Московский государственный университет имени М. В. Ломоносова.

Градиентные методы обучения линейных моделей.

Чванкина Дарья
Октябрь 2021
Отчет по курс "Практикум на ЭВМ 2021/2022"

1 Введение

В данной работе исследуются градиентные методы обучения логистической регрессии. В экспериментах используются собственные реализации градиентного спуска и стохастического градиентного спуска. Градиентные методы обучения сравниваются на датасете, содержащим комментарии из раздела обсуждений английской Википедии. Решалась задача бинарной классификации: является ли данный комментарий токсичным или нет.

2 Постановка задачи

2.1 Вывод формулы градиента функции потерь для задачи бинарной логистической регрессии

Пусть дана обучающая выборка $X = (x_i, y_i)_{i=1}^l$, где $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{Y} = \{-1, 1\}$. Тогда линейная модель бинарной классификации определяется следующим образом:

$$a(x) = \text{sign}(\langle w, x \rangle).$$

Функция потерь равна:

$$\mathcal{L}(a, y_i) = [a(x_i)y_i < 0] = [\langle w, x_i \rangle y_i < 0].$$

Заменим её на непрерывную функцию, убывающую с возрастанием величины $M_i = \langle w, x_i \rangle y_i$, которая называется отступом. Чем больше отступ, тем дальше находится объект от разделяющей плоскости. Если $M_i < 0$, то наша модель неправильно классифицировала объект, если $M_i > 0$, то объект классифицирован правильно. Модель, использующая логистическую функцию потерь, называется логистической регрессией. Таким образом

$$\mathcal{L}(a, y_i) = \log(1 + \exp(-M_i)). \tag{1}$$

Тогда во время обучения минимизируется функция:

$$Q(X, w) = \frac{1}{l} \sum_{i=1}^{l} \log(1 + \exp(-\langle w, x_i \rangle y_i)) \to \min_w$$

Минимизация функционала логистической регрессии эквивалентна максимизации логарифма функции правдоподобия, то есть с помощью этого алгоритма можно оценить вероятность принадлежности объекта одному из классов:

$$p(y = +1|x) = \frac{1}{1 + \exp(-\langle w, x \rangle)} = \sigma(M)$$

В алгоритме градиентного спуска веса изменяются в направлении убывания градиента функционала. Найдем градиент функции потерь.

$$\frac{dQ(X, w)}{dw} = \frac{1}{l} \sum_{i=1}^{l} \frac{\exp(-\langle w, x_i \rangle y_i)}{1 + \exp(-\langle w, x_i \rangle y_i)} (-\langle dw, x_i \rangle y_i)$$

$$\nabla_w Q = \frac{1}{l} \sum_{i=1}^{l} \frac{\exp(-\langle w, x_i \rangle y_i)}{1 + \exp(-\langle w, x_i \rangle y_i)} (-x_i y_i)$$

2.2 Вывод формулы градиента функции потерь для задачи мультиномиальной логистической регрессии

Пусть $\mathbb{Y} = \{1, \ldots, K\}$. Сведем задачу многоклассовой классификации к бинарной. Построим K разделяющих плоскостей, то есть обучим C_K^2 линейных моделей $a_{sj}(x), s, j \in \{1, \ldots, K\}$, $s < j$. Каждая модель строит плоскость между двумя классами s и j. Также сведем задачу так, чтобы она могла оценивать вероятность принадлежности классу. Вероятность принадлежности классу k равна

$$p(y = k|x) = \frac{\exp(\langle w_k, x\rangle)}{\sum_{i=1}^{K} \exp(\langle w_i, x\rangle)}.$$

Тогда во время обучения минимизируем минус логарифм правдоподобия:

$$Q = -\frac{1}{l}\sum_{i=1}^{l} \log \frac{\exp(\langle w_{y_i}, x_i\rangle)}{\sum_{k=1}^{K} \exp(\langle w_k, x_i\rangle)} = -\frac{1}{l}\sum_{i=1}^{l} \big(\langle w_{y_i}, x_i\rangle - \log \sum_{k=1}^{K} \exp(\langle w_k, x_i\rangle\big) \to \min_{w_1,\ldots,w_k}$$

$$\frac{dQ}{dw_j} = -\frac{1}{l}\sum_{i=1}^{l} \big(\langle dw_{y_i}, x_i\rangle - \frac{1}{\sum_{k=1}^{K} \exp(\langle w_k, x_i\rangle)} \exp(\langle w_j, x_i\rangle)\langle dw_j, x_i\rangle\big)$$

$$\nabla_{w_j} Q = \frac{1}{l}\sum_{i=1}^{l} \big(\frac{\exp\langle w_j, x_i\rangle}{\sum_{k=1}^{K} \exp(\langle w_k, x_i\rangle)} - [y_i = j]\big)x_i$$

При k = 2:

$$Q = -\frac{1}{l}\sum_{i=1}^{l}([y_i = 1]\log\frac{\exp(\langle w_1, x_i\rangle)}{\exp(\langle w_1, x_i\rangle) + \exp(\langle w_2, x_i\rangle)} + [y_i = -1]\log\frac{\exp(\langle w_2, x_i\rangle)}{\exp(\langle w_1, x_i\rangle) + \exp(\langle w_2, x_i\rangle)})$$

Обозначим $w = w_1 - w_2$.

$$Q = -\frac{1}{l}\sum_{i=1}^{l}([y_i = 1]\log\frac{\exp(\langle w, x_i\rangle)}{1 + \exp(\langle w, x_i\rangle)} + [y_i = -1]\log\frac{\exp(\langle w, x_i\rangle)}{1 + \exp(\langle w, x_i\rangle)})$$

$$Q = \frac{1}{l}\sum_{i=1}^{l}(-[y_i = 1]\log\frac{1)}{\exp(\langle -w, x_i\rangle) + 1} - [y_i = -1]\log\frac{1}{\exp(\langle -w, x_i\rangle) + 1})$$

$$Q = \frac{1}{l}\sum_{i=1}^{l} \log(1 + \exp(-\langle w, x_i\rangle y_i))$$

Таким образом задача мультиномиальной регрессии сводится к логистической.

3 Эксперимент № 1

Для дальнейших исследований данные были предобработаны следующим образом. Все комментарии были переведены к нижнему регистру, а все символы, не являющейся буквами и цифрами, были заменены на пробелы. Каждый комментарий был представлен в виде вектора, с помощью конструктора CountVectorizer из библиотеки sklearn, который использует представление BagOfWords. Данная векторизация представляет каждый документ как мешок слов и выдаёт вектор, составленный на основе частоты вхождения слов в документ.

В градиентном спуске веса изменяются по следующей формуле:

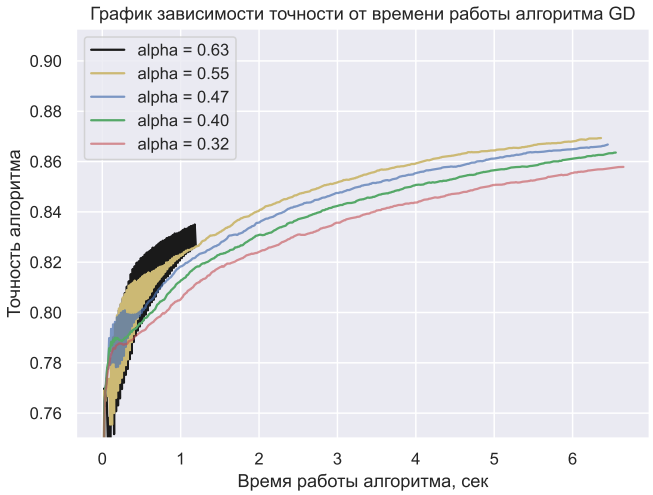
$$w^{(t+1)} := w^{(t)} - \eta_t \nabla_w Q(X, w),$$

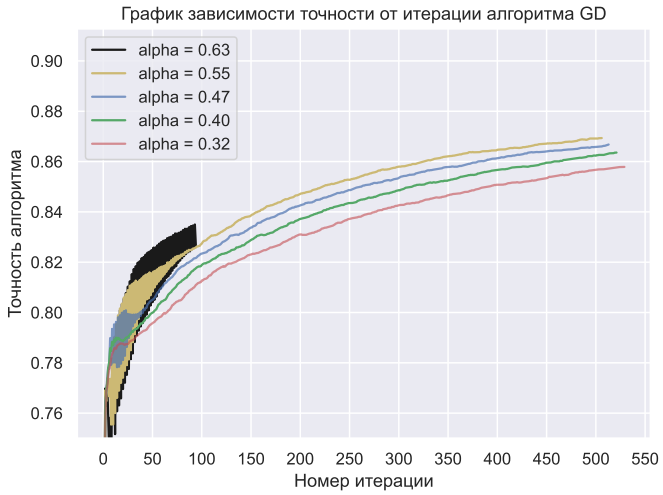
где $\eta_t = \frac{\alpha}{t^\beta}$, $\alpha, \beta -$ const. Исследование поведение модели в зависимости от параметров α и β проводилось при следующих параметрах модели:

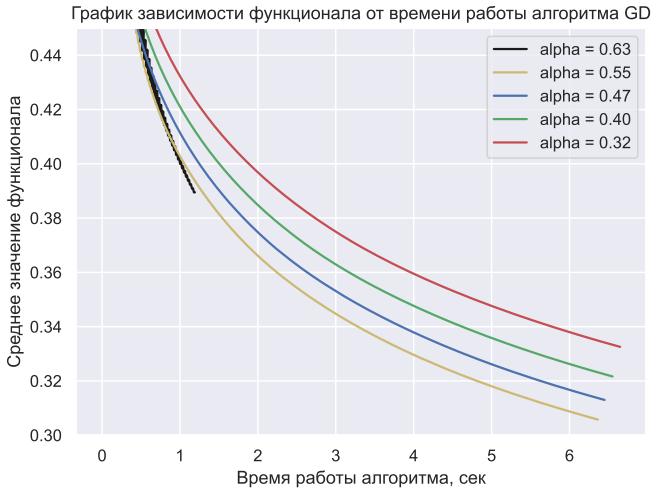
- min_df = 0.001 - параметр конструктора векторизации (для уменьшения признакого пространства)
- tolerance = 1e-4 - параметр задающий критерий останова алгоритма
- max_iter = 1000 - максимальное число итераций
- l2_coef = 0 - коэффициент L_2 регуляризации
- w_0 = [0, ..., 0] - начальное приближение вектора весов.

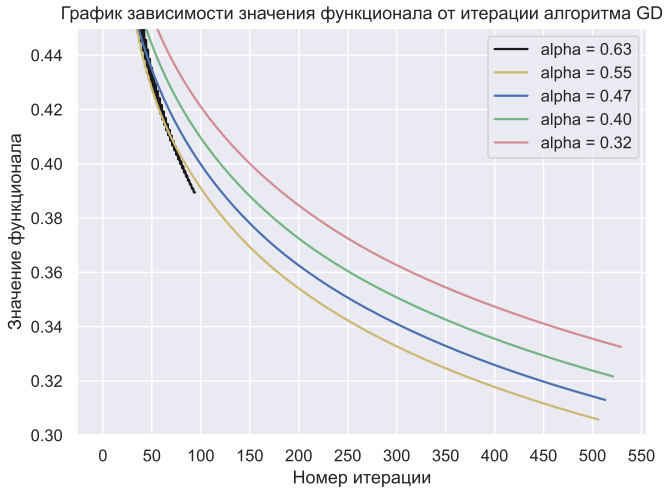
3.1 Исследование параметра α

Зафиксируем параметр β равным 0. Графики зависимостей от итерации и от времени ничем не отличаются, так как время замерялось каждую итерацию. Можно заметить, что алгоритм сходится при $\alpha < 0.55$. Чем выше параметр, тем больше точность на валидационной выборке и тем быстрее алгоритм сходится. Поэтому этот параметр надо выбирать максимальным из гарантированно сходящихся.



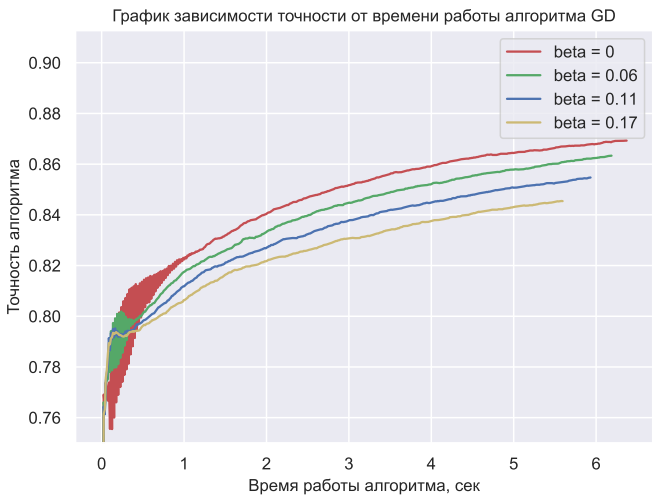


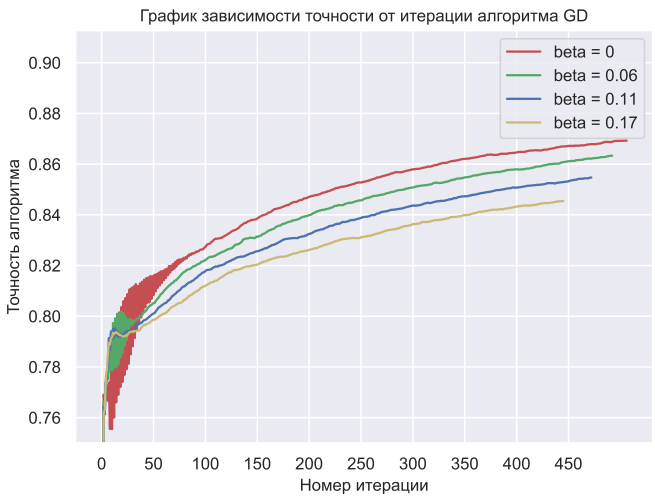




3.2 Исследование параметра β

Зафиксируем параметр альфа равным 0.55 и будем изменять бетта. В данном случае параметр $\beta = 0$ показывает наилучший результат, а с ростом значения параметра, точность алгоритма падает. Но также при увеличении параметра β алгоритм быстрее, а график точности (соответственно и функционала) меньше колеблется в начале времени работы алгоритма. Можно предположить, что при использовании этого параметра, алгоритм будет сходиться при больших значений параметра α.





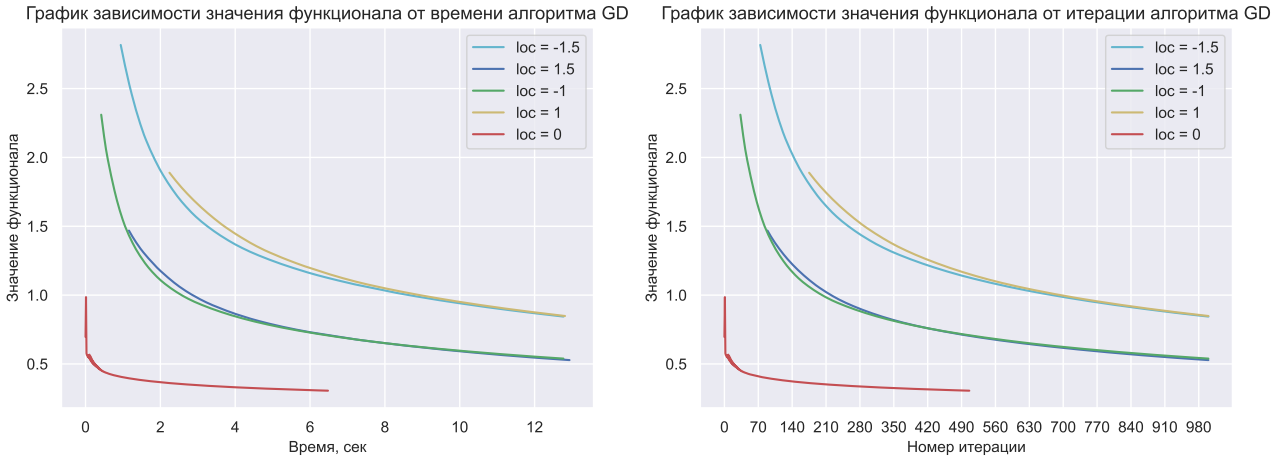
Для высокого результата точности лучше подбирать параметры шага одновременно.

Графики точности и функционала очень похожи между собой.(Графики функционала см. в Приложении Рис. 1) Скорость убывания функционала равна скорости возрастания точности, что подтверждает вероятностную интерпретацию алгоритма.

3.3 Исследование начального приближения

Для исследования зависимости от начального приближение попробуем начать сходиться из разных точек. Для этого вектор весов генерировался из нормального распределения с разным математическим ожиданием и небольшой дисперсией равной 0.1. Математические

ожидания были равны 0, +1, -1, +1.5, -1.5 Как видно из графиков для данного алгоритма знак математического ожидания не важен. То есть для вектора весов с математическим ожиданием +1 алгоритм стартует из той же точки, что и алгоритм с вектором весов из распределения с математическим ожиданием -1. Причём алгоритм при ненулевом математическом ожидании сходится к локальным минимумам, не являющимся оптимумами. Наименьшее значение функционала даёт вектор с нулевым мат.(Графики функционала см. в Приложении Рис. 2) ожиданием, причём время работы алгоритма меньше, чем с другими начальными приближениями.



4 Эксперимент № 2

На каждой итерации градиентного спуска расчет градиента производился на всей выборке. Алгоритм точно рассчитывает градиент, но это занимает много времени. Для ускорения вместо подсчета точного значения градиента можно считать приближенное значение, производя расчет на подвыборке. Сначала перемешиваются все объекты в выборке, далее на каждой итерации выбирается некоторая подвыборка размером batch_size, по ней вычисляется градиент, на следующей итерации выбирается следующая подвыборка размером batch_size и так далее. Если элементы закончились, то начальная выборка перемешивается, и процесс повторяется.

Так как основная цель - ускорить работу градиентного спуска, то для отслеживания динамики функционала также не используется вся выборка. Расчет функционала производится с помощью экспоненциального сглаживания:

$$Q^0 := Q;$$

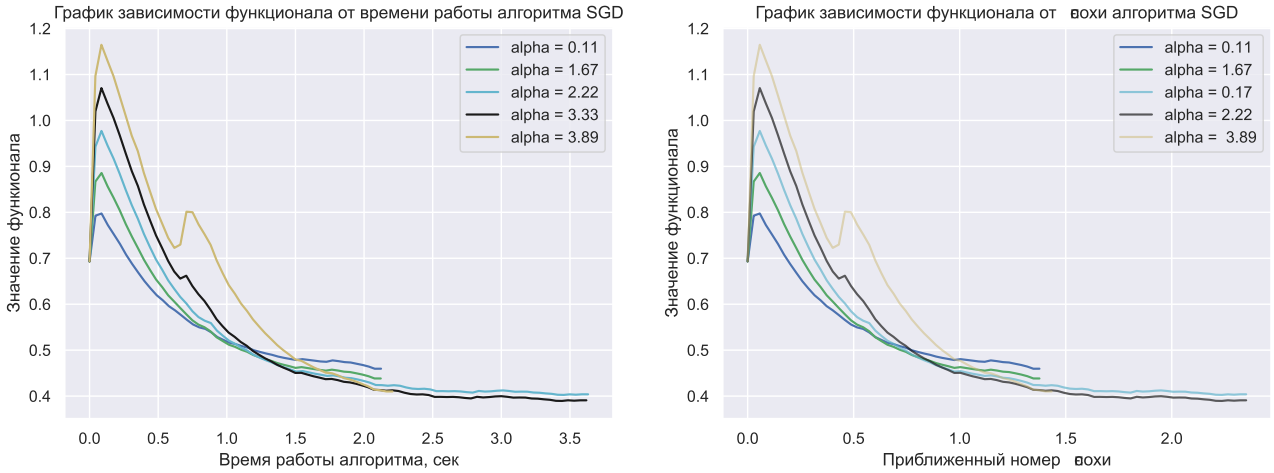
$$Q^{t+1} := (1 - \lambda)Q^t + \lambda\epsilon_t,$$

где ϵ_t - значение функционала на подвыборке, которая используется на данной итерации. В данных экспериментах $\lambda = 0.1$.

В этом разделе приведено исследование стохастического градиентного спуска (SGD) при следующих параметрах:

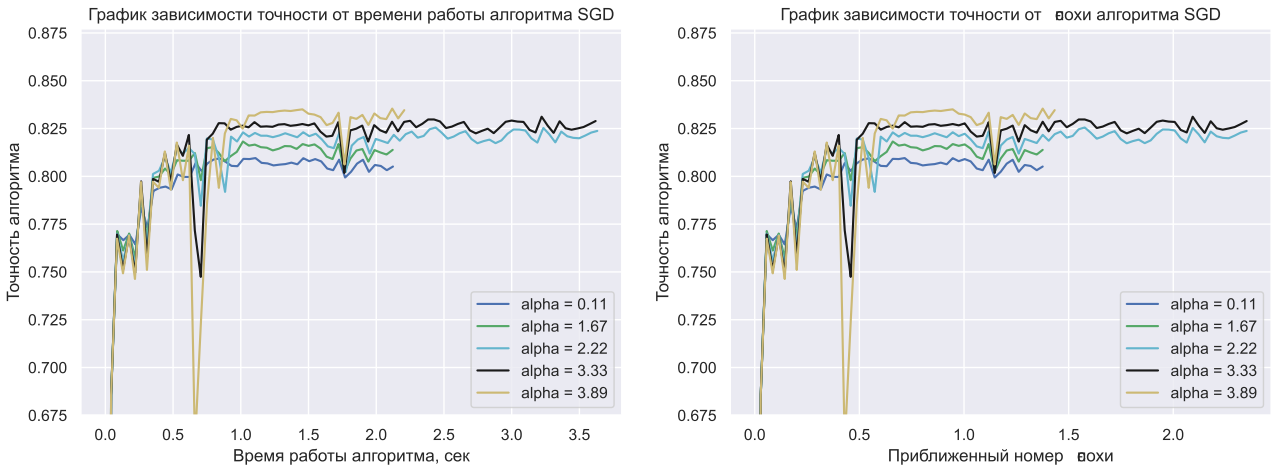
- tolerance = 1e-4 - параметр задающий критерий останова алгоритма

- max_iter = 1000 - максимальное число итераций
- l2_coef = 0 - коэффициент L_2 регуляризации
- w_0 = [0, ..., 0] - начальное приближение вектора весов.



4.1 Исследование параметра α

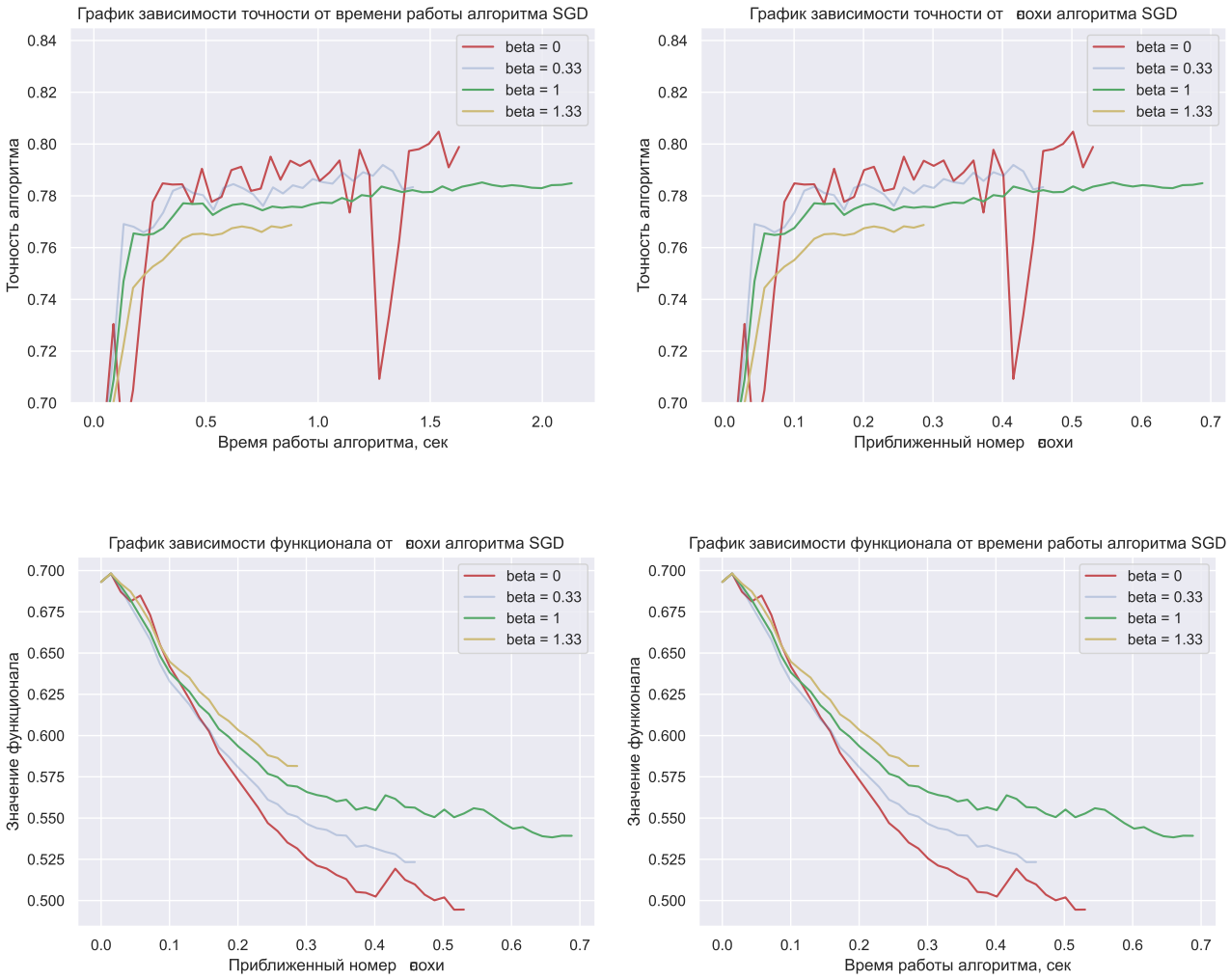
Пусть параметр $\beta = 0.5$ (при $\beta = 0$ алгоритм не сходился), batch_size = 1000. При $\alpha < 2$ алгоритм быстро сходился, но алгоритм даёт точность меньше 0.82. При $\alpha > 3.33$ алгоритм быстро сходится, даёт наибольшую точность, но не наименьший функционал. Так как модель может переобучаться в данном случае наилучший параметр равен 3.33. Можно заметить, что чем больше параметр α, тем больше точность, но графики функционала имеют больше колебаний, чем при меньших значений параметра.



4.2 Исследование параметра β

Пусть $\alpha = 0.33$. При больших значениях параметр β алгоритм недообучается, функционал не доходит до минимума. При маленьких значениях параметра функционал минимален,

но точность алгоритма сильно колеблется, что может говорить о несходимости алгоритма. В данном случае оптимальным параметром является значение 0.33, так как функционал меньше, чем при больших значениях β, а с другой стороны графики точности и функционала не колеблются, то есть алгоритм сходится.



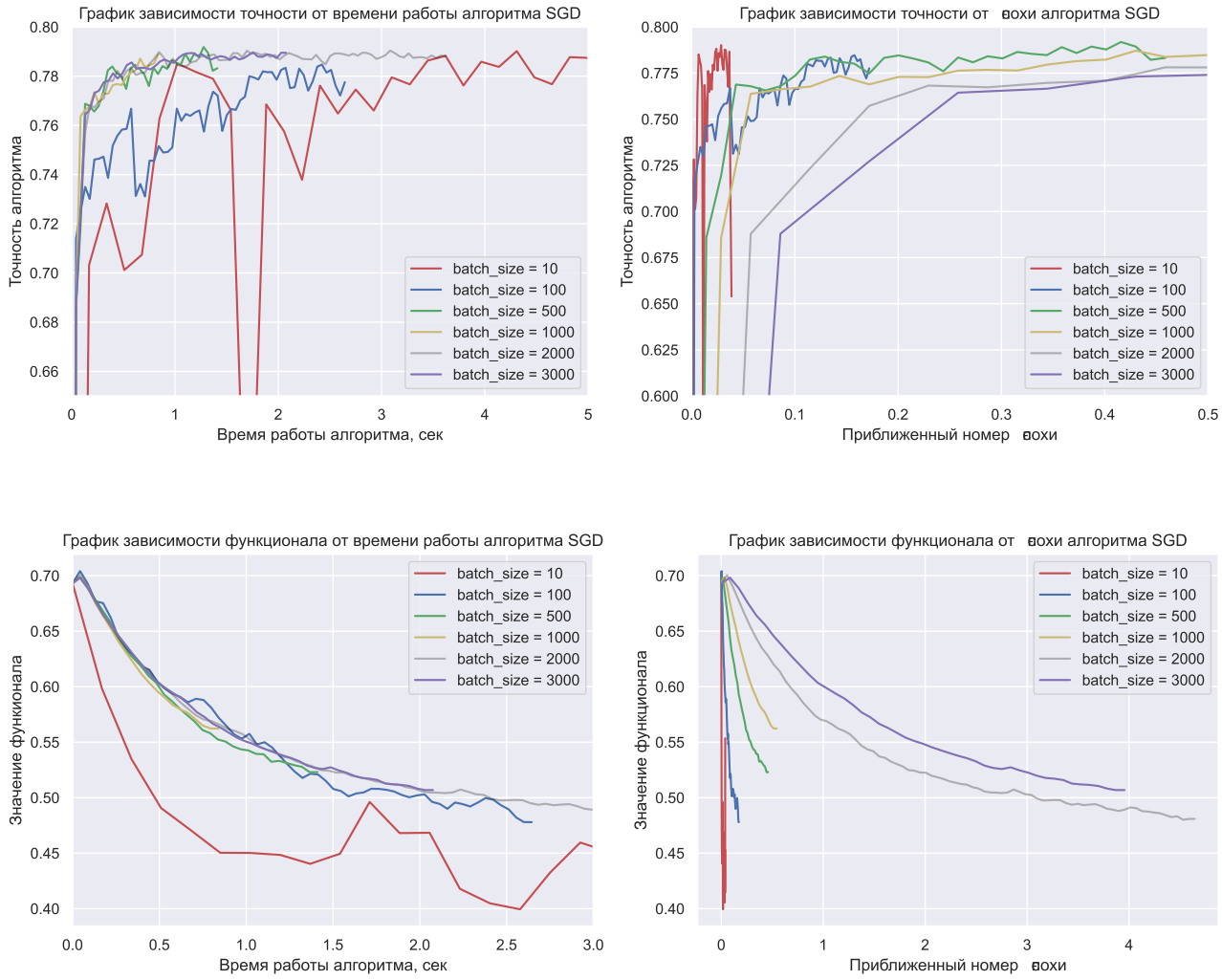
4.3 Зависимость от начального приближения

Подход как в градиентном спуске не подходит. Сходимость алгоритма SGD сильно зависит от начального приближения и от параметров. Алгоритм в данном исследовании сошелся только при начальном векторе из нормального распределения с нулевым математическим ожиданием.

4.4 Зависимость от параметра batch_size

Если брать слишком маленькую подвыборку, то алгоритм сильно колеблется и работает долго, так как не может сойтись. Но уже при подвыборке размером 100 объектов, изменение функционала происходит примерно так же, как и при выборке размером 2000. Но точность алгоритма при таком размере недостаточна. При выборке размером 500 объектов, график точности ведёт себя так же как и при больших размерах подвыборки. Чем больше batch_size,

тем дольше работает алгоритм, так как сложность вычислений растёт, но при маленьких значениях алгоритм может не сходиться.



5 Сравнение алгоритмов

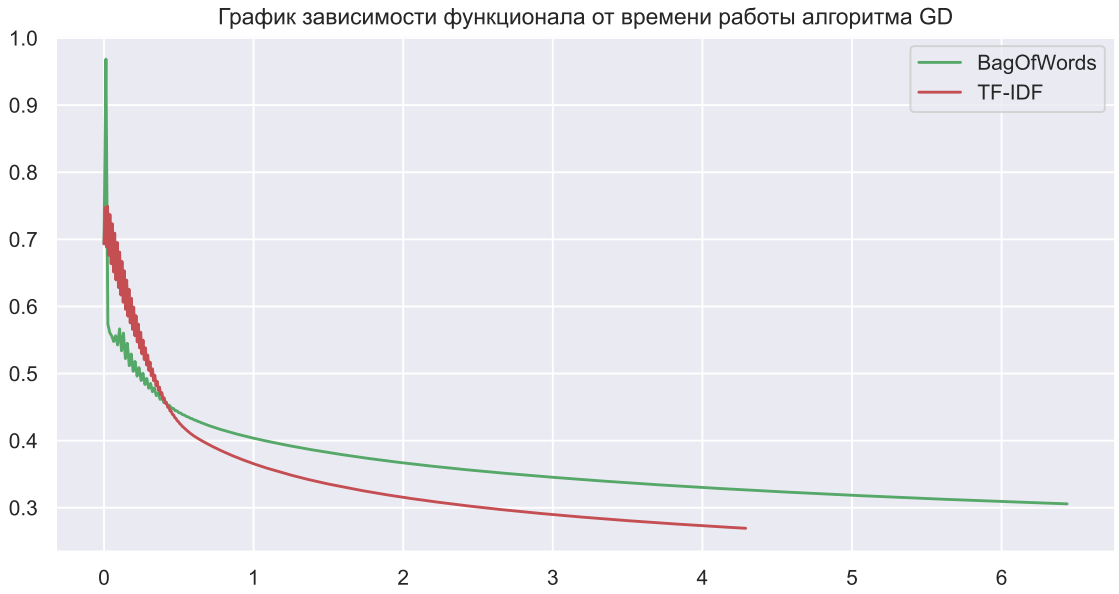
Стохастический метод градиентного спуска работает менее точно, так как градиент функционала считается не на всей выборке, но работает быстрее. Точность при исследуемых параметрах ($\alpha = 0.33, \beta = 0.33, batch_size = 1000$) в SGD составила 0.79, а в GD - 0.87. Потеря в точности является существенной. Скорее всего в данном эксперименте не очень хорошо подобраны параметры для стохастического градиентного спуска. Время работы стохастического градиента примерно в три раза меньше. Также можно заметить, что начальное приближение сильно влияет на стохастический градиентный спуск, он может не сходится в отличие от неоптимизированного алгоритма. Также можно заметить по предыдущим экспериментам, что графики точности и функционала сильнее колеблются в алгоритме SGD, а в GD они имееют более сглаженный вид.

6 Эксперимент 3

Исследуем как предобработка данных повлияет на работу алгоритма. Применим алгоритм лемматизации к документам. Лемматизация - это приведение слов к начальной форме. После лемматизации размерность признакого пространства стала меньше на 700, что является неплохим ускорением вычисления алгоритма. Вычисления точности и времени происходили следующим образом. На выборке без лемматизации ранее в экспериментах были подобраны параметры $\alpha = 0.55, \beta = 0$, дающие самую высокую точность на валидационной выборке. При тех же параметрах tolerance, max_iter, l2_coef были подобраны параметры шага для выборки с лемматизацией. Наилучшими параметрами считались параметры, дающие сходимость и наименьший функционал.

Точность без лемматизации составила 87%, а с лемматизацией - 86%. С помощью такой обработки текстов точность классификации понизилась на 1%, а время работы алгоритма уменьшилось почти не изменилось. Размерность пространства уменьшилась на 600. Ожидалось существенное изменение во времени работы алгоритма и повышение точности.

Теперь выборку из выборки с лемматизацией отбросим все стоп-слова. Это такие слова, которые являются наиболее встречающимся словами в текстах. После данной обработки также были подобраны новые параметры. И точность алгоритма составила почти 90%. Время работы алгоритма уменьшилось в 4 раза и составило 1,4 секунды. Размерность признакого пространства уменьшилось на 100. Данная предобработка данных сильно повлияла на точность классификации в положительную сторону.



7 Эксперимент № 4

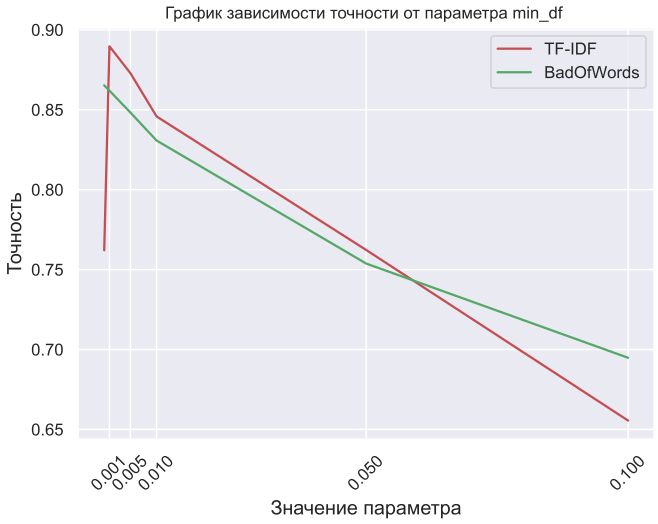
В данном эксперименте рассматривается другой вид векторизации. В представлении TF - IDF в каждом документе вычисляется две величины. TF(term frequency) - отношение числа вхождений некоторого слова в данный документ к общему числу слов документа. IDF (inverse

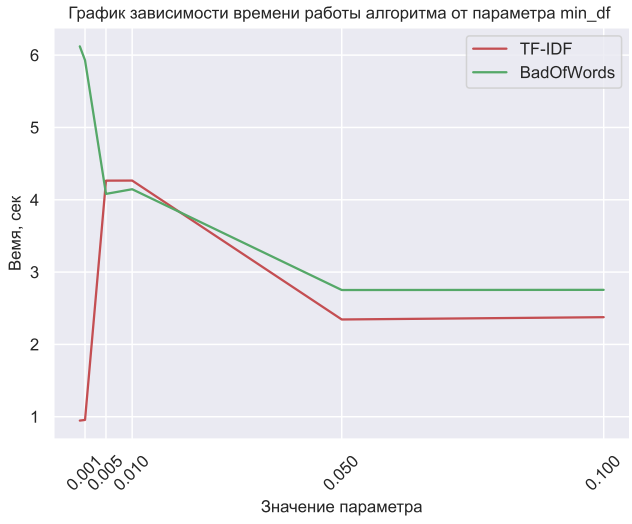
document frequency):

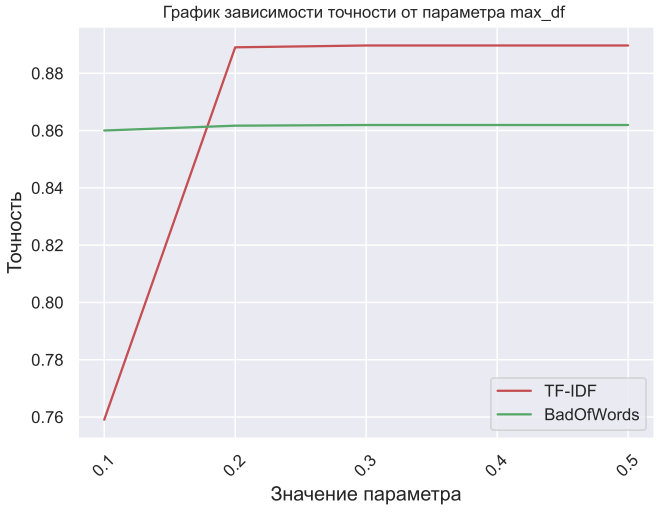
$$idf(t) = \log\frac{1+n}{1+df(t)} + 1,$$

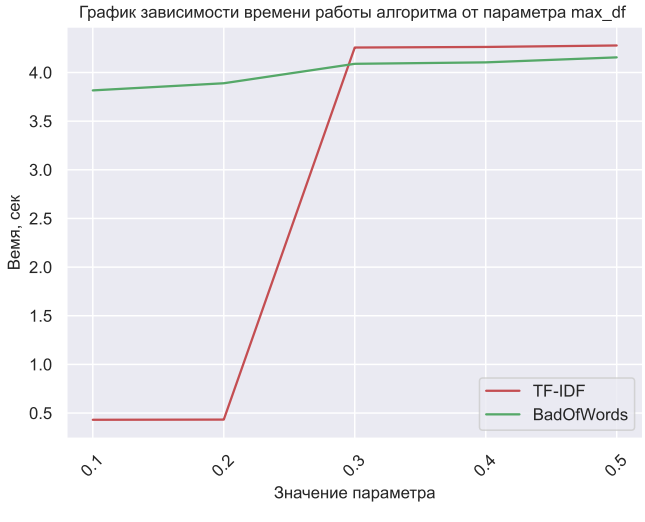
где n - число документов в коллекции, df(t) - количество документов в коллекции, содержащих слово t. Таким образом данная векторизация учитывает частоту встречаемости слова в языке.

Векторизация была осуществлена с помощью TfidfTransformer из библиотеки sklearn с параметром min_df = 0.001. Размерность признакого пространства совпала с размерностью признакого пространства с представлением BagOfWords с применением лемматизацией и удалением стоп-слов. Точность алгоритма составила 89%. Время работы алгоритма - 4,3 секунды, что на 2 секунды меньше, чем метод BagOfWords без обработки данных, но больше, чем BagOfWorgs с обработкой.









Также у конструкторов есть два параметра:

- min_df - при составлении словаря игнорируются термины, частота которых меньше данного параметра

- max_df - при составлении словаря игнорируются термины, частота которых больше данного параметра

Пусть max_df = 0. При увелечении параметра min_df точность классификации у обоих алгоритмов снижается, но у представления TF-IDF точность с применением данного параметра больше, чем без применения, в отличии от BagOfWords. Размерность признакого пространства в обоих случаях совпадает и убывает с ростом значений параметра. Можно заметить, что при больших значениях параметра алгоритм BadOfWords работает точнее. Время обоих алгоритмов убывает, но TF-IDF работает дольше с применением данного параметра, чем без него.

Пусть теперь min_df = 0.001, а max_df изменяется. Размерность признакого пространства одинакого возрастает с ростом параметра. TF-IDF работает дольше при большом значении параметра, но более точно. При маленьких значениях параметра точность выше у BagOfWords, но время работы алгоритма тоже больше. Данный параметр сильно влияет на работу алгоритма TF-IDF и почти не влияет на BagOfWords.

8 Эксперимент № 5

Проанализируем ошибки алгоритма. Для этого применим сначала к выборке лемматизацию, уберём все стоп-слов и представим их в виде TF-IDF. Посмотрим на каких комментариях алгоритм ошибается.

Например, комментарий 'Hey... what is it..n@ | talk .nWhat is it... an exclusive group of some WP TALIBANS...who are good at destroying, self-appointed purist who GANG UP any one who asks them questions abt their ANTI-SOCIAL and DESTRUCTIVE (non)-contribution at WP?nnAsk Sityush to clean up his behavior than issue me nonsensical warnings...' алгоритм посчитал нетоксичным, но он является токсичным. Скорее всего алгоритм ошибается на этом документе из-за того, что здесь нет мата. В большинстве токсичных комментариях присутствуют нецензурные слова. Также пользователи могут использовать различные символы ('#', '*', '@') вместо букв в нецензурной лексике, а при обработке данных все такие символы заменялись на пробелы.

Также оценка токсичности комментарии имеет субъективный характер или зависит от контекста. Например, комментарий God is dead I don't mean to startle anyone but God is dead. We should not worry about him anymore. Just thought I would let everyone know. Well, goodbye and good luck with your newfound crisis of faith!"одной группе людей покажется обидным и неприятным, а другим нет.

9 Заключение

Параметры шага градиентного алгоритма лучше всего подбирать одновременно. Алгоритм стохастического градиентного спуска работает быстрее, но менее точно. При применении алгоритма SGD надо аккуратно подбирать параметры, сходимость метода сильно зависит от гиперпараметров. В обоих методах параметр α надо выбирать наибольшим из тех, при которых алгоритм сходится. В случае GD параметр β модно положить равным 0, но в SGD данный параметр является обязательным для сходимости метода.

Предварительная обработка текстов очень важна.Представление текста в виде вектора даёт разреженную матрицу, то есть огромный размер признакого пространства. Для увелечения

скорости применяют разные методы. С представлением TF-IDF работает быстрее, чем с векторизацией BagOfWords. Применение TF-IDF почти эквивалентно применению BagOfWords с предварительной лемматизацией и выкидыванием стоп-слов из словаря. Но последний метод в данном случае работает быстрее. Также размерность пространства можно регулировать параметрами min_df и max_df конструкторов.

В данных экспериментах удалось достигнуть точности 89%. Но алгоритм ошибается на комментариях, которые являются токсичными, но в них нет мата или мат зашифрован.

10 Приложение

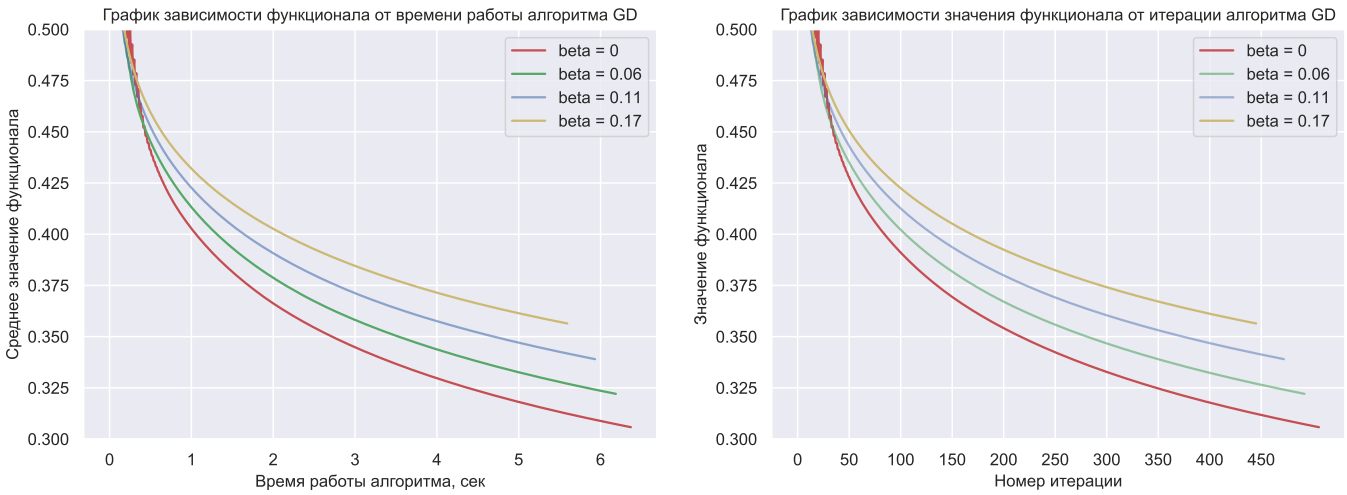


Рис. 1

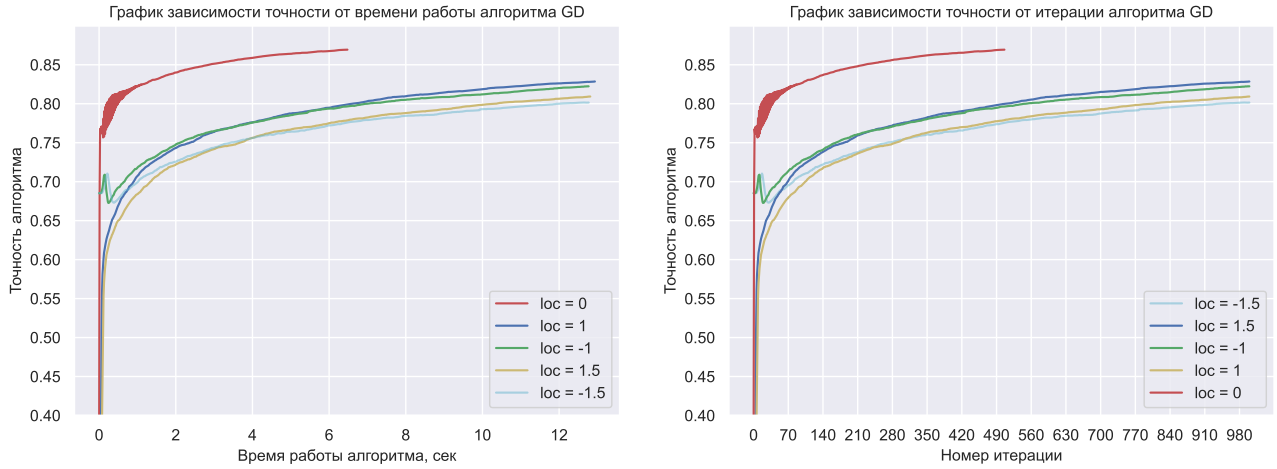


Рис. 2

11 Литература

1. sklearn.feature_extraction.text.TfidfVectorizer
2. sklearn.feature_extraction.text.CountVectorizer