Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра математических методов прогнозирования

Градиентные методы обучения линейных моделей. Применение линейных моделей для определения токсичности комментария.

Практикум на ЭВМ

Выполнил:
Кадченко И. Е.

2022

Содержание

Постановка задачи

Данный отчёт отражает проделанную мною работу по ознакомлению с линейными моделями и градиентными методами обучения, их собственной реализации, а также по проведёнными экспериментами с модельными данными. Исследовать линейные модели предлагается в задаче определения токсичности комментария.

Данные содержат порядка 72000 комментариев, 52000 из которых используется для обучения. В свою очередь, 20% от обучающей выборки используется в качестве валидационной для подбора гиперпараметров модели. Оцениваться предсказательная способность модели будет по метрике качества accuracy.

Теоретическая часть

Градиент бинарной логистической регрессии

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

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

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

Градиент многоклассовой логистической регресии

$$Q(X,w) = -\frac{1}{l}\sum_{i=1}^{l}\log(P(y_i|x_i)) = -\frac{1}{l}\sum_{i=1}^{l}\sum_{j=1}^{K}[y_i = j] log(\frac{exp w_j^T x_i}{\sum_{k=1}^{K} exp w_k^T x_i})$$

$$\nabla Q = -\frac{1}{l}\sum_{i=1}^{l}[y_i = j](x_i - \frac{\exp w_j^T x_i}{\sum_{k=1}^{K}\exp(w_k^T x_i)} x_i) + [y_i \neq j] - \frac{\exp(w_j^T x_i)}{\sum_{k=1}^{K}\exp(w_k^T x_i)} x_i) = \\ -\frac{1}{l}\sum_{i=1}^{l}(x_i(-P(y_i = j|x) + [y_i = j]))$$

Эквивалентность многоклассовой и бинарной логистической регрессии при $K = 2$

Вероятность k-ого класса для многоклассовой регрессии определяется по формуле:

$$P(y = j|x) = \frac{\exp(w_j^T x)}{\sum_{k=1}^{K}\exp(w_k^T x)}$$

Для случая с $K = 2$ (классы 1 и -1):

$$P(y = 1|x) = \frac{\exp(w_1^T x)}{\exp(w_1^T x)+\exp(w_{-1}^T x))} = \frac{1}{1+\exp((w_{-1}^T - w_1^T)x)} = \sigma((w_1^T - w_{-1}^T)x) = \sigma(-w^T x)$$

Здесь мы сделали замену $w_{-1}^T - w_1^T = w^T$, сведя к вероятности класса для бинарной логистической регрессии.

	Численно	Аналитически
Норма	2.29894598	2.29894605
Время(с)	164.353	0.026

Рис. 1: Сравнение способов вычисления градиента

Предварительная обработка текста

Для улучшения качества работы алгоритмов произведём предобработку комментариев. Приведём все буквы к нижнему регистру, также удалим все лишние символы, не относящиеся к буквам и цифрам. Стоит заметить, что знаки пунктуации, регистр могут придавать комментарию дополнительный окрас, превращая его в токсичный. Мы не будем учитывать данное замечение, а лишь будем опираться на зависимость от самих слов.

Преобразование в разреженную матрицу

Текст трудно анализировать алгоритмами машинного обучения, поэтому приведём его к виду матрицы чисел. Будем использовать представление BoW (BagOfWords), по которому каждый комментарий преобразуется в вектор, в позициях которого будет количество встретившегося в данном комментарии соответствующего слова. При таком кодировании размерность каждого вектора будет равна размерности словаря слов из всех комментариев, а каждый вектор преимущественно будет состоять из нулей, поэтому будет использоваться разреженное представление матрцицы.

Сравнение численного и аналитического вычисления градиента

Посчитаем градиент на обучающей выборке численным и аналитическим способами. Оценим численно вычисленный градиент, используя евклидову норму вектора, а также посмотрим на время работы каждого метода.

Значения градиента, вычисленные по аналитической и численной формулам, отличаются незначительно (см. Рис.1). Время, затраченное на подсчёт численным способом, почти в 1000 раз больше аналогичного времени для аналитического подхода. Градиентный и стохастический спуски используют вычисление градиента на каждой итерации, поэтому время его подсчёта должно быть как можно меньше. Далее для нашей задачи будет использоваться аналитический подход.

Поведение градиентного спуска в зависимости от гиперпараметров

Скорость обучения в задании считается по формуле:

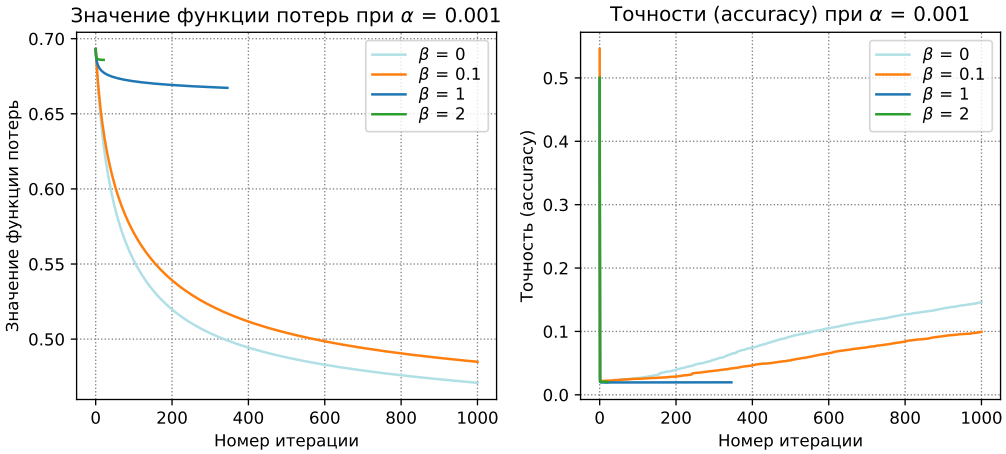

Рис. 2: GD при $\alpha = 0.001$

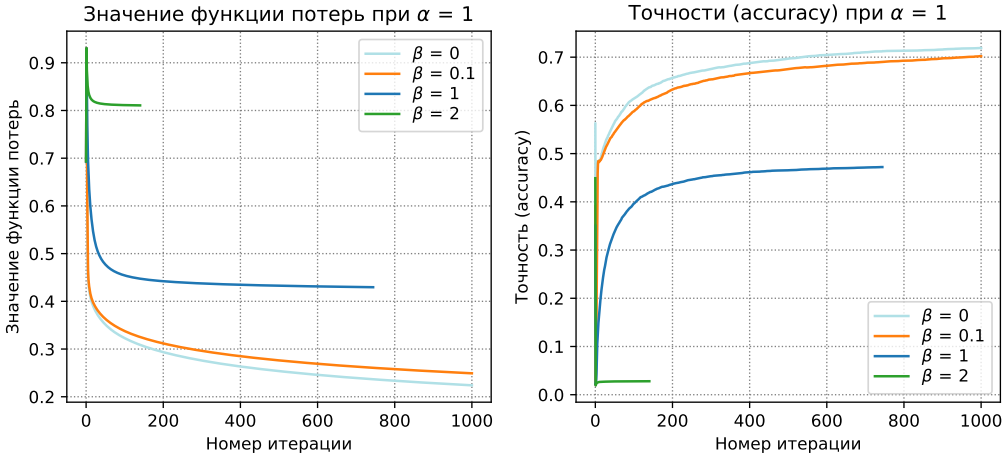

Рис. 3: GD при $\alpha = 1$

$\eta_k = \frac{\alpha}{k^\beta}$, где k - номер итерации, α и β - подбираемые параметры.

Исследуем значения функции потерь и точность предсказания в зависимости от номера итерации метода для разных параметров конструктора $GD-Classifier$. Будем рассматривать зависимости от параметров $step\_alpha$ из множества [0.001, 0.01, 0.1, 1, 2], $step\_beta$ из множества $[0, 0.1, 1, 2]$, а также рассмотрим разные начальные приближения параметра w_0. Параметры по умолчанию: $max\_iter = 1000$, $l2\_coef = 0$. Будем проводить исследование на валидационной выборке.

Рассмотрим сначала зависимости от параметров $step\_alpha$ и $step\_beta$ вместе, так как скорость обучения зависит от обеих констант, фиксируя α:

Для $\alpha = 0.001$ (Рис.2) градиентный спуск показывает очень низкую точность предсказания, модель обучается плохо. Заметим, что наименьшие значения функции потерь достигаются при $\beta = 0$, как и большее значение $accuracy$. Хуже всего модель обучается при $\beta = 2$.

Для $\alpha = 1$ (Рис.3) также лучшим значением β с точки зрения функции потерь является $\beta = 0$, как и для $accuracy$. При $\beta = 2$ снова худшие показатели.

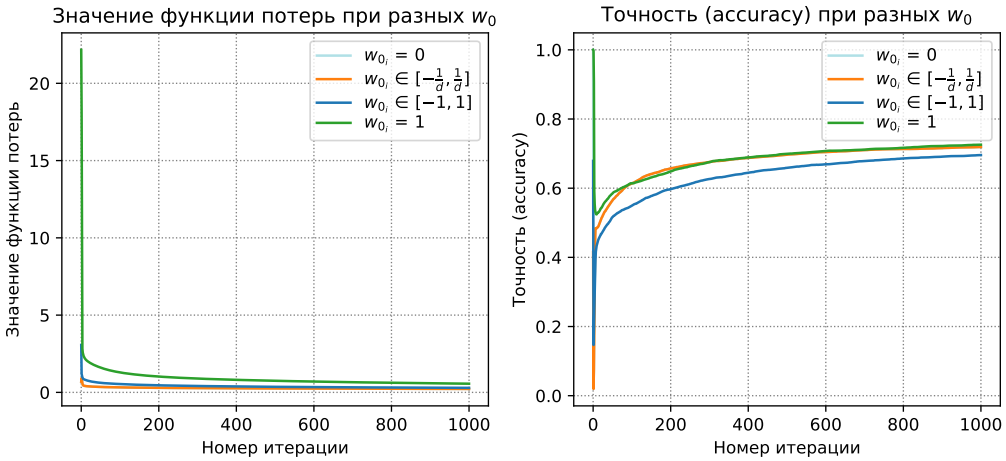

Рис. 4: GD при разных w_0

Это связано с тем, что градиентный спуск быстро сходится и веса не успевают приблизиться к оптимальным значениям, о чём можно утверждать по низкой точности модели. При $\alpha = 1$ почти в два раза меньше значения функции потерь для соотвествующих итераций, а также повышается точность по сравнению с $\alpha = 0.001$.

Графики зависимости для $\alpha = 0.01, 0.1, 2$ (Рис.16-18) можно посмотреть во вложении.

Исходя из графиков, оптимальными значениями параматеров *step_alpha* и *step_beta* являются 1 и 0 соответственно. Далее будем использовать их по умолчанию.

Рассмотрим зависимость значения функции потерь и точности от начального приближения весов. Инициализировать будем следующими способами:

- $w_{0_i} = 0$
- $w_{0_i} \in [-\frac{1}{d}, \frac{1}{d}]$, где d - размер вектора
- $w_{0_i} \in [-1, 1]$
- $w_{0_i} = 1$

Константное начальное приближение и инициализация случайными числами из окрестнсти нуля демонстрируют похожую точность, выше, чем случайные числа из отрезка $[-1, 1]$. Но приближение из окрестности даёт меньшие значения функции потерь, поэтому будем считать его оптимальным. Начальная точность для всех приближений оказывается нестабильной (зигзагообразность на графике) ввиду особенностей инициализации (точность может оказаться нулевой), но с небольшим увелечением номера итераций точность становится более стабильной. Также из особенностей, велико начальное значение функции потерь для начального приближения весов единицами. Это объясняется тем, что мы берём большие веса, и алгоритм оказывается далеко от оптимальных значений.

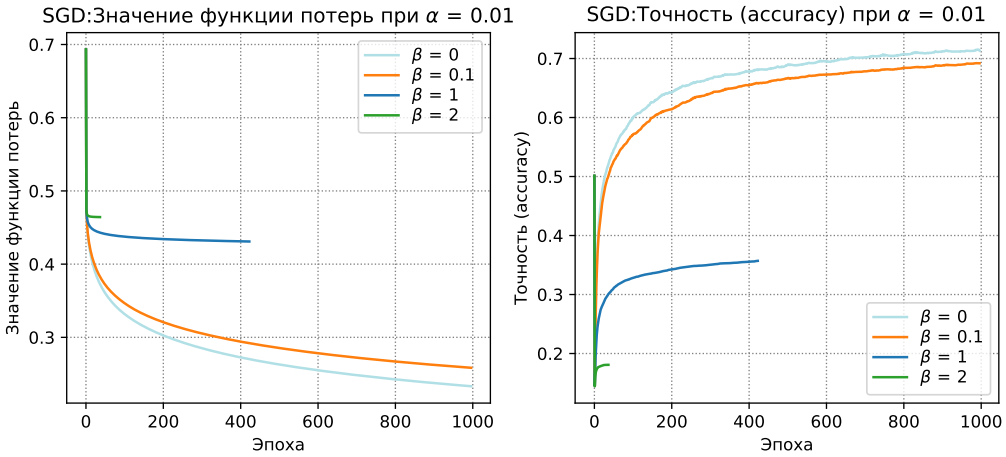

Рис. 5: SGD при $\alpha = 0.01$

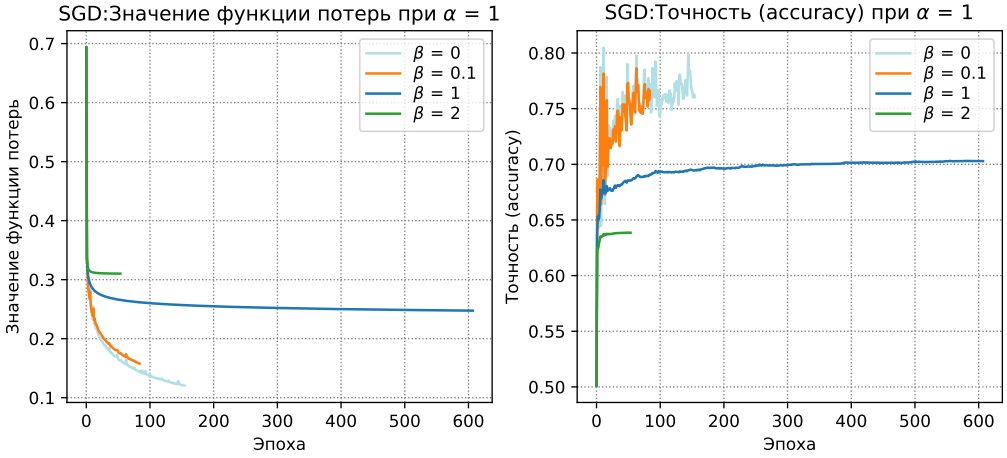

Рис. 6: SGD при $\alpha = 1$

Поведение стохастического спуска в зависимости от гиперпараметров

Аналогично предыдущему эксперименту, исследуем зависимость значения функции потерь и точности от приближённого номера эпохи. Анализ будем производить по средством варьирования гиперпараметров конструктора $SGD-Classifier$: $step\_alpha$ и $step\_beta$ в тех же диапазонах, что и для GD; $batch\_size$ из множества $[8, 64, 512, 4096]$; начальное приближение w_0 - аналогично GD.

Рис.5-6 отражают зависимость стохастичекого спуска от $step\_alpha$ и $step\_beta$. Как и для градиентного, получаем большее качество при больших α, а большие β обеспечивают меньшую зигзагообразность. Аналогичные рассуждения предыдущему пункту приводят нас к тому, что лучшими значениями параметров являются $step\_alpha = 1$, $step\_beta = 0$. Для $\alpha \in [0.001, 0.1, 2]$ см. Рис.19-21.

Исследуем зависимость стохастического спуска теперь от начального приближения весов. Способы инициалирации возьмём подобные рассматрению гра-

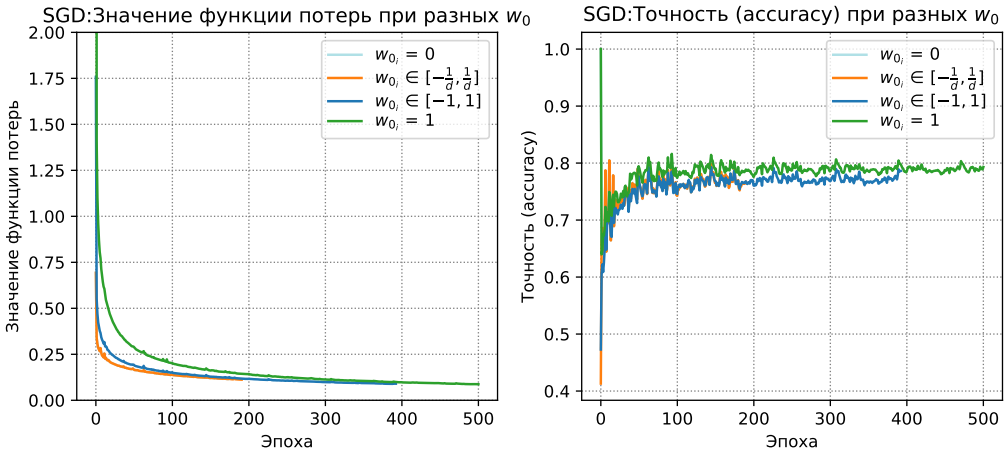

Рис. 7: SGD при разных w_0

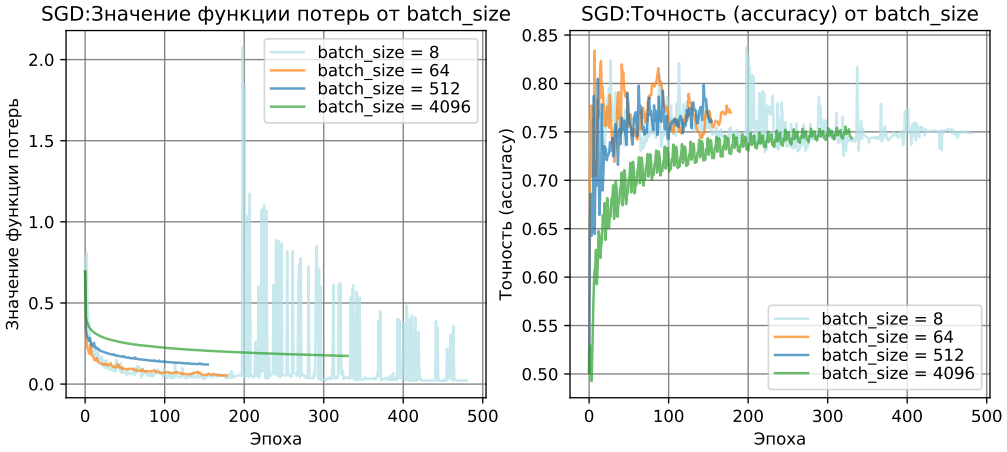

Рис. 8: SGD при разных $batch\_size$

диентного спуска. Результат отражён в Рис.7.

Рассуждения также повторяются, оптимальным начальным приближением возьмём вектор из окрестности нуля $w_{0_i} \in [-\frac{1}{d}, \frac{1}{d}]$. Алгоритм сходится быстрее остальных рассматриваемых, при этом не уступая в точности и показывая лучшие результаты для значений функции потерь.

Найдём оптимальное значение параметра $batch\_size$ (см.Рис.8).

Маленькое значение $batch\_size$ приводит к большим колебаниям (заметно как по accuracy, так и по функции потерь). Большое значение $batch\_size$ - стохастический спуск становится похожим на обычный градиентный (см.Рис.8: SGD:Точность (accuracy) от $batch\_size$). Поэтому оптимально будет брать размер батча каким-нибудь промежуточным. По графикам видно, что $batch\_size = 64$ показывает меньшее значение функции потерь и лучшую точность. Далее будем использовать его. Стоит также заметить, что чем меньше размер батча, тем дольше время работы алгоритма.

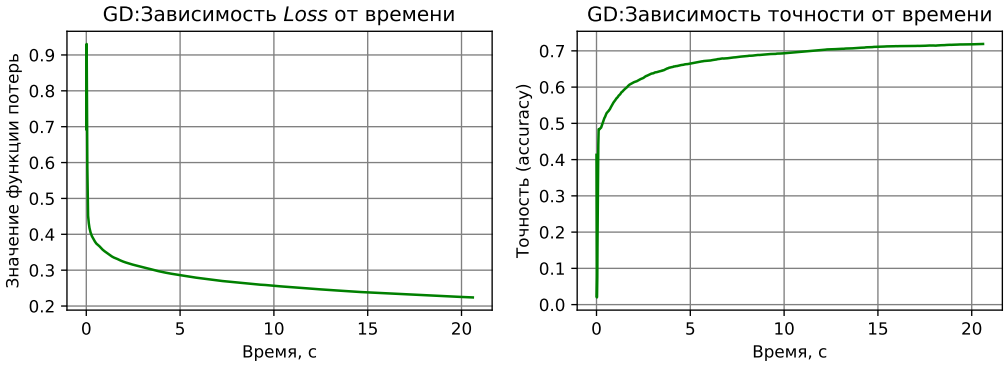

Рис. 9: Зависимость GD от времени

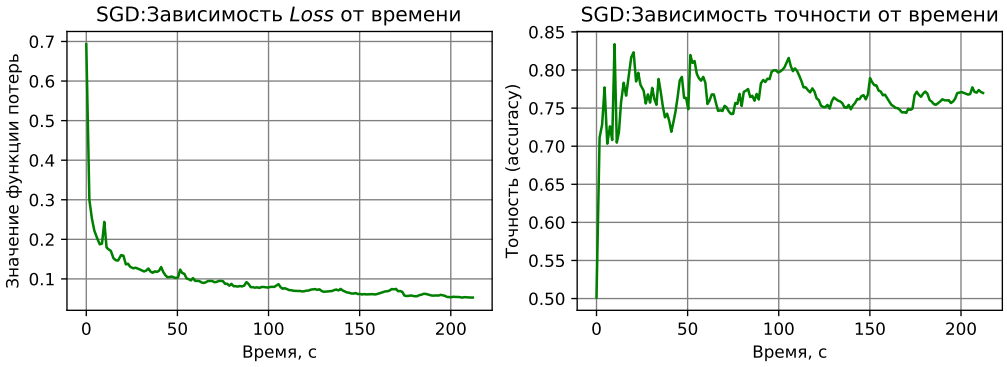

Рис. 10: Зависимость SGD от времени

Сравнение GD и SGD между собой

Посмотрим, какой метод покажет себя лучше на валидационной выборке. Проанализируем зависимости значения функции потерь и точности от времени, а также общую точность каждого метода.

На Рис.9-10 показаны графики для градиентного и стохастического спусков соответственно. В нашей задаче стохастический спуск сходится гораздо медленнее градиентного, так как оптимальность параметров мы определяли в большей степени отталкиваясь от точности результатов (тот же $batch\_size$ для уменьшения времени стоило взять больше 64 если мы хотим уменьшить время, не сильно потеряв в точности). При этом стохастический спуск показывает лучшую точность: 0.7699 против 0.7190.

Далее будем использовать стохастичекский спуск с лучшими для него параметрами для оптимизации accuracy.

Алгоритм лемматизации и удаление стоп-слов

Произведём предобработку корпуса, удалив стоп-слова из обработанных комментариев и приведя слова к начальной форме (осуществим лемматизацию). Благодаря лемматизации мы можем заметно сократить признаковое пространство, удалив похожие слова. Проверим, повысит ли лемматизация точность

	Размерность	Accuracy
С лемматизацией	13005	0.7655
Без лемматизации	18252	0.7699

Рис. 11: Влияние лемматизации

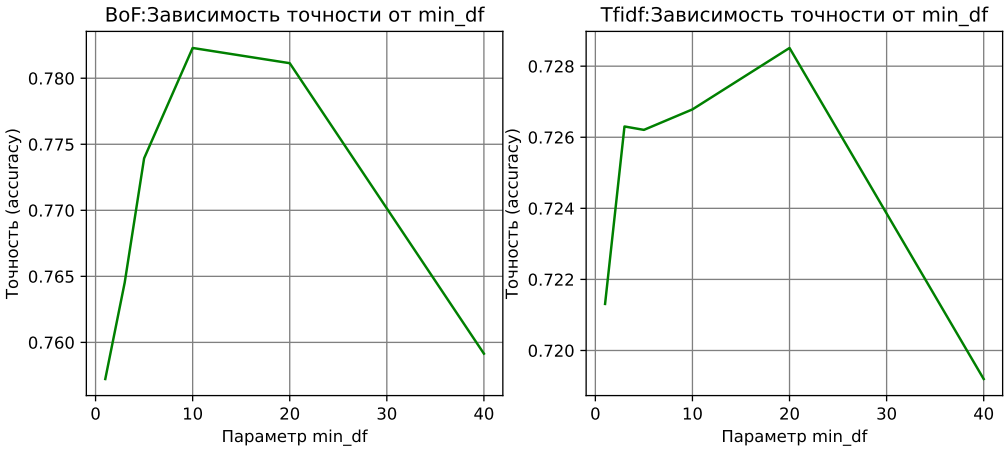

Рис. 12: Зависимость точности от $\min_d f$

предсказания. Построим сравнительную таблицу (Рис.11).

Несмотря на ожидание значительного повышения точности предсказания, лемматизация понизила accuracy. Это можно связывать с тем, что некоторые слова могли потерять свой негативный окрас после преобразования к начальной форме, из-за чего модель стала их неправильно классифицировать. Отметим, что удаление стоп-слов значительно помогло сократить признаковое пространство.

Поведение алгоритма в зависимости от представления текста и параметров min_df, max_df

Требуется исследовать качество, время работы алгоритма и размер признакового пространства в зависимости от того, какое конкретно представление текста используется: $BagOfWords$ или $Tfidf$; какие значения min_df, max_df подобраны. Посмотрим для каждого представления зависимость от min_df, max_df.

Параметр min_df отвечает за редко встречаемые в комментариях слова, выберем оптимальный из множества $[1, 3, 5, 10, 20, 40]$. Параметр max_df оперирует часто встречающимися словами, выберем оптимальный из множества $[1.0, 0.8, 0.5, 0.2, 0.05]$.

По графику на Рис.12 можно видеть, что оптимальным значением параметра min_df для BoW является 10, для $Tfidf$ – 20. С учётом лучшего min_df найдём оптимальный max_df.

Из графика на Рис.13 находим, что лучший max_df для BoW - 0.8, для Tfidf - 0.5. Для нашей задачи большую точность показало представление $BagOfWords$,

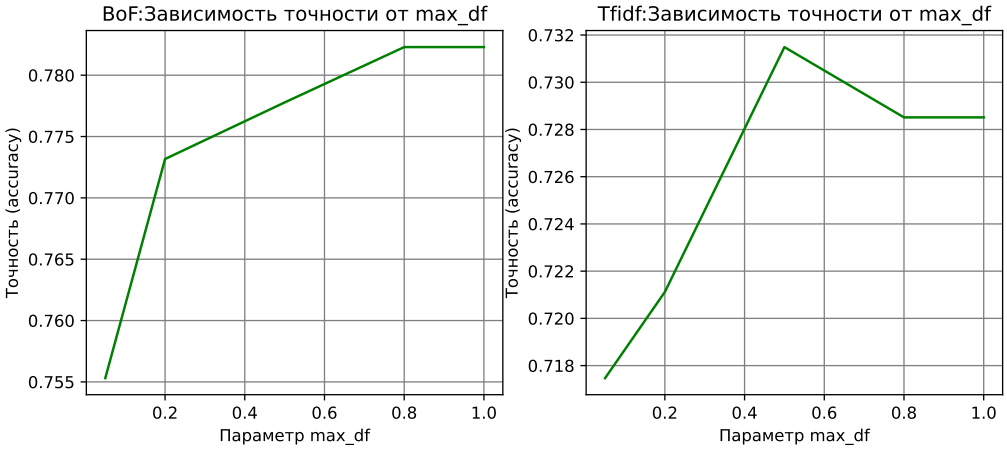

Рис. 13: Зависимость точности от $\max_d f$

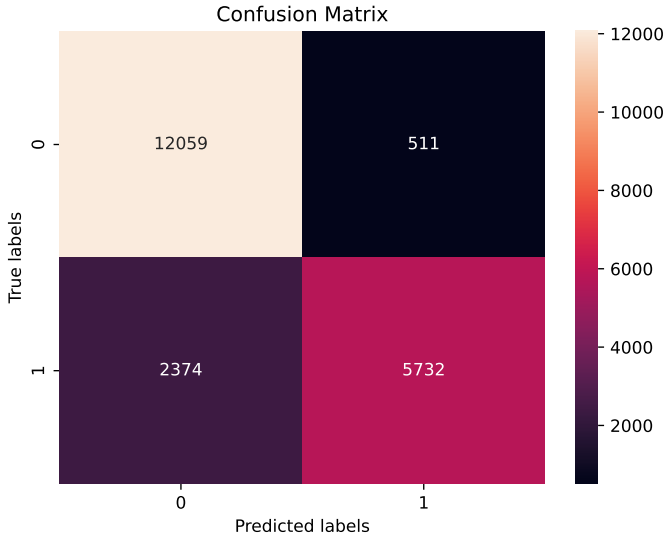

Рис. 14: Матрица ошибок для тестовой выборки

поэтому в следующем эксперименте будем использовать его.

Лучший алгоритм для тестовой выборки. Ошибки.

На основе предыдущих экспериментов мы подобрали оптимальные параметры моделей для нашей задачи. Так, стохастический спуск на тестовой выборке достигает точности 0.8605, градиентный - 0.8521. Причём с лемматизацией обе модели показали небольшое ухудшение точности и на целой тестовой выборке.

Построим матрицу ошибок для тестовой выборки (Рис.14).

Чаще алгоритм ошибается, принимая токсичный комментарий за обратный. Противоположную ошибку алгоритм делает в 4.5 раза реже. В матрице ошибок отражается относительно невысокая точность предсказания: доля ошибок от общего числа объектов велика.

Рассмотрим теперь конкретные комментарии, на которых алгоритм ошибся

№	Комментарий	Предсказание	Ответ
1	"How dare you vandalize that page about the HMS Beagle! Don't vandalize again, demon!"	False	True
2	'I WILL BURN YOU TO HELL IF YOU REVOKE MY TALK PAGE ACCESS!!!!!!!!!!!!'	True	False
3	"Hello everyone I'm just here to tell you that you're all freaks"	True	False
4	'HHAHAHA THE BROWN MAN LOVES SPERM HAH'	True	False
5	"she is also a stipid boild dumpling who can't read or write"	False	True

Рис. 15: Таблица совершённых ошибок

(Рис.15).

В целом можно выделить общие черты объектов, на которых были допущены ошибки. Модель ошибается, когда токсичность может наблюдаться в конктексте целого предложения, как для комментария 1 из таблицы. Также модель не учитывает пунктуацию (восклицательные знаки, например, зачастую являются признаком токсичности комментария), регистр текста (комментарий капсом в большинстве случаев токсичный). Если в комментарии встречаются слова негативного оттенка, с большой долей вероятности линейный классификатор присвоет большой вес этому слову так, что модель ошибётся, как в примере 3.

Также есть комментарии в нашем датасете, ответы для которых вызывают сомнения, как например комментарий 2 и ответ для него. Вероятно, точность модели была бы выше, если таких сомненений не возникало.

Вывод

В ходе выполнения задания была построена собственная реализация линейного классификатора и градиентных методов обучения, были подобраны оптимальные параметры градиентного и стохастического спусков для задачи определения токсичности комментария. Были опробованы разные представления текстовых данных в виде чисел. Лучший алгоритм показал точность 0.8605. Работа подошла к завершению рассмотрением объектов, на которых модель ошибается, были сделаны предположения о природе ошибок.

Источники

Занятие 8: Обработка текстов в Python https://github.com/mmp-practicum-team/mmp_practicum_fall_2022/blob/main/Seminars/08-text-processing-and-logreg(некоторые функции обработки были скопированы отсюда)

Приложение

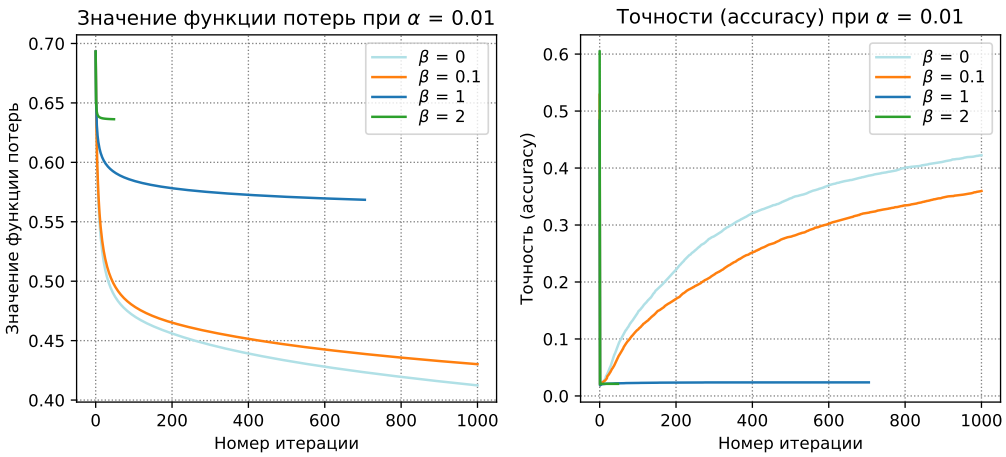

Рис. 16: GD при $\alpha = 0.01$

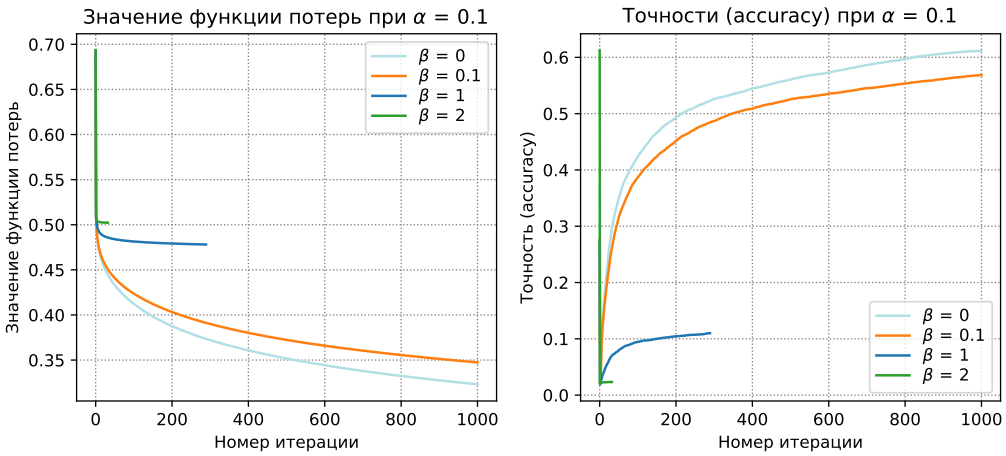

Рис. 17: GD при $\alpha = 0.1$

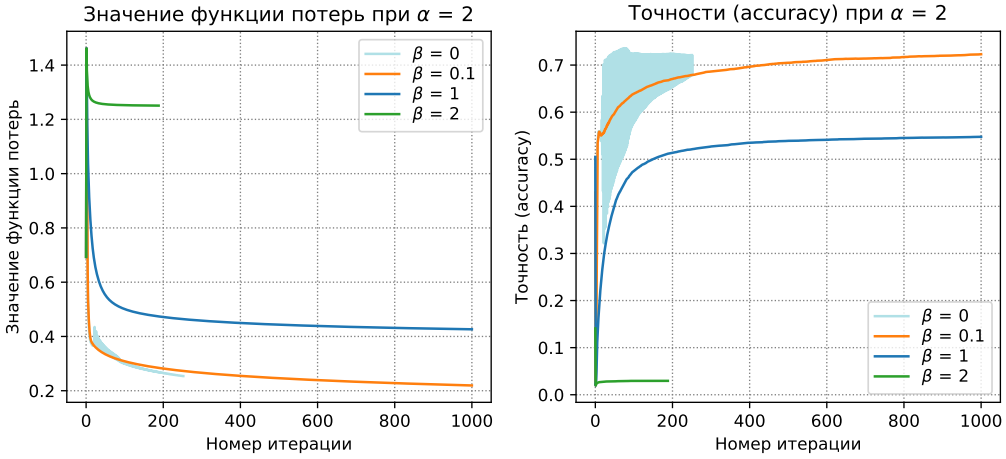

Рис. 18: GD при $\alpha = 2$

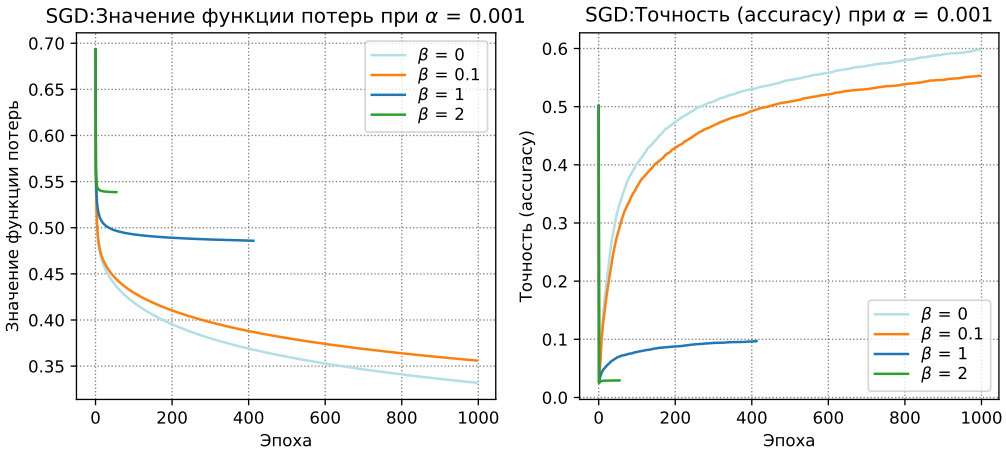

Рис. 19: SGD при $\alpha = 0.001$

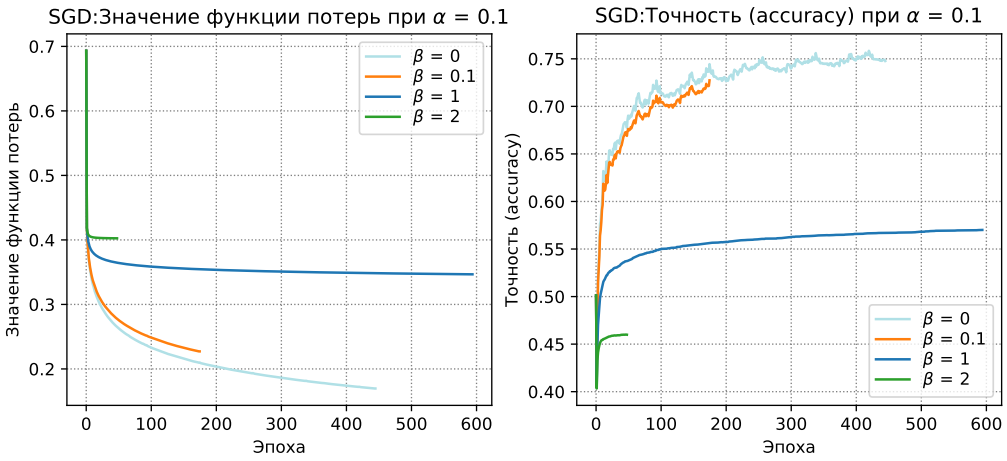

Рис. 20: SGD при $\alpha = 0.1$

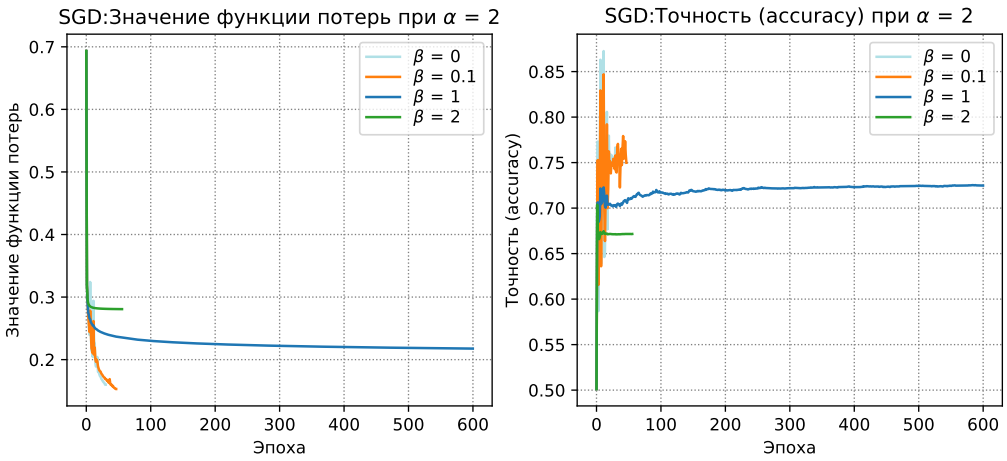

Рис. 21: SGD при $\alpha = 2$