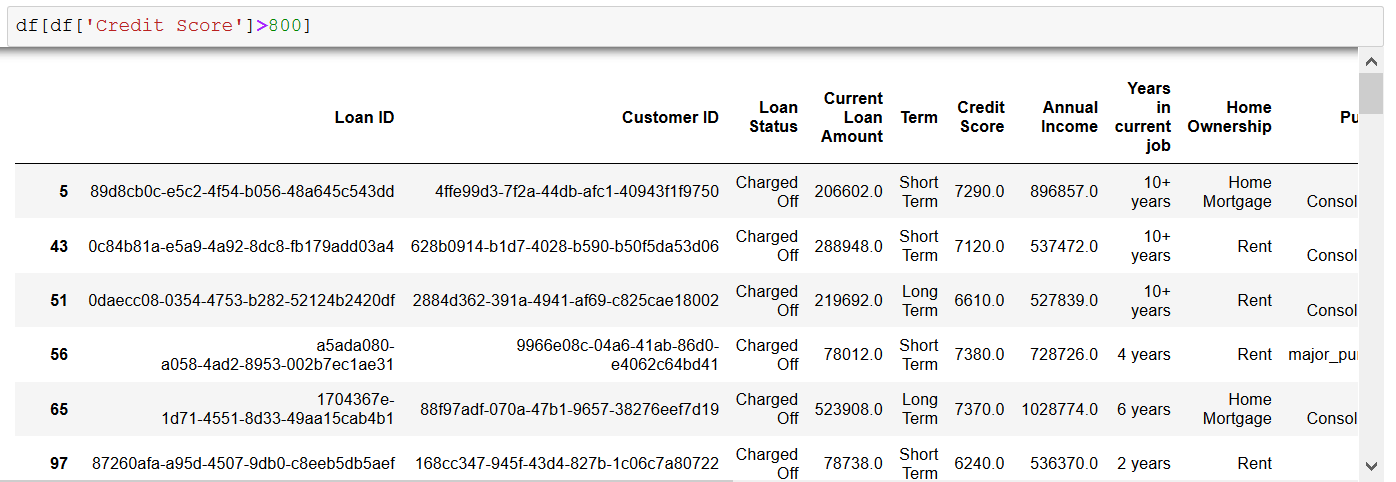
Проверяем количество оставшихся пустых значений:



Так как переменные Credit Score и Annual Income – численные переменные, их нельзя заполнить самым часто встречающимся значением. Выведем описательные статистики для Credit Score:



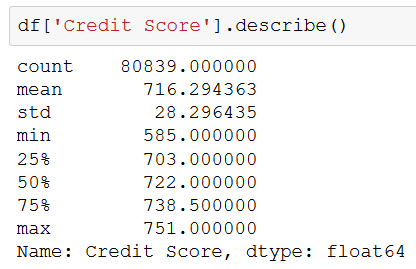
Можно сделать вывод: среднее значение = 1076, тогда как значение Credit Score не может превышать 850. Необходимо найти причину такого значения. Выведем все строки, где Credit Score больше 800:



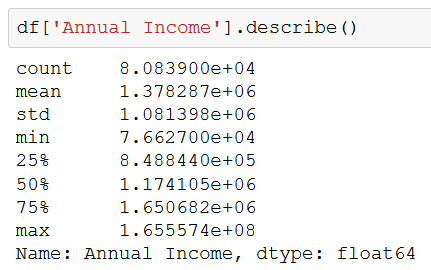
Анализ показывает, что в некоторых строках значение credit score увеличено в 10 раз. Разделим данные значения на 10:

https://lh4.googleusercontent.com/9GpEqx9uXLvivF1FeoSwfkIKKqNgRsujA2_-_sSUAC2ivO30BeYiNhL3JoS2AkHDK1BF5VlWQm0U-zKM1Fpa9vpziGRbuQTwF2ZnR-hqf9p2C1b0M34Z1gSsk9bIHcomUYv1rziB

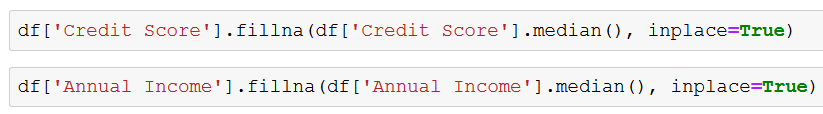
Повторно выведем описательные статистики:



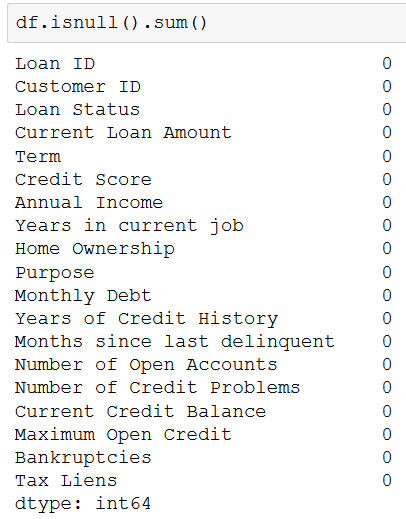
Затем, выводим описательные статистики для Annual Income:



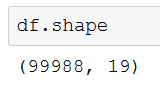
В обоих случаях можно заполнить пустые значения средними значениями, однако арифметическое среднее часто бывает нерепрезентативно. Заполним значения медианой.

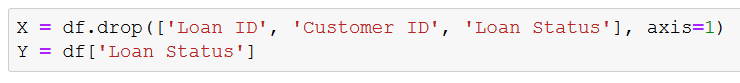


Проведем финальный анализ данных.

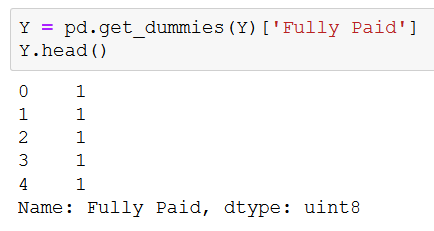


После очистки данных в выборке осталось 99988 строк:



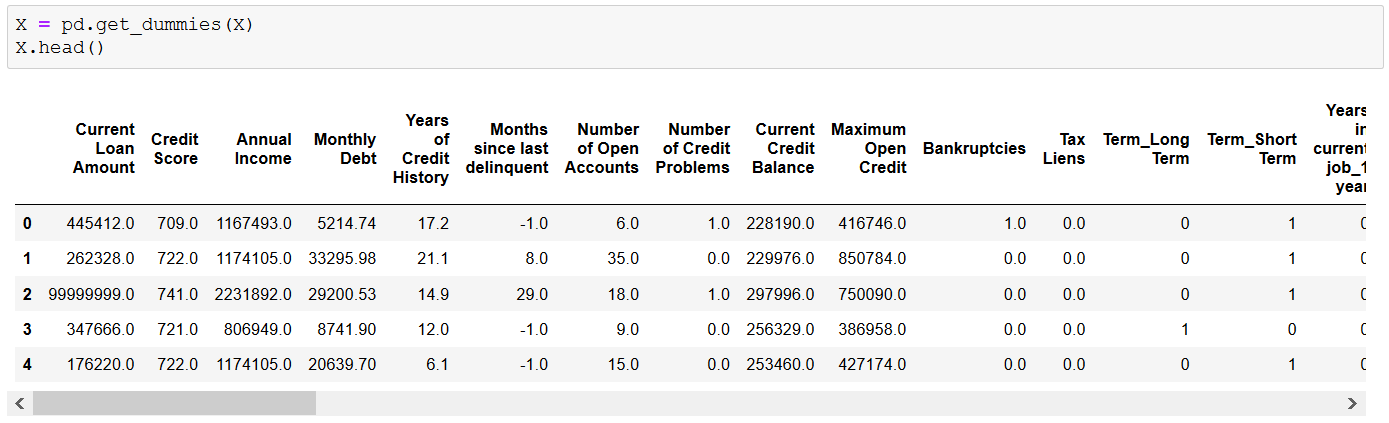
Разделим набор данных на целевую и независимые переменные. Loan ID и Customer ID неинформативны и использоваться в дальнейшем анализе не будут.

Закодируем значения целевой переменной в 0 и 1 (1 – ‘fully paid’, 0 – ‘charge off’).



## 2.4. Введение фиктивных переменных

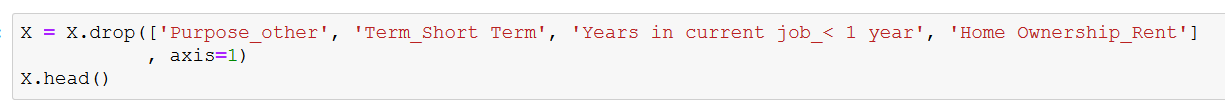
Необходимо сформировать фиктивные переменные для категориальных независимых переменных:



Таким образом, получили следующий набор признаков:

* Current Loan Amount
* Credit Score
* Annual Income
* Monthly Debt
* Years of Credit History
* Months since last delinquent
* Number of Open Accounts
* Number of Credit Problems
* Current Credit Balance
* Maximum Open Credit
* Bankruptcies
* Tax Liens
* Term\_Long Term
* Term\_Short Term
* Years in current job\_1 year
* Years in current job\_10+ years
* Years in current job\_2 years
* Years in current job\_3 years
* Years in current job\_4 years
* Years in current job\_5 years
* Years in current job\_6 years
* Years in current job\_7 years
* Years in current job\_8 years
* Years in current job\_9 years
* Years in current job\_< 1 year
* Home Ownership\_HaveMortgage
* Home Ownership\_Home Mortgage
* Home Ownership\_Own Home
* Home Ownership\_Rent
* Purpose\_Business Loan
* Purpose\_Buy House
* Purpose\_Buy a Car
* Purpose\_Debt Consolidation
* Purpose\_Educational Expenses
* Purpose\_Home Improvements
* Purpose\_Medical Bills
* Purpose\_Other
* Purpose\_Take a Trip
* Purpose\_major\_purchase
* Purpose\_moving
* Purpose\_other
* Purpose\_renewable\_energy
* Purpose\_small\_business
* Purpose\_vacation
* Purpose\_wedding

Необходимо избежать мультиколлинеарности - уберем те фиктивные переменные, которые дублируют информацию:



## 2.5. Удаление лишних фиктивных переменных

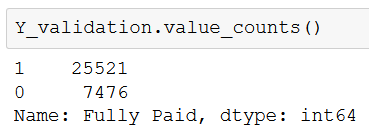
Для деления выборки на обучающую, валидационную и тестовую воспользуемся функцией: https://lh6.googleusercontent.com/M6CMLE-dO1D6lSAIAwuJHH9OlWntX3KDvTEX24elWRAWIUyXK3WbhZcy0505K8erJvumNB1mxCAhKEI0Q8PvBuBbSBaD7s7toCCo2AVjcsZs-1dQUJVmguE5gdagwx5r3h2J4xCb

# 3. Формирование обучающей, тестовой и валидационной выборок

## 3.1. Формирование валидационной выборки

Выделим валидационную выборку (33% от общего объема) и проверим, что она содержит в себе объекты обоих классов:

https://lh6.googleusercontent.com/kzcTqm2KImLrcqHvMup6kW0oFLZsR41sqcQFvfB_N351qHaNKpje4L-owHGfCC3X3MlicA60Fq9mldQORKUhKjcdezSEIxTAxWrcpccx0ueoZm4X4bPvMqzmoYGHvNKxR1x5m0jO

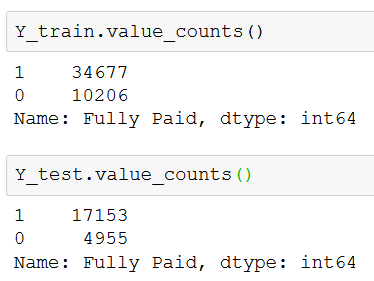


## 3.2. Разделение на тестовую и обучающую выборки

Разделим оставшуюся выборку на обучающую (67%) и тестовую (33%) подвыборки:

https://lh6.googleusercontent.com/J4tOlW5nAp5PVmJbt3_6bKFkUmip4C7hCjthMzM2e_U6AlTCceMRZXx2JWtOqqKhFU9rGeOyr1r7Kuqqhe-ZzRPxZkSPORj5PKZIavub_VDJ-Ckr8848vONEs_9CKPOLDq2zQ63R

Также проверим, что они содержат в себе объекты обоих классов:



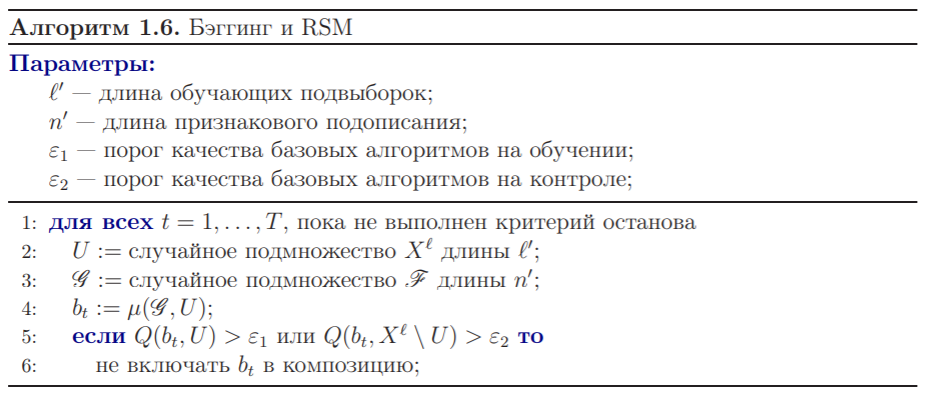
**4. Бэггинг**

Базовые алгоритмы, составляющие линейную комбинацию, должны быть в достаточной степени различными, чтобы их погрешности компенсировали друг друга. Нет никакого смысла складывать одинаковые (или почти одинаковые) алгоритмы. В бустинге и других последовательных методах различность достигается благодаря пересчёту весов объектов. Но возможна и другая стратегия повышения различности, когда базовые алгоритмы настраиваются независимо друг от друга на случайно выбранных подмножествах обучающей выборки, либо на различных случайных подмножествах признаков. Ещё один способ обеспечить различность — выбирать случайные начальные приближения при оптимизации вектора параметров (обычно так поступают при настройке нейронных сетей), либо применять стохастические алгоритмы оптимизации. Полученные базовые алгоритмы объединяются в композицию с помощью простого или взвешенного голосования. В случае взвешенного голосования для настройки коэффициентов αt применяются стандартные линейные методы.

Метод бэггинга (bagging, bootstrap aggregation) был предложен Л. Брейманом в 1996 году. Из исходной обучающей выборки длины ℓ формируются различные обучающие подвыборки той же длины ℓ с помощью бутстрепа — случайного выбора с возвращениями. При этом некоторые объекты попадают в подвыборку по несколько раз, некоторые — ни разу. Можно показать, что доля объектов, оказавшихся в каждой подвыборке, стремится к 1 − e −1 ≈ 0.632 при ℓ → ∞. Базовые алгоритмы, обученные по подвыборкам, объединяются в композицию с помощью простого голосования. Эффективность бэггинга объясняется двумя обстоятельствами. Во-первых, благодаря различности базовых алгоритмов, их ошибки взаимно компенсируются при голосовании. Во-вторых, объекты-выбросы могут не попадать в некоторые обучающие подвыборки. Тогда алгоритм, построенный по подвыборке, может оказаться даже точнее алгоритма, построенного по полной выборке. Бэггинг особенно эффективен на малых выборках, когда исключение даже небольшой доли обучающих объектов приводит к построению существенно различных базовых алгоритмов. В случае сверхбольших избыточных выборок приходится строить подвыборки меньшей длины ℓ0 ≪ ℓ, при этом возникает задача подбора оптимального значения ℓ0.

Так как после введения фиктивных переменных количество признаков стало большим, используем RSM.

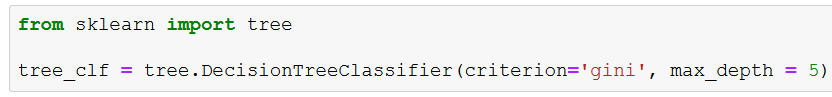
В методе случайных подпространств (random subspace method, RSM) базовые алгоритмы обучаются на различных подмножествах признакового описания, которые также выделяются случайным образом. Этот метод предпочтителен в задачах с большим числом признаков и относительно небольшим числом объектов, а также при наличии избыточных неинформативных признаков. В этих случаях алгоритмы, построенные по части признакового описания, могут обладать лучшей обобщающей способностью по сравнению с алгоритмами, построенными по всем признакам.

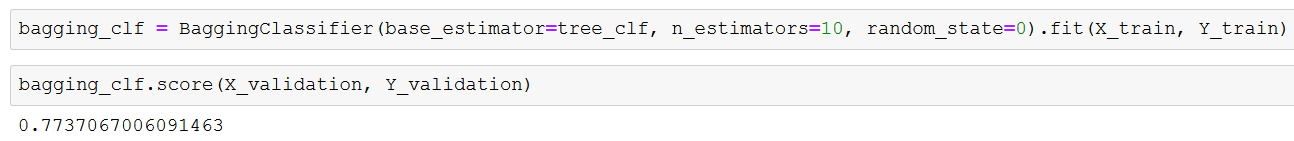


## 4.1. Встроенный бэггинг

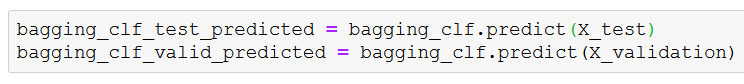
Минус данной функции в том, что на вход принимает только один тип базового классификатора (например, только дерево или только перцептрон). Однако, протестируем его. В качестве базового алгоритма выберем решающее дерево.

https://lh5.googleusercontent.com/20FMTjTiX7tPXY4Jw45MRloknLfi1YOQFffRG8kY34Hyy4sIi5lVgl1Vyc1oDDSseVAMN2xvnj4nE9ClLVqyaqf4F3GwYt8JXC4h1AnuMoeEiW2RzNZ4MpKXyPxGhLWTNiq5Q-r2





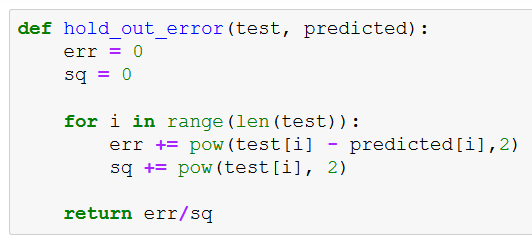
Рассчитаем ответы на тестовой и валидационной выборках:



## 4.2. Разработанный бэггинг

Плюсы данного подхода заключается в гибкости алгоритма и возможности индивидуальной настройки.

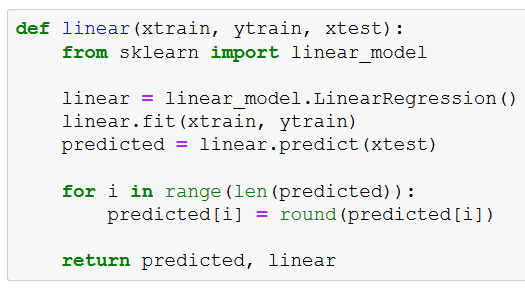
1. Сначала необходимо задать критерий оценивания. Средняя ошибка на контрольных данных:



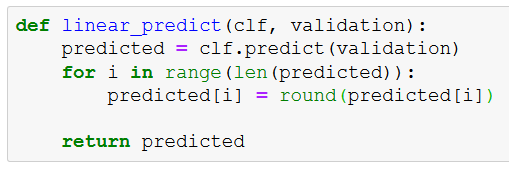
1. Далее, задаем базовые алгоритмы. Всего выбрали 5 базовых алгоритмов:

2.1. линейная регрессия

обучение:

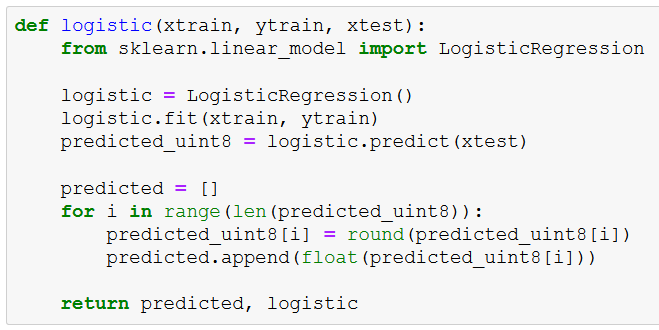


предсказание:

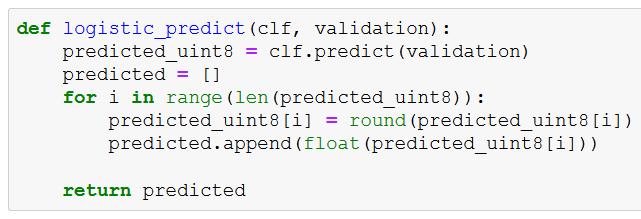


2.2. логистическая регрессия

обучение:

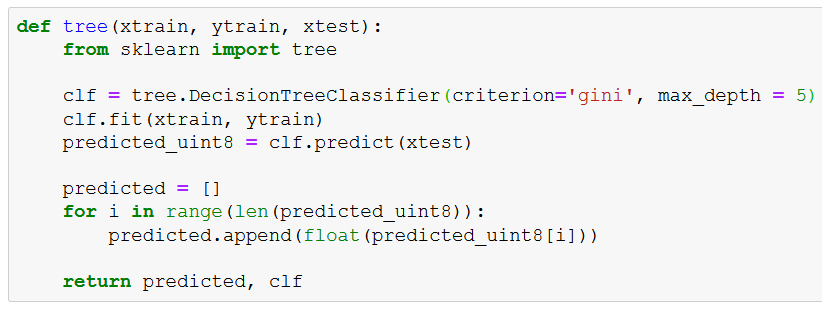


предсказание:

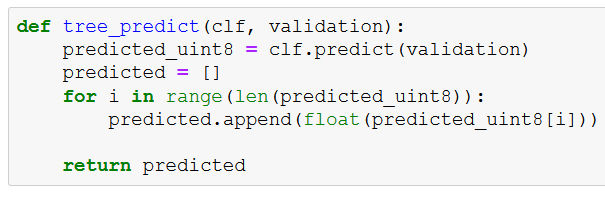


2.3. дерево решений

обучение:

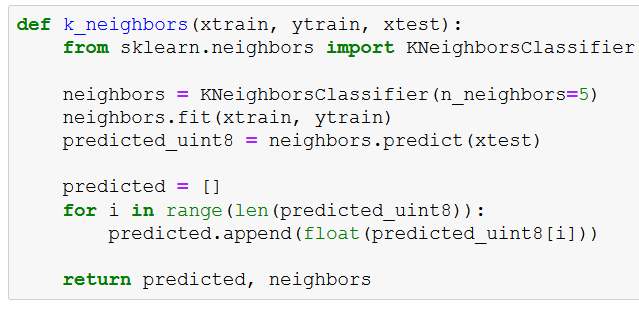


предсказание:

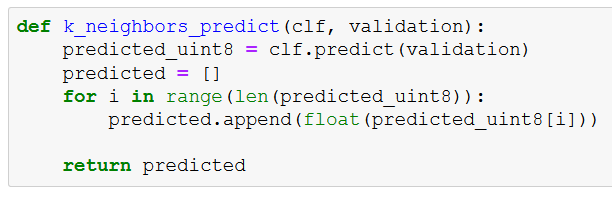


2.4. метод k ближайших соседей

Обучение

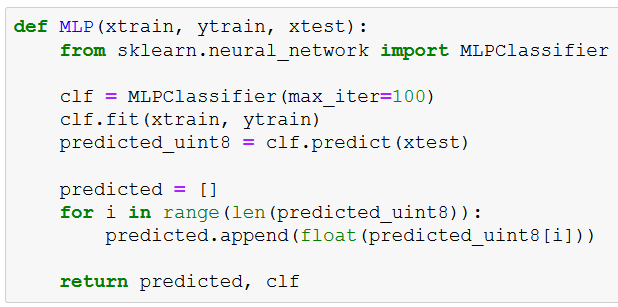


Предсказание

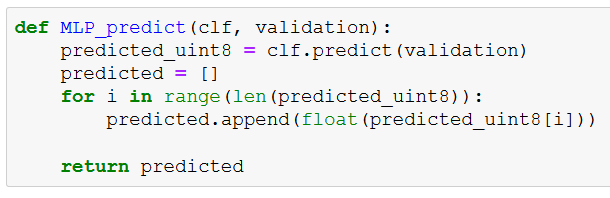


2.5. многослойный перцептрон

Обучение



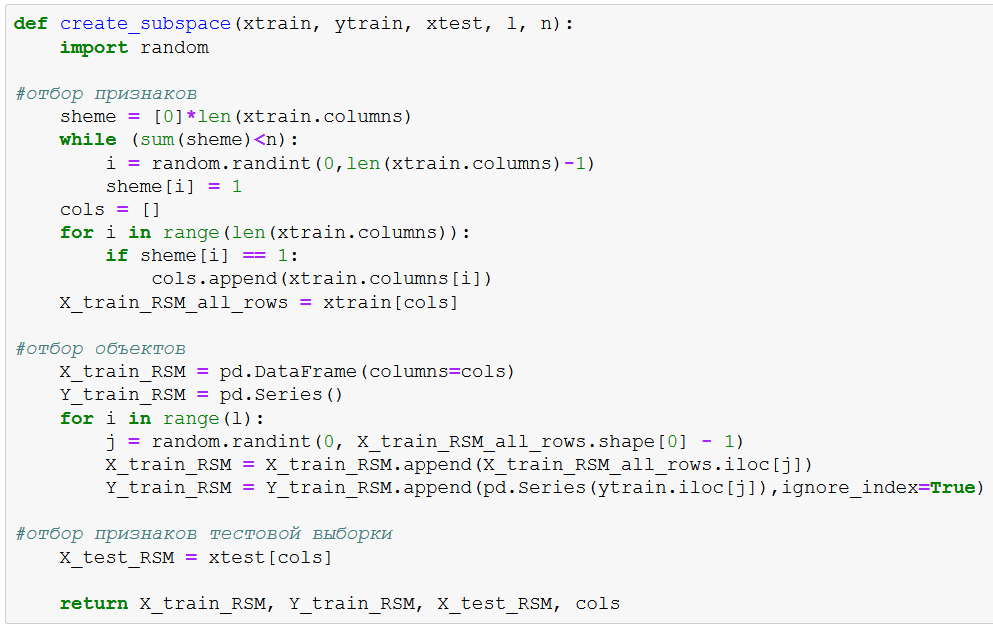
Предсказание



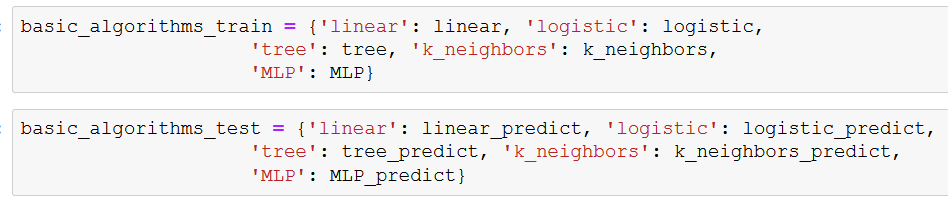
1. Метод случайных подпространств: формирование подпространства

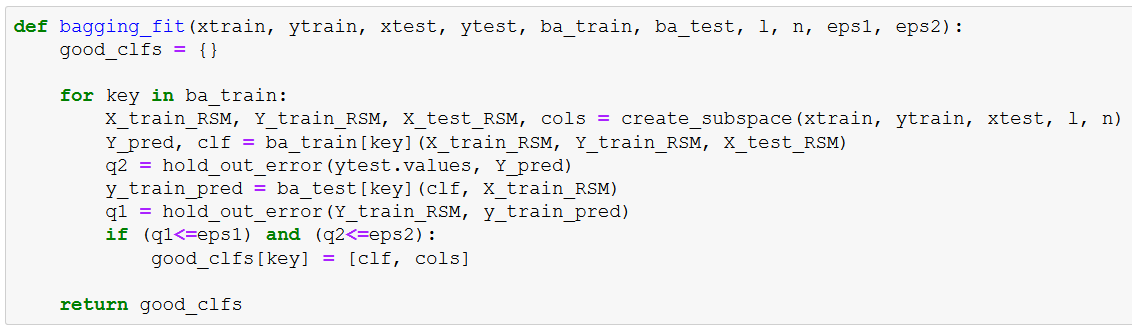
n - длина признакового подописания

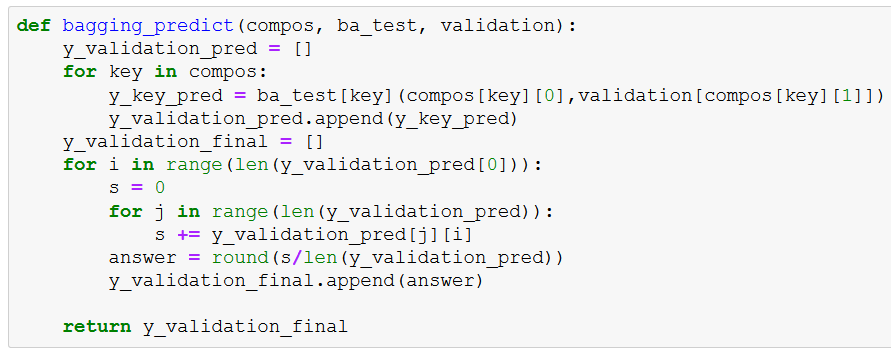
 l - длина выборки



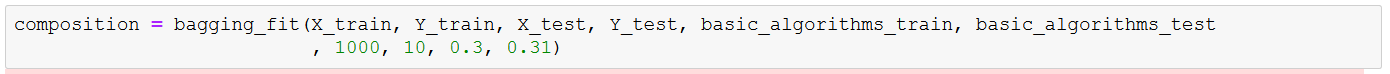
1. Бэггинг

4.1. словари базовых алгоритмов (для обучения и для предсказания): 

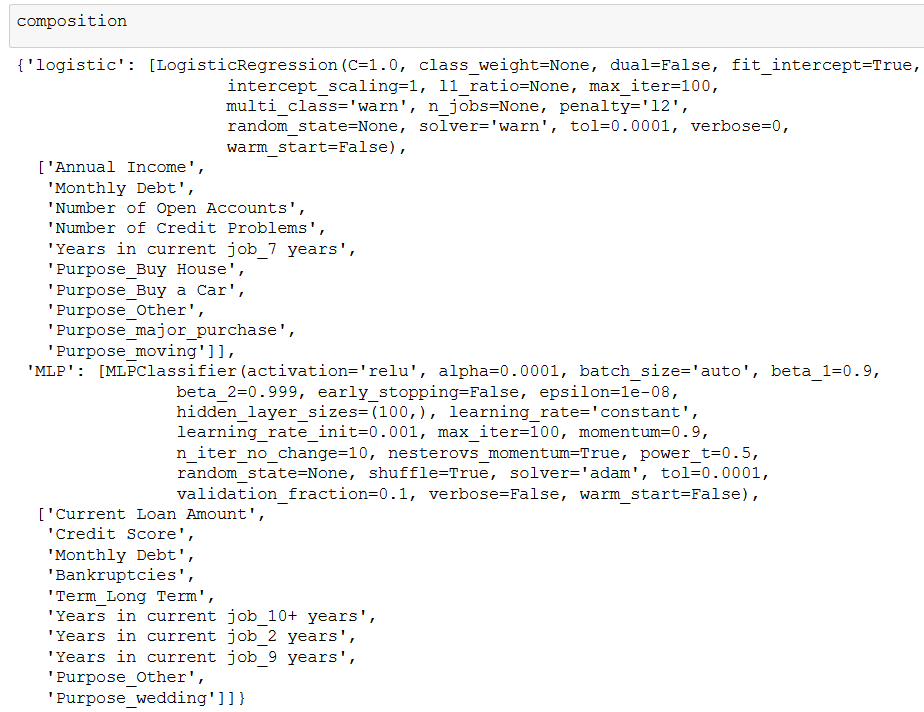
4.2. обучение:

4.3. предсказание:

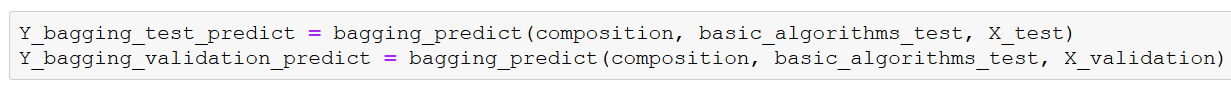
5. Вызов бэггинга



Получившаяся композиция:



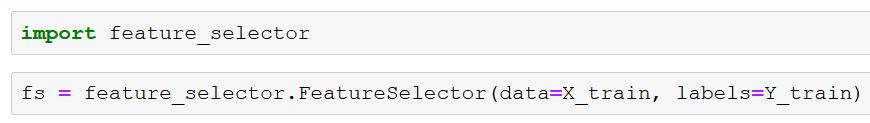
Рассчитаем ответы для теста и валидации:

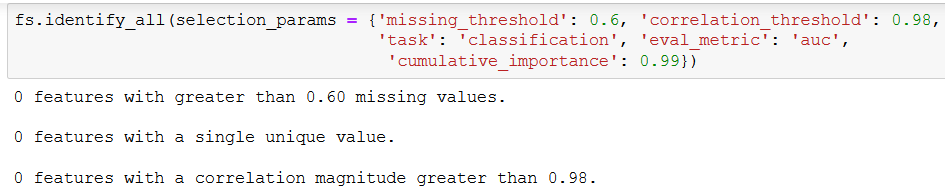


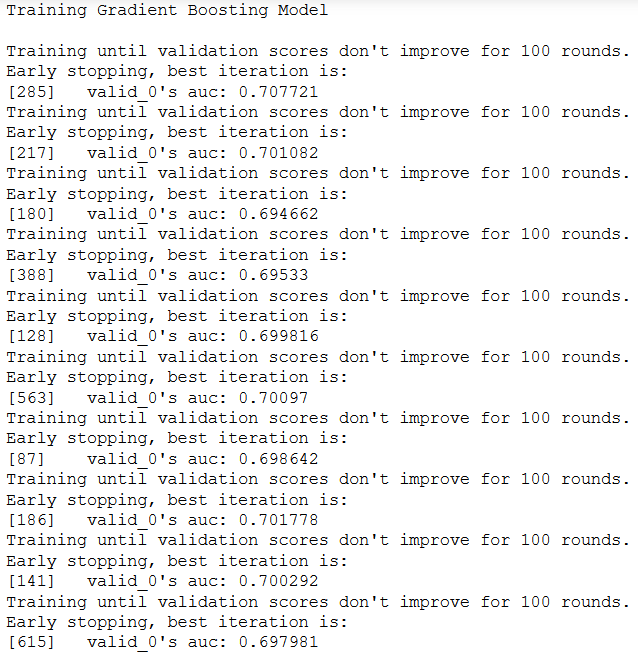
# 5. Отбор признаков

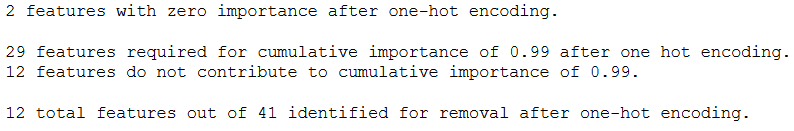
**5.1. Отбор с помощью feature selector**

Вызовем функцию и проведем анализ результатов:

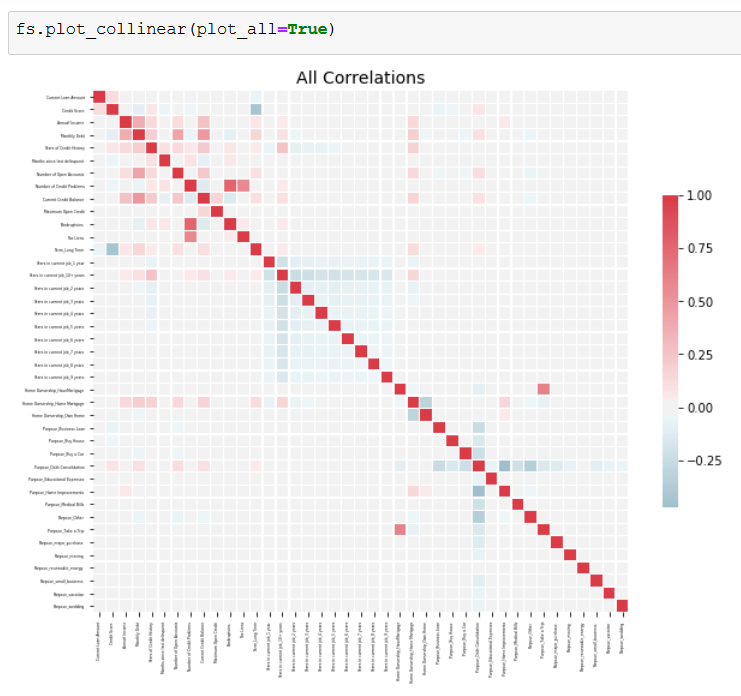




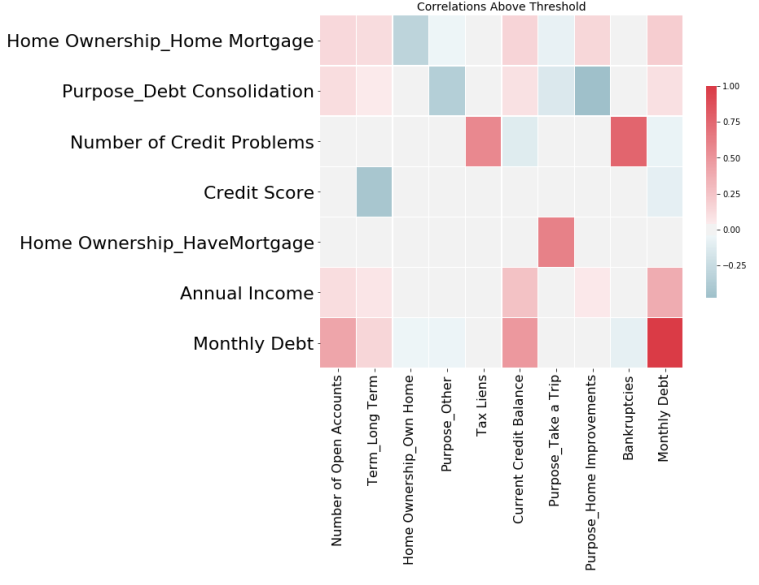




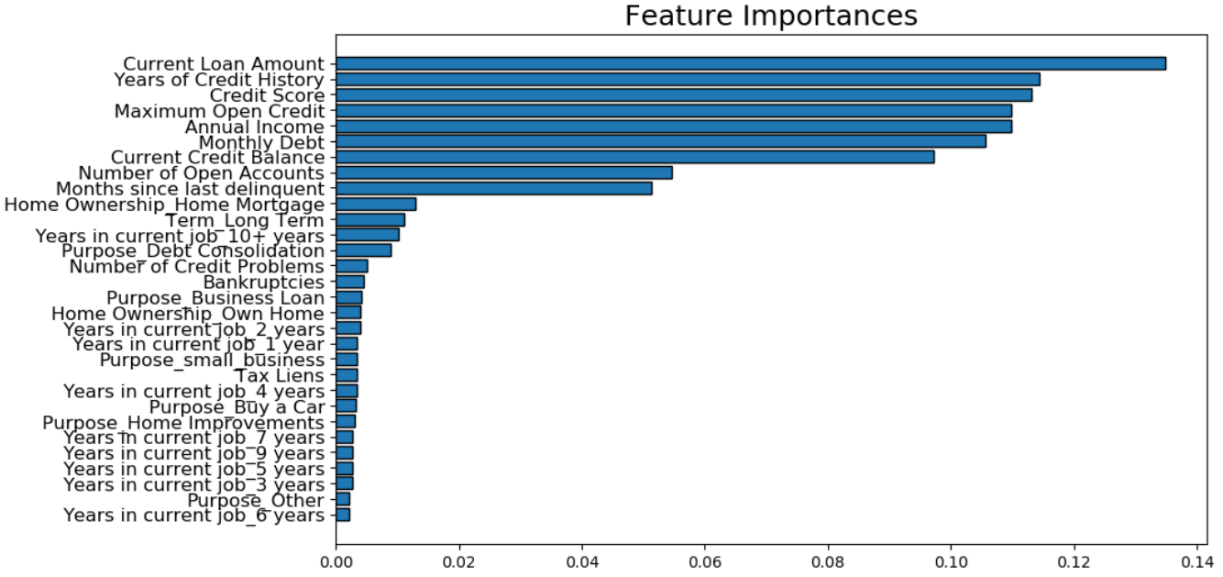
Построим матрицу корреляций между независимыми переменными:



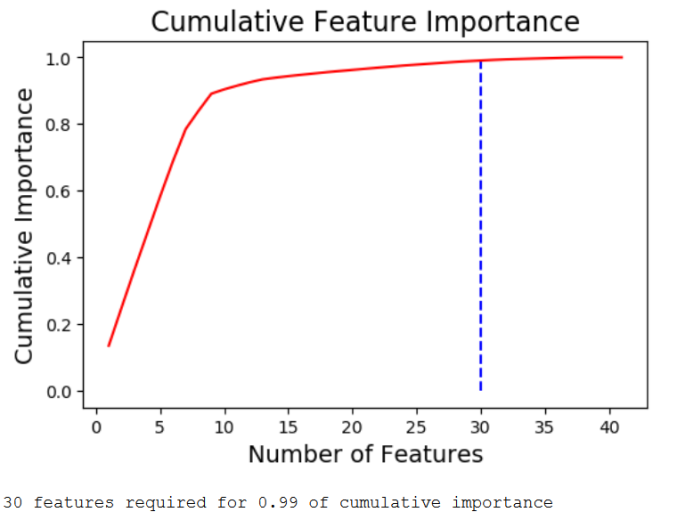
Коэффициент корреляции больше 0,3 для признаков:



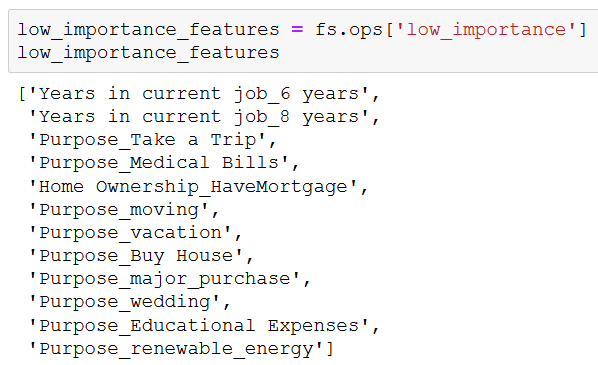
Значимость каждого из признаков:



Накопленная значимость признаков:



Неинформативные признаки:



**5.2. С помощью случайного леса**

Обучается модель классификации, в процессе будет получена значимость признаков:

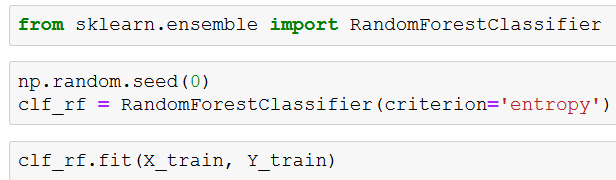
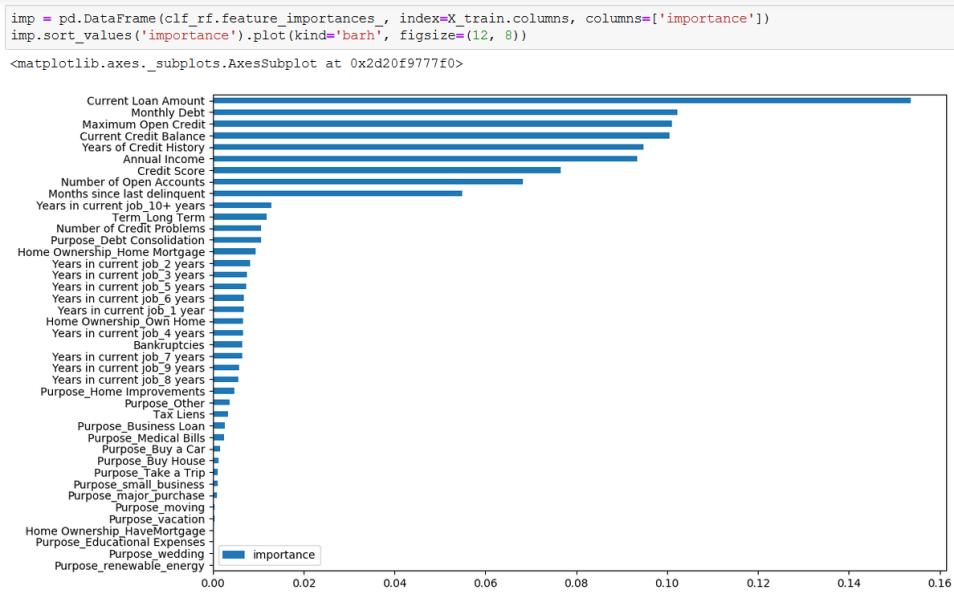


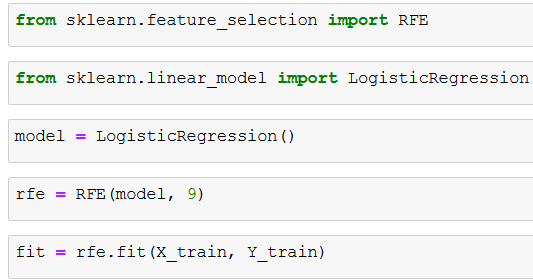
График значимости приведен ниже:



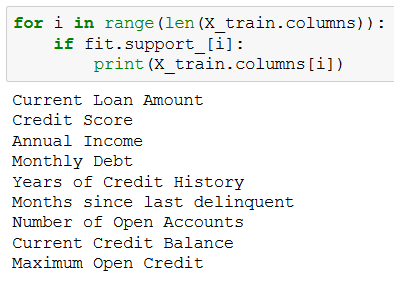
Признаки и их значимость:

|  |  |
| --- | --- |
| Current Loan Amount | 0.153712 |
| Monthly Debt | 0.102289 |
| Maximum Open Credit | 0.101119 |
| Current Credit Balance | 0.100560 |
| Years of Credit History | 0.094729 |
| Annual Income | 0.093459 |
| Credit Score | 0.076604 |
| Number of Open Accounts | 0.068211 |
| Months since last delinquent | 0.054843 |
| Years in current job\_10+ years | 0.012950 |
| Term\_Long Term | 0.011765 |
| Number of Credit Problems | 0.010670 |
| Purpose\_Debt Consolidation | 0.010644 |
| Home Ownership\_Home Mortgage | 0.009459 |
| Years in current job\_2 years | 0.008157 |
| Years in current job\_3 years | 0.007536 |
| Years in current job\_5 years | 0.007314 |
| Years in current job\_6 years | 0.006788 |
| Years in current job\_1 year | 0.006786 |
| Home Ownership\_Own Home | 0.006666 |
| Years in current job\_4 years | 0.006558 |
| Bankruptcies | 0.006499 |
| Years in current job\_7 years | 0.006409 |
| Years in current job\_9 years | 0.005859 |
| Years in current job\_8 years | 0.005672 |
| Purpose\_Home Improvements | 0.004778 |
| Purpose\_Other | 0.003742 |
| Tax Liens | 0.003362 |
| Purpose\_Business Loan | 0.002650 |
| Purpose\_Medical Bills | 0.002491 |
| Purpose\_Buy a Car | 0.001618 |
| Purpose\_Buy House | 0.001336 |
| Purpose\_Take a Trip | 0.001062 |
| Purpose\_small\_business | 0.001027 |
| Purpose\_major\_purchase | 0.000987 |
| Purpose\_moving | 0.000466 |
| Purpose\_vacation | 0.000437 |
| Home Ownership\_HaveMortgage | 0.000290 |
| Purpose\_Educational Expenses | 0.000244 |
| Purpose\_wedding | 0.000232 |
| Purpose\_renewable\_energy | 0.000017 |

**5.3. С помощью логистической регрессии**

Обучается модель классификации, в процессе будет получена значимость признаков:

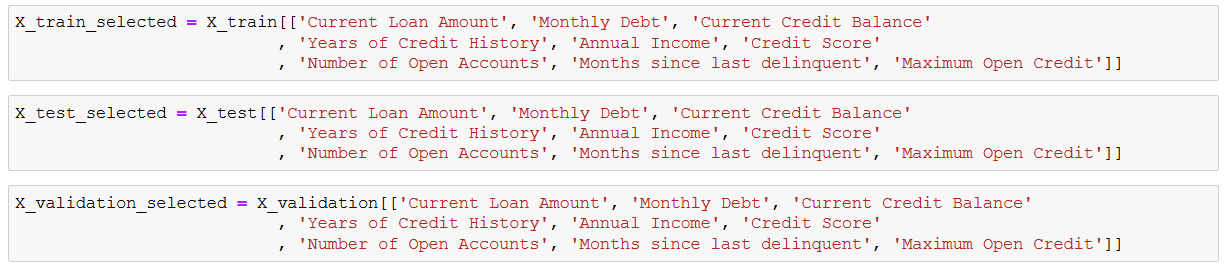
Значимы следующие признаки:



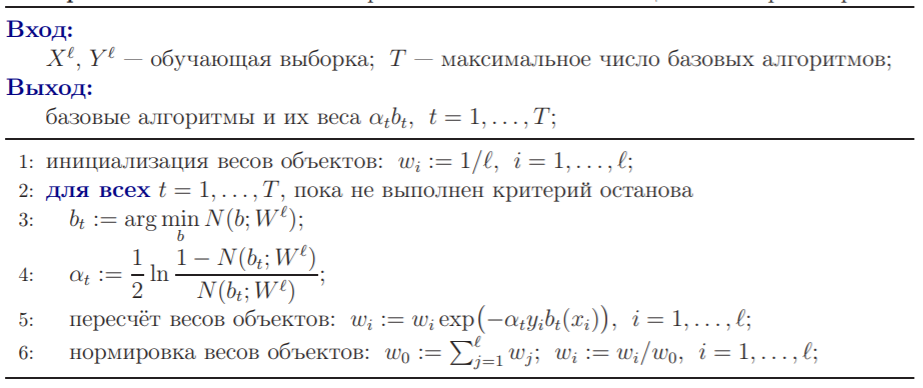
По итогам для дальнейшего анализа были выбраны следующие признаки

* Current Loan Amount
* Monthly Debt
* Current Credit Balance
* Years of Credit History
* Annual Income
* Credit Score
* Number of Open Accounts
* Months since last delinquent
* Maximum Open Credit

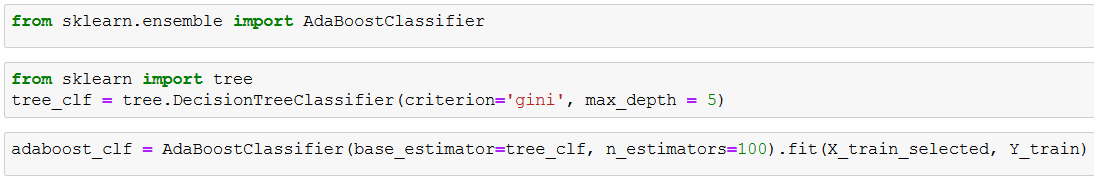
Сформируем новые обучающую, тестовую и валидационную подвыборки, оставив только выбранные признаки:



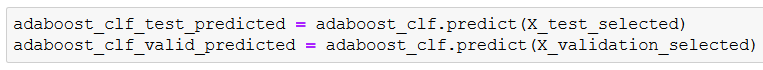
**6. AdaBoost**



Используем встроенный метод. Так же, как и бэггинг, он принимает на вход классификатор только одного вида.

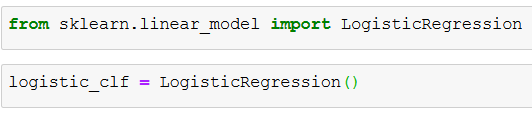


Рассчитаем ответы для тестовой и валидационной выборок.



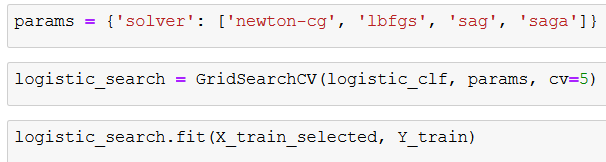
**7. Логистическая регрессия**

Используем встроенный метод

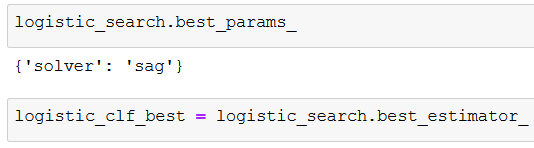


Найдем оптимальные параметры логистической регрессии

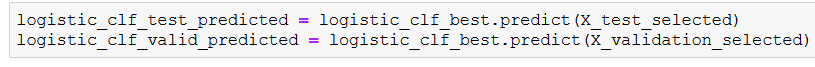
https://lh4.googleusercontent.com/tmoo9t5KM6pOgZhgRpHV_BxgT4bfekbKIevivxGQJt4g41eplvRLfev6p9c01KHhJ53RyENkMSM_qwomwHwrbyZEnZ8H2dpJkv3EEqps3YnNPSaFRBeNJ90_icBdZ3X_YguIFBcU



Задаем лучший классификатор:



Рассчитаем ответы для тестовой и валидационной выборок:



**8. Метрики качества**

## 8.1. Критерий средней ошибки на контрольных данных

Рассчитаем критерий средней ошибки для разных алгоритмов:

https://lh6.googleusercontent.com/5QTSPDuyevBhPyA5WSZ6tXSQoE_CLdUlnSAWEpLRc6KM0syUg-pH-xR3yGN5UtpMK-6o_PSho06-mRJNndipr4bRZtdHQkdz6ypLUVaIkYWV7APCv-gYtfTeX5hAx7kphZTsCI5o

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Train | Test | Validation |
| Встроенный бэггинг | 0.77292 | 0.77605 | 0.77371 |
| Разработанный бэггинг | 0.77189 | 0.77524 | 0.77279 |
| AdaBoost | 0.83147 | 0.74358 | 0.74309 |
| Логистическая регрессия | 0.77261 | 0.77587 | 0.77343 |

**8.2. Критерий «Precision»**

Рассчитаем критерий Precision для разных алгоритмов

https://lh5.googleusercontent.com/23YwxRFJ4D42aTmWXENIN1EX16hBrkb1qUvOwFMSRhmrtqocpH7SOLAfzBxfZdyjjlU2bb6bTsI1I2r2nypklSXbrm4dFuDH3906VsCP3hGD7dBc__7TQhQdeae9uXPKZ4YFLZDG

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Train | Test | Validation |
| Встроенный бэггинг | 0.99991 | 0.99988 | 0.99988 |
| Разработанный бэггинг | 0.99771 | 0.99772 | 0.99798 |
| AdaBoost | 0.96868 | 0.91506 | 0.91642 |
| Логистическая регрессия | 1.0 | 1.0 | 1.0 |

**8.3. Критерий «Recall»**

Рассчитаем критерий Recall для разных алгоритмов

https://lh3.googleusercontent.com/GpXzuVCq9Bpc0ZtO7e0LkwO1IyZ2kuZ-n5sEjpA3smnsQ6G0mMjVe3EX7M01jVBbke6pE7vwXJi1SJze3gV_BNqDnJgOtrhg8Q0jiQaYyQcCx00dZ_zUZtqzC29sLKAyP9Ph1E2W

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Train | Test | Validation |
| Встроенный бэггинг | 0.77289 | 0.77606 | 0.77369 |
| Разработанный бэггинг | 0.77189 | 0.77524 | 0.77279 |
| AdaBoost | 0.83833 | 0.78843 | 0.78663 |
| Логистическая регрессия | 0.77261 | 0.77588 | 0.77343 |

**8.5 F-мера**

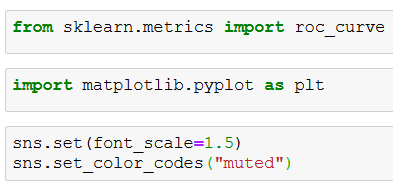
Рассчитаем F-меру для разных алгоритмов.



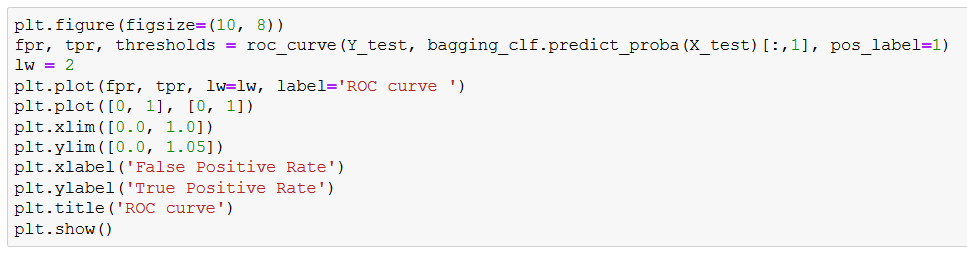
|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Train | Test | Validation |
| Встроенный бэггинг | 0.87186 | 0.87387 | 0.87237 |
| Разработанный бэггинг | 0.87028 | 0.87238 | 0.87098 |
| AdaBoost | 0.89880 | 0.84703 | 0.84658 |
| Логистическая регрессия | 0.87172 | 0.87379 | 0.87224 |

**8.5. Критерий «ROC-curve»**

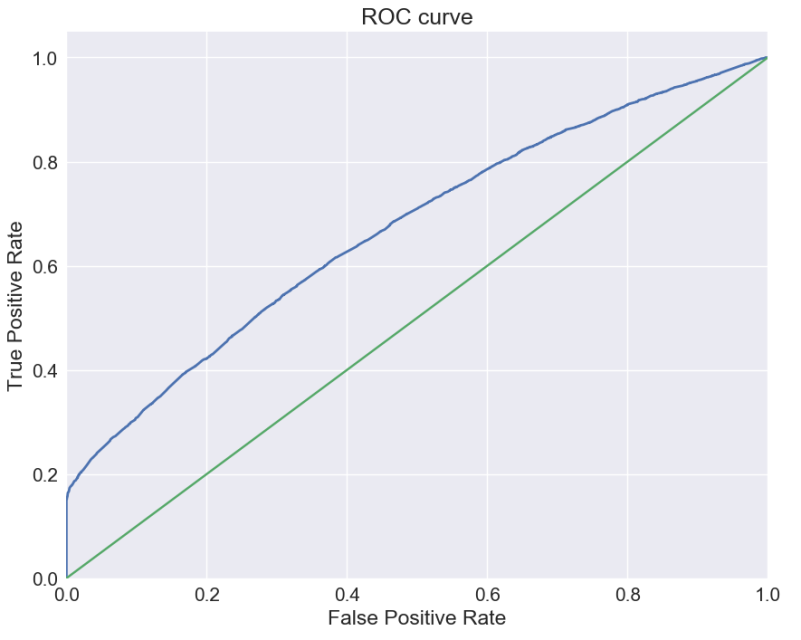
Рассчитаем критерий ROC-curve для разных алгоритмов



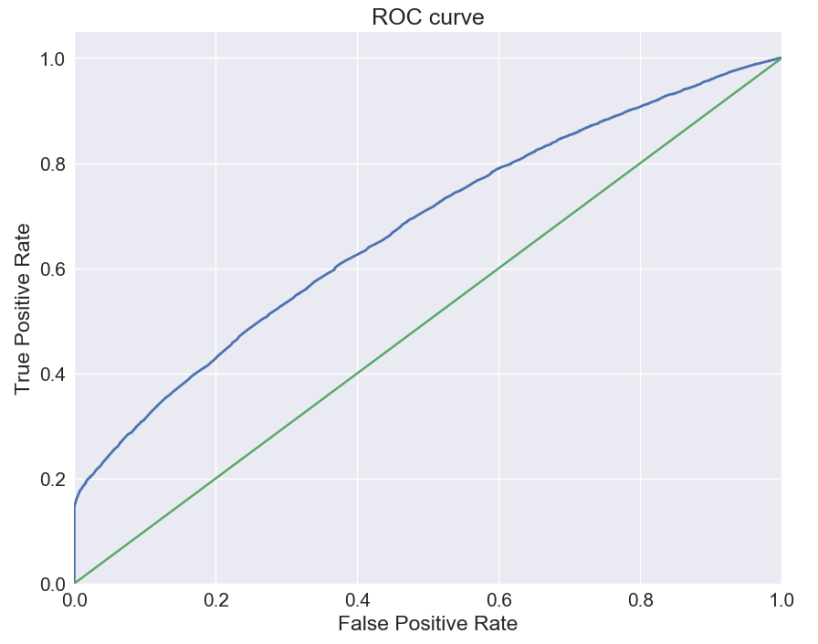
Пример построения графика



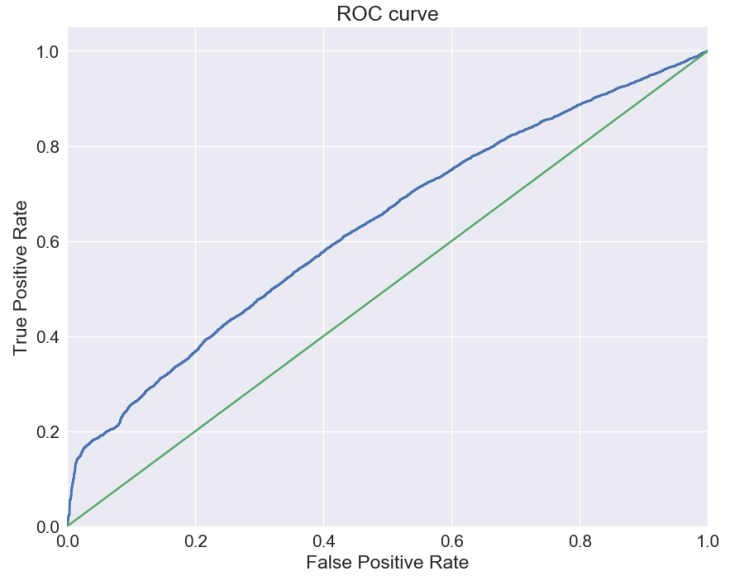
Встроенный бэггинг (тестовая выборка)



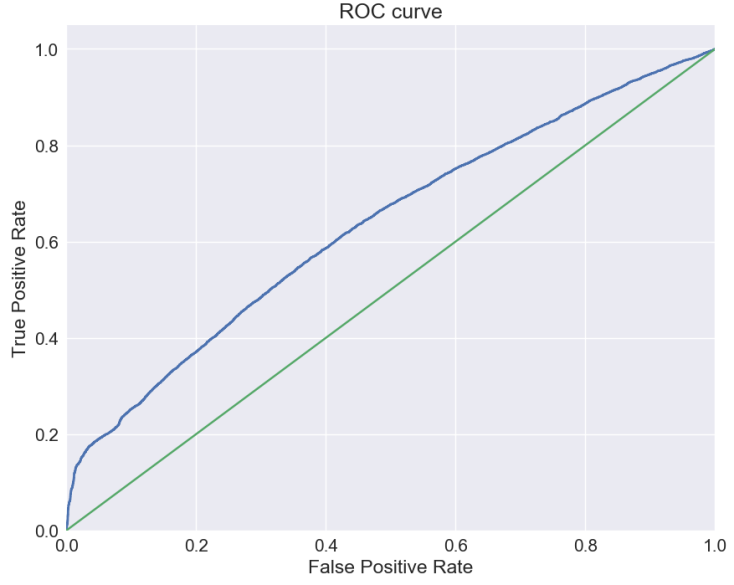
Встроенный бэггинг (валидационная выборка)



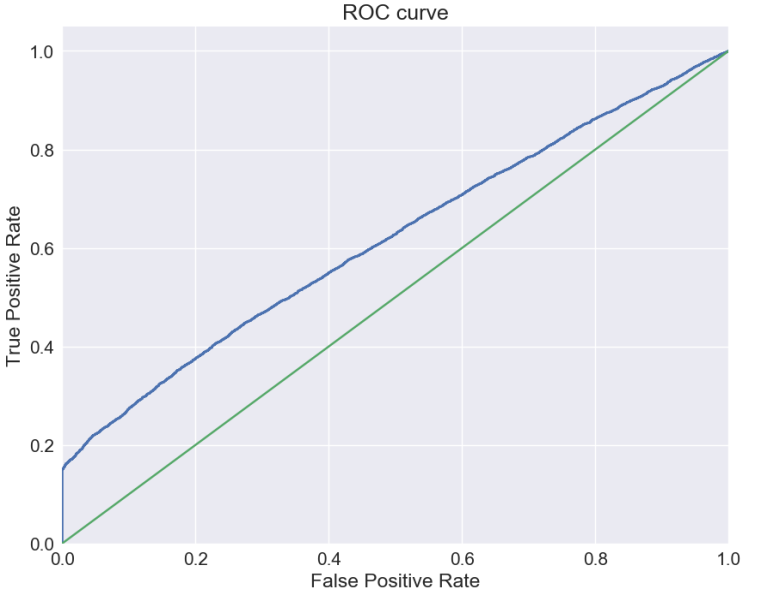
AdaBoost (тестовая выборка)



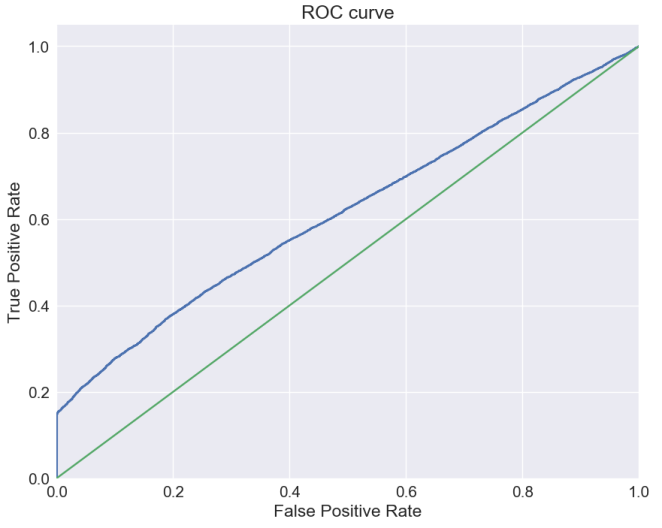
AdaBoost (валидационная выборка)



Логистическая регрессия (тестовая выборка)



Логистическая регрессия (валидационная выборка)



# Выводы

В ходе лабораторной работы было проведено исследование данных по выданным кредитам. В начале было проведено первичное исследование и очистка данных: удалены 514 пустых строк и заполнены пропущенные значения в оставшихся. Далее на данных, предварительно разбитых на обучающую и тестовую выборки, было проведено обучение алгоритмов классификации: бэггинга (с использованием встроенной функции и разработанный самостоятельно), бустинга (AdaBoost) и логистической регрессии. При этом при обучении алгоритмов бустинга и логистической регрессии требовался предварительный отбор признаков, который был проведен тремя методами. Путем анализа результатов отбора признаков было сформировано признаковое пространство. Далее для каждого из алгоритмов были посчитаны метрики качества. Лучшими алгоритмами по критерию средней ошибки на контрольных данных, критерий «Precision» являются логистическая регрессия и разработанный бэггинг. Лучшим алгоритмом по критерию «Recall» является бустинг. Лучшим алгоритмом по F-мере и критерию «ROC-curve» является встроенный бэггинг.

В качестве итогового алгоритма классификации был выбран встроенный метод Python BaggingClassifier, т.к. он дает лучшие значения метрик качества по сравнению с другими алгоритмами.

# Приложение

Исходный код

import pandas as pd

pd.set\_option('display.max\_columns', 100)

df = pd.read\_csv('credit\_train.csv')

df.head()

df.shape

df.dtypes

df.nunique()

df['Loan Status'].value\_counts()

df.corr()

**Очистка данных**

df.isnull().sum()

df[df['Loan ID'].isnull() == True]

df[df['Loan ID'].isnull() == True].isnull().sum()

df.drop(df.tail(514).index, inplace=True)

df.isnull().sum()

for i in df['Tax Liens'][df['Tax Liens'].isnull() == True].index: df.drop(labels=i, inplace=True)

for i in df['Maximum Open Credit'][df['Maximum Open Credit'].isnull() == True].index: df.drop(labels=i, inplace=True)

df.isnull().sum()

df = df.fillna({'Months since last delinquent': -1})

df['Years in current job'].unique()

import matplotlib.pyplot as plt

import seaborn as sns

get\_ipython().run\_line\_magic('matplotlib', 'inline')

plt.figure(figsize=(20,8))

plt.grid()

sns.countplot(df['Years in current job'])

df['Years in current job'].fillna('10+ years', inplace=True)

df.isnull().sum()

df['Bankruptcies'].unique()

plt.figure(figsize=(20,8))

plt.grid()

sns.countplot(df['Bankruptcies'])

df['Bankruptcies'].fillna(0, inplace=True)

df.isnull().sum()

df['Credit Score'].describe()

df[df['Credit Score']>800]

df['Credit Score'] = df['Credit Score'].apply(lambda val: (val /10) if val>850 else val)

df.head()

df['Credit Score'].describe()

df['Annual Income'].describe()

df['Credit Score'].fillna(df['Credit Score'].median(), inplace=True)

df['Annual Income'].fillna(df['Annual Income'].median(), inplace=True)

df.isnull().sum()

df.shape

df.describe()

**Формирование обучающей, тестовой и валидационной выборок**

X = df.drop(['Loan ID', 'Customer ID', 'Loan Status'], axis=1)

Y = df['Loan Status']

Y = pd.get\_dummies(Y)['Fully Paid']

X = pd.get\_dummies(X)

X = X.drop(['Purpose\_other', 'Term\_Short Term', 'Years in current job\_< 1 year', 'Home Ownership\_Rent']

, axis=1)

**Делим выборку на train, test, validation**

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

X\_wov, X\_validation, Y\_wov, Y\_validation = train\_test\_split(X, Y, test\_size=0.33, random\_state=42)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_wov, Y\_wov, test\_size=0.33, random\_state=42)

**Встроенный Bagging**

from sklearn.ensemble import BaggingClassifier

from sklearn import tree

tree\_clf = tree.DecisionTreeClassifier(criterion='gini', max\_depth = 5)

bagging\_clf = BaggingClassifier(base\_estimator=neighbors, n\_estimators=10, random\_state=0).fit(X\_train, Y\_train)

bagging\_clf.score(X\_validation, Y\_validation)

bagging\_clf = BaggingClassifier(base\_estimator=tree\_clf, n\_estimators=10, random\_state=0).fit(X\_train, Y\_train)

bagging\_clf.score(X\_validation, Y\_validation)

bagging\_clf\_test\_predicted = bagging\_clf.predict(X\_test)

bagging\_clf\_valid\_predicted = bagging\_clf.predict(X\_validation)

**Разработанный Bagging**

**Критерий регулярности - средняя ошибка на контрольных данных**

def hold\_out\_error(test, predicted):

err = 0

sq = 0

for i in range(len(test)):

err += pow(test[i] - predicted[i],2)

sq += pow(test[i], 2)

return err/sq

**Далее задаем базовые алгоритмы**

**Линейная регрессия**

def linear(xtrain, ytrain, xtest):

from sklearn import linear\_model

linear = linear\_model.LinearRegression()

linear.fit(xtrain, ytrain)

predicted = linear.predict(xtest)

for i in range(len(predicted)):

predicted[i] = round(predicted[i])

return predicted, linear

def linear\_predict(clf, validation):

predicted = clf.predict(validation)

for i in range(len(predicted)):

predicted[i] = round(predicted[i])

return predicted

**Логистическая регрессия**

def logistic(xtrain, ytrain, xtest):

from sklearn.linear\_model import LogisticRegression

logistic = LogisticRegression()

logistic.fit(xtrain, ytrain)

predicted\_uint8 = logistic.predict(xtest)

predicted = []

for i in range(len(predicted\_uint8)):

predicted\_uint8[i] = round(predicted\_uint8[i])

predicted.append(float(predicted\_uint8[i]))

return predicted, logistic

def logistic\_predict(clf, validation):

predicted\_uint8 = clf.predict(validation)

predicted = []

for i in range(len(predicted\_uint8)):

predicted\_uint8[i] = round(predicted\_uint8[i])

predicted.append(float(predicted\_uint8[i]))

return predicted

**Дерево решений**

def tree(xtrain, ytrain, xtest):

from sklearn import tree

tclf = tree.DecisionTreeClassifier(criterion='gini', max\_depth = 5)

tclf.fit(xtrain, ytrain)

predicted\_uint8 = tclf.predict(xtest)

predicted = []

for i in range(len(predicted\_uint8)):

predicted.append(float(predicted\_uint8[i]))

return predicted, tclf

def tree\_predict(clf, validation):

predicted\_uint8 = clf.predict(validation)

predicted = []

for i in range(len(predicted\_uint8)):

predicted.append(float(predicted\_uint8[i]))

return predicted

**k ближайших соседей**

def k\_neighbors(xtrain, ytrain, xtest):

from sklearn.neighbors import KNeighborsClassifier

neighbors = KNeighborsClassifier(n\_neighbors=5)

neighbors.fit(xtrain, ytrain)

predicted\_uint8 = neighbors.predict(xtest)

predicted = []

for i in range(len(predicted\_uint8)):

predicted.append(float(predicted\_uint8[i]))

return predicted, neighbors

def k\_neighbors\_predict(clf, validation):

predicted\_uint8 = clf.predict(validation)

predicted = []

for i in range(len(predicted\_uint8)):

predicted.append(float(predicted\_uint8[i]))

return predicted

**Многослойный персептрон**

def MLP(xtrain, ytrain, xtest):

from sklearn.neural\_network import MLPClassifier

clf = MLPClassifier(max\_iter=100)

clf.fit(xtrain, ytrain)

predicted\_uint8 = clf.predict(xtest)

predicted = []

for i in range(len(predicted\_uint8)):

predicted.append(float(predicted\_uint8[i]))

return predicted, clf

def MLP\_predict(clf, validation):

predicted\_uint8 = clf.predict(validation)

predicted = []

for i in range(len(predicted\_uint8)):

predicted.append(float(predicted\_uint8[i]))

return predicted

**Bagging смешиваем с RSM, т.к. у нас много лишних признаков**

**Метод случайных подпространств - формирование подпространства**

**n - длина признакового подописания**

**l - длина выборки**

def create\_subspace(xtrain, ytrain, xtest, l, n):

import random

#отбор признаков

sheme = [0]\*len(xtrain.columns)

while (sum(sheme)<n):

i = random.randint(0,len(xtrain.columns)-1)

sheme[i] = 1

cols = []

for i in range(len(xtrain.columns)):

if sheme[i] == 1:

cols.append(xtrain.columns[i])

X\_train\_RSM\_all\_rows = xtrain[cols]

#отбор объектов

X\_train\_RSM = pd.DataFrame(columns=cols)

Y\_train\_RSM = pd.Series()

for i in range(l):

j = random.randint(0, X\_train\_RSM\_all\_rows.shape[0] - 1)

X\_train\_RSM = X\_train\_RSM.append(X\_train\_RSM\_all\_rows.iloc[j])

Y\_train\_RSM = Y\_train\_RSM.append(pd.Series(ytrain.iloc[j]),ignore\_index=True)

#отбор признаков тестовой выборки

X\_test\_RSM = xtest[cols]

return X\_train\_RSM, Y\_train\_RSM, X\_test\_RSM, cols

X\_train\_RSM, Y\_train\_RSM, X\_test\_RSM, cols = create\_subspace(X\_train, Y\_train, X\_test, 1000, 10)

basic\_algorithms\_train = {'linear': linear, 'logistic': logistic,

'tree': tree, 'k\_neighbors': k\_neighbors,

'MLP': MLP}

basic\_algorithms\_test = {'linear': linear\_predict, 'logistic': logistic\_predict,

'tree': tree\_predict, 'k\_neighbors': k\_neighbors\_predict,

'MLP': MLP\_predict}

def bagging\_fit(xtrain, ytrain, xtest, ytest, ba\_train, ba\_test, l, n, eps1, eps2):

good\_clfs = {}

for key in ba\_train:

X\_train\_RSM, Y\_train\_RSM, X\_test\_RSM, cols = create\_subspace(xtrain, ytrain, xtest, l, n)

Y\_pred, clf = ba\_train[key](X\_train\_RSM, Y\_train\_RSM, X\_test\_RSM)

q2 = hold\_out\_error(ytest.values, Y\_pred)

y\_train\_pred = ba\_test[key](clf, X\_train\_RSM)

q1 = hold\_out\_error(Y\_train\_RSM, y\_train\_pred)

if (q1<=eps1) and (q2<=eps2):

good\_clfs[key] = [clf, cols]

return good\_clfs

def bagging\_predict(compos, ba\_test, validation):

y\_validation\_pred = []

for key in compos:

y\_key\_pred = ba\_test[key](compos[key][0],validation[compos[key][1]])

y\_validation\_pred.append(y\_key\_pred)

y\_validation\_final = []

for i in range(len(y\_validation\_pred[0])):

s = 0

for j in range(len(y\_validation\_pred)):

s += y\_validation\_pred[j][i]

answer = round(s/len(y\_validation\_pred))

y\_validation\_final.append(answer)

return y\_validation\_final

composition = bagging\_fit(X\_train, Y\_train, X\_test, Y\_test, basic\_algorithms\_train, basic\_algorithms\_test

, 1000, 10, 0.3, 0.31)

Y\_bagging\_test\_predict = bagging\_predict(composition, basic\_algorithms\_test, X\_test)

Y\_bagging\_validation\_predict = bagging\_predict(composition, basic\_algorithms\_test, X\_validation)

hold\_out\_error(Y\_validation.values, Y\_bagging\_validation\_predict)

**Отбор признаков**

import feature\_selector

fs = feature\_selector.FeatureSelector(data=X\_train, labels=Y\_train)

fs.identify\_all(selection\_params = {'missing\_threshold': 0.6, 'correlation\_threshold': 0.98,

'task': 'classification', 'eval\_metric': 'auc',

'cumulative\_importance': 0.99})

fs.plot\_collinear(plot\_all=True)

fs.identify\_collinear(correlation\_threshold=0.3)

fs.plot\_collinear()

fs.plot\_feature\_importances(threshold = 0.99, plot\_n = 30)

low\_importance\_features = fs.ops['low\_importance']

low\_importance\_features

**Попробуем сделать то же самое с помощью RandomForest**

import numpy as np

from sklearn.ensemble import RandomForestClassifier

np.random.seed(0)

clf\_rf = RandomForestClassifier(criterion='entropy')

clf\_rf.fit(X\_train, Y\_train)

imp = pd.DataFrame(clf\_rf.feature\_importances\_, index=X\_train.columns, columns=['importance'])

imp.sort\_values('importance').plot(kind='barh', figsize=(12, 8))

imp.sort\_values('importance', ascending=False)

**Отбор признаков через логистическую регрессию**

from sklearn.feature\_selection import RFE

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

rfe = RFE(model, 9)

fit = rfe.fit(X\_train, Y\_train)

for i in range(len(X\_train.columns)):

if fit.support\_[i]:

print(X\_train.columns[i])

**Сформируем новые обучающую, тестовую и валидационную выборки, оставив только необходимые признаки.**

X\_train\_selected = X\_train[['Current Loan Amount', 'Monthly Debt', 'Current Credit Balance'

, 'Years of Credit History', 'Annual Income', 'Credit Score'

, 'Number of Open Accounts', 'Months since last delinquent', 'Maximum Open Credit']]

X\_test\_selected = X\_test[['Current Loan Amount', 'Monthly Debt', 'Current Credit Balance'

, 'Years of Credit History', 'Annual Income', 'Credit Score'

, 'Number of Open Accounts', 'Months since last delinquent', 'Maximum Open Credit']]

X\_validation\_selected = X\_validation[['Current Loan Amount', 'Monthly Debt', 'Current Credit Balance'

, 'Years of Credit History', 'Annual Income', 'Credit Score'

, 'Number of Open Accounts', 'Months since last delinquent', 'Maximum Open Credit']]

**AdaBoost**

from sklearn.ensemble import AdaBoostClassifier

from sklearn import tree

tree\_clf = tree.DecisionTreeClassifier(criterion='gini', max\_depth = 5)

adaboost\_clf = AdaBoostClassifier(base\_estimator=tree\_clf, n\_estimators=100).fit(X\_train\_selected, Y\_train)

adaboost\_clf.score(X\_validation\_selected, Y\_validation)

adaboost\_clf\_test\_predicted = adaboost\_clf.predict(X\_test\_selected)

adaboost\_clf\_valid\_predicted = adaboost\_clf.predict(X\_validation\_selected)

**Логистическая регрессия**

from sklearn.linear\_model import LogisticRegression

logistic\_clf = LogisticRegression()

Подберем оптимальные параметры с помощью gridSearchCV

from sklearn.model\_selection import GridSearchCV

params = {'solver': ['newton-cg', 'lbfgs', 'sag', 'saga']}

logistic\_search = GridSearchCV(logistic\_clf, params, cv=5)

logistic\_search.fit(X\_train\_selected, Y\_train)

logistic\_search.best\_params\_

logistic\_clf\_best = logistic\_search.best\_estimator\_

logistic\_clf\_best.score(X\_validation\_selected, Y\_validation)

logistic\_clf\_test\_predicted = logistic\_clf\_best.predict(X\_test\_selected)

logistic\_clf\_valid\_predicted = logistic\_clf\_best.predict(X\_validation\_selected)

**Метрики качества**

**Критерий средней ошибки на контрольных данных**

from sklearn.metrics import accuracy\_score

accuracy\_score(bagging\_clf.predict(X\_train), Y\_train)

accuracy\_score(bagging\_clf\_test\_predicted, Y\_test)

accuracy\_score(bagging\_clf\_valid\_predicted, Y\_validation)

accuracy\_score(bagging\_predict(composition, basic\_algorithms\_test, X\_train), Y\_train)

accuracy\_score(Y\_bagging\_test\_predict, Y\_test)

accuracy\_score(Y\_bagging\_validation\_predict, Y\_validation)

accuracy\_score(adaboost\_clf.predict(X\_train\_selected), Y\_train)

accuracy\_score(adaboost\_clf\_test\_predicted, Y\_test)

accuracy\_score(adaboost\_clf\_valid\_predicted, Y\_validation)

accuracy\_score(logistic\_clf\_best.predict(X\_train\_selected), Y\_train)

accuracy\_score(logistic\_clf\_test\_predicted, Y\_test)

accuracy\_score(logistic\_clf\_valid\_predicted, Y\_validation)

**Precision**

from sklearn.metrics import precision\_score

precision\_score(bagging\_clf.predict(X\_train), Y\_train, average='binary')

precision\_score(bagging\_clf\_test\_predicted, Y\_test, average='binary')

precision\_score(bagging\_clf\_valid\_predicted, Y\_validation, average='binary')

precision\_score(bagging\_predict(composition, basic\_algorithms\_test, X\_train), Y\_train, average='weighted')

precision\_score(Y\_bagging\_test\_predict, Y\_test, average='weighted')

precision\_score(Y\_bagging\_validation\_predict, Y\_validation, average='weighted')

precision\_score(adaboost\_clf.predict(X\_train\_selected), Y\_train, average='binary')

precision\_score(adaboost\_clf\_test\_predicted, Y\_test, average='binary')

precision\_score(adaboost\_clf\_valid\_predicted, Y\_validation, average='binary')

precision\_score(logistic\_clf\_best.predict(X\_train\_selected), Y\_train, average='binary')

precision\_score(logistic\_clf\_test\_predicted, Y\_test, average='binary')

precision\_score(logistic\_clf\_valid\_predicted, Y\_validation, average='binary')

**Recall**

from sklearn.metrics import recall\_score

recall\_score(bagging\_clf.predict(X\_train), Y\_train, average='binary')

recall\_score(bagging\_clf\_test\_predicted, Y\_test, average='binary')

recall\_score(bagging\_clf\_valid\_predicted, Y\_validation, average='binary')

recall\_score(bagging\_predict(composition, basic\_algorithms\_test, X\_train), Y\_train, average='weighted')

recall\_score(Y\_bagging\_test\_predict, Y\_test, average='weighted')

recall\_score(Y\_bagging\_validation\_predict, Y\_validation, average='weighted')

recall\_score(adaboost\_clf.predict(X\_train\_selected), Y\_train, average='binary')

recall\_score(adaboost\_clf\_test\_predicted, Y\_test, average='binary')

recall\_score(adaboost\_clf\_valid\_predicted, Y\_validation, average='binary')

recall\_score(logistic\_clf\_best.predict(X\_train\_selected), Y\_train, average='binary')

recall\_score(logistic\_clf\_test\_predicted, Y\_test, average='binary')

recall\_score(logistic\_clf\_valid\_predicted, Y\_validation, average='binary')

**F-мера**

from sklearn.metrics import f1\_score

f1\_score(bagging\_clf.predict(X\_train), Y\_train, average='binary')

f1\_score(bagging\_clf\_test\_predicted, Y\_test, average='binary')

f1\_score(bagging\_clf\_valid\_predicted, Y\_validation, average='binary')

f1\_score(bagging\_predict(composition, basic\_algorithms\_test, X\_train), Y\_train, average='weighted')

f1\_score(Y\_bagging\_test\_predict, Y\_test, average='weighted')

f1\_score(Y\_bagging\_validation\_predict, Y\_validation, average='weighted')

f1\_score(adaboost\_clf.predict(X\_train\_selected), Y\_train, average='binary')

f1\_score(adaboost\_clf\_test\_predicted, Y\_test, average='binary')

f1\_score(adaboost\_clf\_valid\_predicted, Y\_validation, average='binary')

f1\_score(logistic\_clf\_best.predict(X\_train\_selected), Y\_train, average='binary')

f1\_score(logistic\_clf\_test\_predicted, Y\_test, average='binary')

f1\_score(logistic\_clf\_valid\_predicted, Y\_validation, average='binary')

**Коэффициент корреляции Мэтьюса**

from sklearn.metrics import matthews\_corrcoef

matthews\_corrcoef(bagging\_clf.predict(X\_train), Y\_train)

matthews\_corrcoef(bagging\_clf\_test\_predicted, Y\_test)

matthews\_corrcoef(bagging\_clf\_valid\_predicted, Y\_validation)

matthews\_corrcoef(bagging\_predict(composition, basic\_algorithms\_test, X\_train), Y\_train)

matthews\_corrcoef(Y\_bagging\_test\_predict, Y\_test)

matthews\_corrcoef(Y\_bagging\_validation\_predict, Y\_validation)

matthews\_corrcoef(adaboost\_clf.predict(X\_train\_selected), Y\_train)

matthews\_corrcoef(adaboost\_clf\_test\_predicted, Y\_test)

matthews\_corrcoef(adaboost\_clf\_valid\_predicted, Y\_validation)

matthews\_corrcoef(logistic\_clf\_best.predict(X\_train\_selected), Y\_train)

matthews\_corrcoef(logistic\_clf\_test\_predicted, Y\_test)

matthews\_corrcoef(logistic\_clf\_valid\_predicted, Y\_validation)

**ROC-curve**

from sklearn.metrics import roc\_curve

import matplotlib.pyplot as plt

sns.set(font\_scale=1.5)

sns.set\_color\_codes("muted")

plt.figure(figsize=(10, 8))

fpr, tpr, thresholds = roc\_curve(Y\_test, bagging\_clf.predict\_proba(X\_test)[:,1], pos\_label=1)

lw = 2

plt.plot(fpr, tpr, lw=lw, label='ROC curve ')

plt.plot([0, 1], [0, 1])

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC curve')

plt.show()

plt.figure(figsize=(10, 8))

fpr, tpr, thresholds = roc\_curve(Y\_validation, bagging\_clf.predict\_proba(X\_validation)[:,1], pos\_label=1)

lw = 2

plt.plot(fpr, tpr, lw=lw, label='ROC curve ')

plt.plot([0, 1], [0, 1])

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC curve')

plt.show()

plt.figure(figsize=(10, 8))

fpr, tpr, thresholds = roc\_curve(Y\_test, adaboost\_clf.predict\_proba(X\_test\_selected)[:,1], pos\_label=1)

lw = 2

plt.plot(fpr, tpr, lw=lw, label='ROC curve ')

plt.plot([0, 1], [0, 1])

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC curve')

plt.show()

plt.figure(figsize=(10, 8))

fpr, tpr, thresholds = roc\_curve(Y\_validation, adaboost\_clf.predict\_proba(X\_validation\_selected)[:,1], pos\_label=1)

lw = 2

plt.plot(fpr, tpr, lw=lw, label='ROC curve ')

plt.plot([0, 1], [0, 1])

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC curve')

plt.show()

plt.figure(figsize=(10, 8))

fpr, tpr, thresholds = roc\_curve(Y\_test, logistic\_clf\_best.predict\_proba(X\_test\_selected)[:,1], pos\_label=1)

lw = 2

plt.plot(fpr, tpr, lw=lw, label='ROC curve ')

plt.plot([0, 1], [0, 1])

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC curve')

plt.show()

plt.figure(figsize=(10, 8))

fpr, tpr, thresholds = roc\_curve(Y\_validation, logistic\_clf\_best.predict\_proba(X\_validation\_selected)[:,1], pos\_label=1)

lw = 2

plt.plot(fpr, tpr, lw=lw, label='ROC curve ')

plt.plot([0, 1], [0, 1])

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC curve')

plt.show()