Задание 2. Градиентные методы обучения линейных моделей.

Отчет

о выполненном задании

студента 317 группы факультета ВМК МГУ
Воробьева Сергея Юрьевича

Ноябрь, 2021

Содержание

1 Введение

Задание направлено на ознакомление с градиентными методами обучения линейных моделей. В данном случае предлагается применять линейные модели для определения токсичности комментариев из раздела обсуждений английской Википедии.
В данных находится около `70000` сообщений. Для обучения предлагается взять примерно `50000` изображений. Алгоритм должен предсказывать, является ли сообщение токсичным – оскорбительным или нецензурным. Оценивать алгоритм мы будем по метрике качества `accuracy` (далее иногда будем называть ее «точность»). В данном случае это информативно, хотя в выборке и присутствует небольшой дисбаланс, и удобно, так как легко интерпретировать эту метрику. Нужно провести ряд экспериментов, направленных на подбор гиперпараметров градиентного спуска для улучшения качества алгоритма, а также поработать с текстовыми данными – провести очистку, лемматизацию. Результаты этих экспериментов описаны в этом отчете.

Теоретическая часть

Для начала выведем градиент для бинарной классификации:

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

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

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

Теперь выведем градиент для мультиноминальной регрессии[1]:

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

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

$$-\frac{1}{l}\sum_{i=1}^{l}(x_i([y_i=j]-P(y_i=j|x)))$$

Покажем эквивалентность этих задач при $K=2$. Для мультиноминальной регрессии:

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

Если классы 1 и -1:

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

Переобозначаем $w_{-1}^T - w_1^T = w^T$ и получаем вероятность класса как в бинарной логистической регрессии. Таким образом, задачи эквивалентны.

2 Эксперименты

2.1 Эксперимент 1

На первом этапе необходимо поработать с текстами комментариев – привести все слова к нижнему регистру, убрать все лишние символы. Стоит отметить, что мы можем потерять часть информации из-за такой очистки, например, смайлы могут очень многое сказать о стилистическом окрасе текста. Однако, мы сильно выиграем благодаря очистке текста от мусора.

2.2 Эксперимент 2

Для кодирования комментариев воспользуемся механизмом `BagOfWords`, реализованном в `CountVectorizer`. Таким образом, каждый комментарий будет представлять из себя разреженный вектор. Каждое значение в векторе будет отвечать за количество употреблений конкретного слова в предложении. В данном случае мы теряем большую часть информации о контексте и взаимном расположении слов, однако выигрываем в простоте модели и быстроте кодирования. Стоит отметить, что при таком кодировании размерность пространства будет равна размерности словаря и в векторе в основном будут стоять нули.

2.3 Эксперимент 3

Перейдем к настройке модели. В этом эксперименте нужно подобрать гиперпараметры для градиентного спуска. Будем исследовать зависимость функции потерь и качества от номера итерации и от параметров `step_alpha`, `step_beta`, а также от начального приближения весов. Все модели в этом задании вызываются с параметрами `max_iter=100, l2_coef=0.1, step_beta=0.5, step_beta=1`, и последние два параметра изменяются согласно данным на графиках.

Обучим линейный классификатор на наших данных и посмотрим на значения `log_loss` и `accuracy` на каждой итерации. Точность считалась по вадидационной выборке, которая была дополнительно передана в метод `fit`. Для начала посмотрим на параметр `step_alpha`:

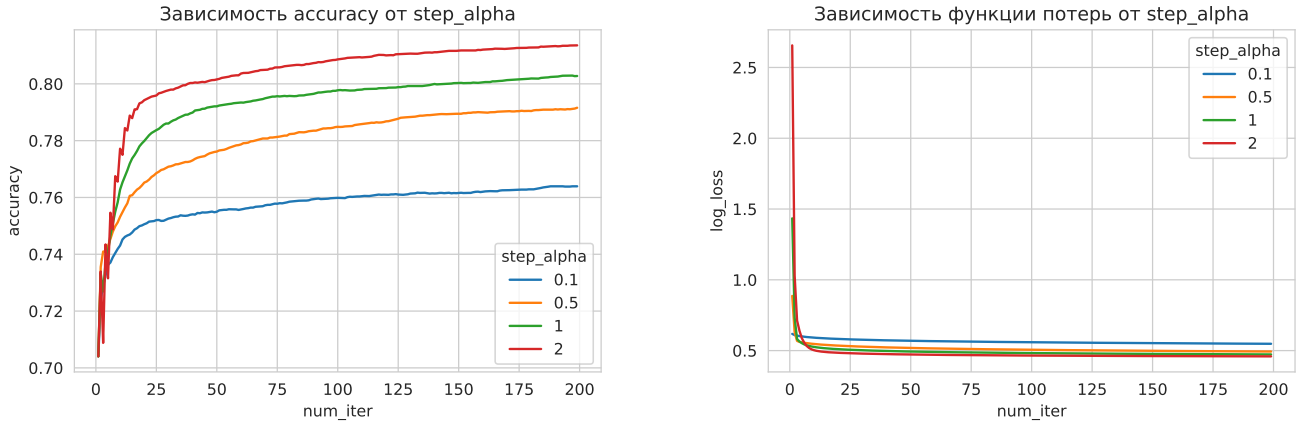


Рис. 1: Исследуем step_alpha.

Можем заметить тенденцию на увеличение качества при увеличении `step_alpha`. Полагаю, это может быть связано с тем, что градиентному шагу проще выбраться из локальных минимумов. Стоит отметить, что при

`step_alpha > 2` заметных улучшений качества не происходит. Также, для всех значений параметра `step_alpha` функция потерь практически не изменяется после нескольких итераций. Таким образом, наш алгоритм сходится довольно медленно. В основном отчете не приложены графики от времени, так как никакой новой информации из них получить нельзя – для данного эксперимента они расположены в приложении. Далее эти графики не строятся.

Посмотрим на графики для `step_beta`:

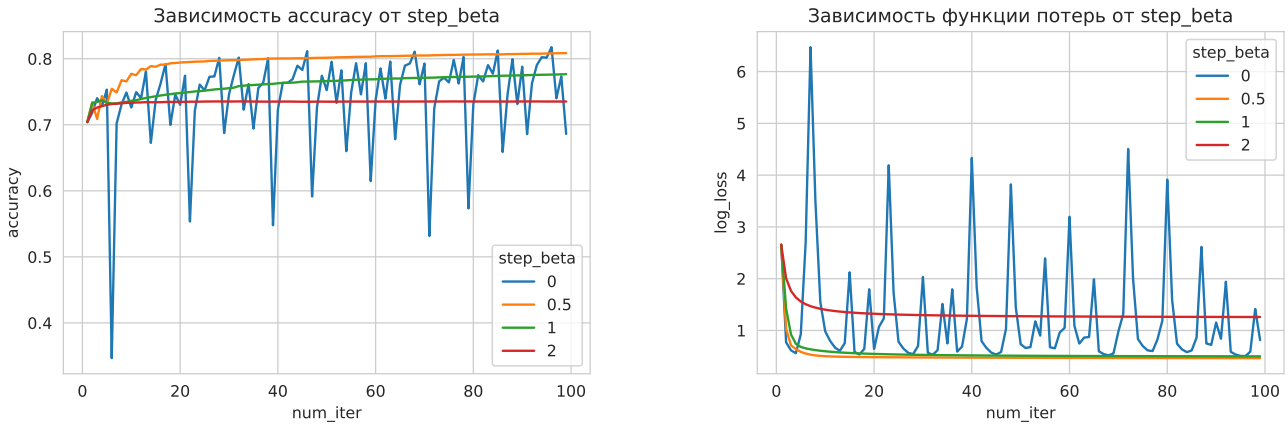


Рис. 2: Исследуем step_beta.

Сразу стоит отметить, что при `step_beta = 0` графики точности и функции потерь сильно колеблются. При `step_beta = 2` наш градиентный спуск попадает в локальный минимум и застревает там. Увеличивать этот параметр нет смысла. Оптимальным значением `step_beta` в данном случае является `0.5` – оно показывает лучшее качество.
Теперь оценим как точность и функция потерь зависят от начальной инициализации весов. Заметим, что признаковое пространство очень большое и, скорее всего, итоговые веса будут сопоставимы с `1/N`, где `N` - количество признаков. В качестве начального приближения были рассмотрены несколько подходов:

- Нулевые веса – простой и логичный (в виду малого порядка весов) вариант.
- Случайные веса – нормально-распределенные веса с центром в 0 и дисперсией `1/N`.
- Равномерные веса – равномерно-распределенные на интервале `[-1, 1]` веса.
- Векторы из `+1` и `-1` – интересный вариант, чтобы посмотреть как поведет себя алгоритм.

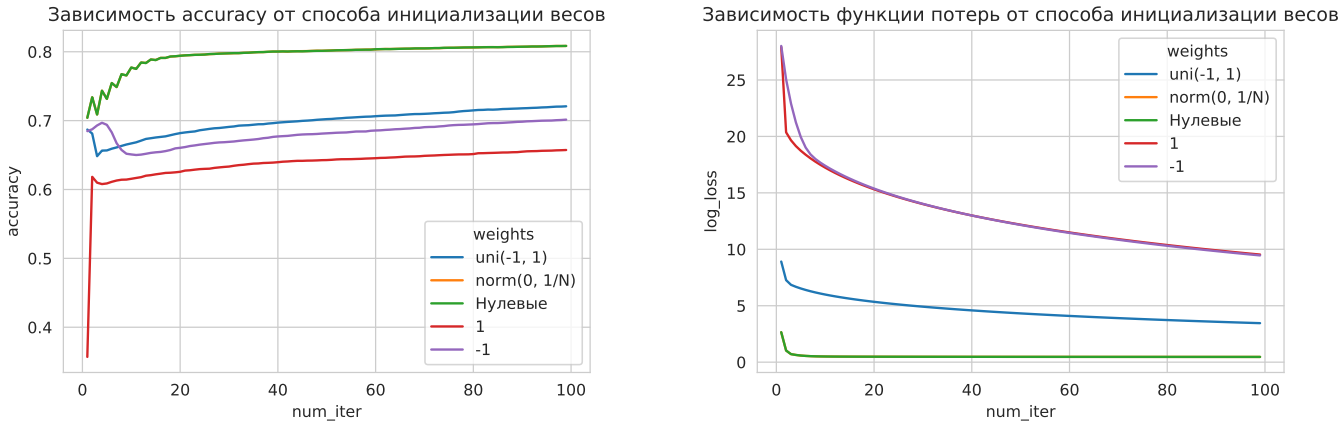


Рис. 3: Исследуем начальные веса.

Сразу заметим, что брать веса большого порядка – `{+/- 1}` было плохой идеей, так как алгоритм изначально находится очень далеко от оптимальных значений и не может попасть в них. Нормально-распределенные веса ведут себя так же как и нулевые, которые в свою очередь являются оптимальным вариантом, судя по графику. Они также выигрывают в простоте реализации. Стоит отметить интересную особенность – из-за того, что предсказания делаются через функцию `np.sign`, которая может возвращать нули, на самой первой итерации точность будет нулевая для нулевых начальных весов. Это стоит учитывать при построении графиков. В данном случае брались значения со второй итерации для удобного масштабирования.

Выводы

Для наилучшего качества нужно взять `step_alpha = 2, step_beta = 0.5, weights = 0`

2.4 Эксперимент 4

В этом эксперименте нужно подобрать гиперпараметры для стохастического градиентного спуска. Все модели в этом задании вызываются с параметрами `max_iter=3000, l2_coef=0.01, step_alpha=2, step_beta=0.5, batch_size=1000`, где последние три варьируются, как на графиках. Дополнительным параметром для исследования тут выступает `batch_size`.

Посмотрим на параметр `step_alpha`:

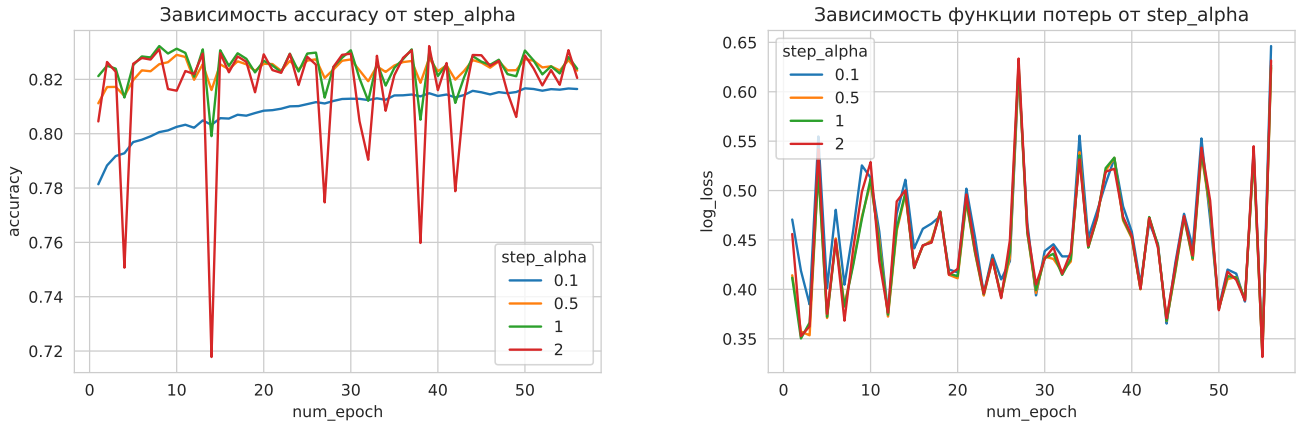


Рис. 4: Исследуем step_alpha.

В первую очередь заметим, что при больших `step_alpha` график колеблется. Это происходит из-за того, что усредняем градиент мы не по всей выборке, а по `batch_size` объектам, поэтому он может колебаться еще сильнее при увеличение шага. В данном случае лучше всего себя показывает значение 1. При `step_alpha < 0.5` наш SGD сходится долго.

Посмотрим теперь на Рис 5 для `step_beta`. Сразу отметим сходство с предыдущим экспериментом. При нулевом значении бета график очень сильно колеблется. В данном случае на это влияет еще и то, что алгоритм обучается на батчах а не на всей выборке. С параметрами `1` и `2` таких сильных колебаний не возникает. В данном случае лучше всего помогают алгоритму сойтись именно они.

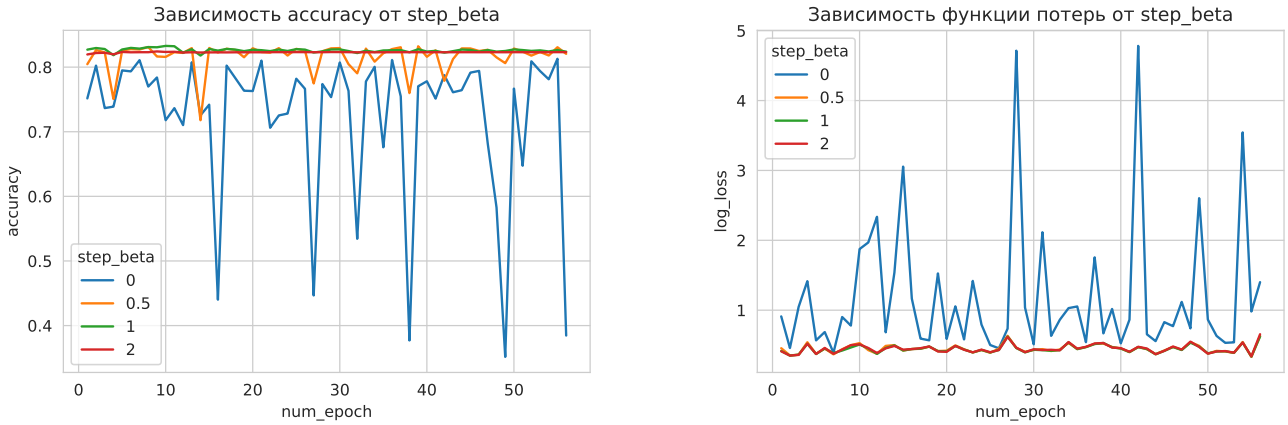


Рис. 5: Исследуем step_beta.

Оценим как точность и функция потерь зависят от начальной инициализации весов:

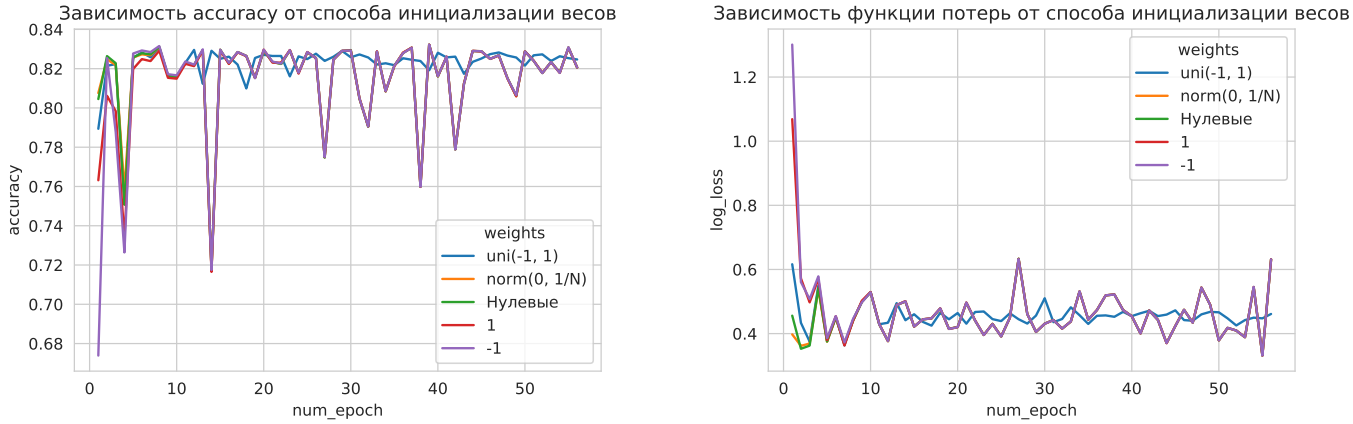


Рис. 6: Исследуем начальные веса.

В этом эксперименте мы пользуемся той же логикой, что и в предыдущем. В данном случае наблюдаем похожую картину: быстрее всего сходятся нулевые и нормальные веса. Однако в SGD и веса, большие по модулю, сходятся быстро.

В стохастическом варианте градиентного спуска мы обучаемся на каждой итерации на каком-то случайном множестве объектов фиксированного размера - батче. Посмотрим, как точность и функция потерь зависят от количества элементов в батче:

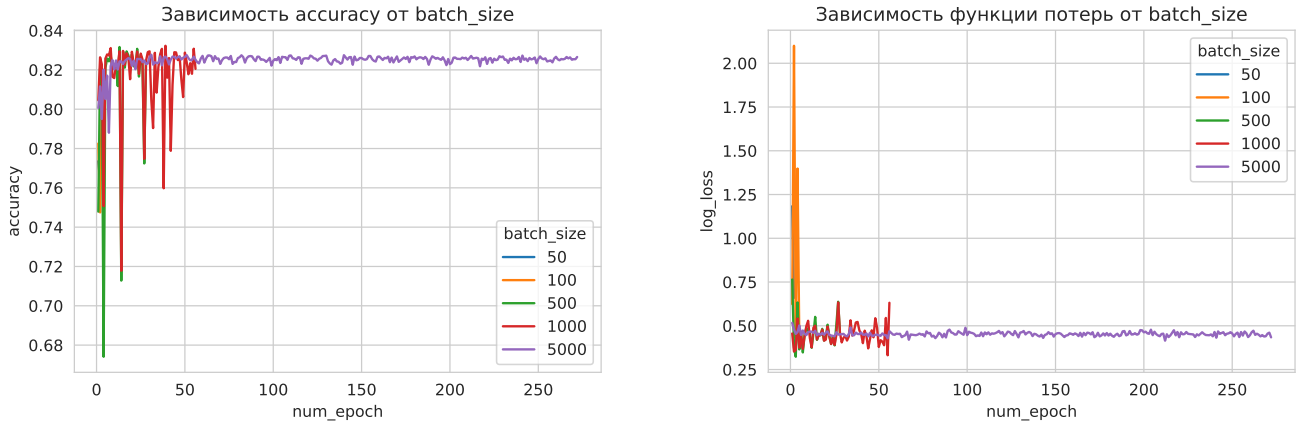


Рис. 7: Исследуем размер батчей.

Судя по графикам, малый размер батча `{50-100}` приводит к большим колебаниям и быстрому застреванию в локальных минимумах. Большой размер батча `5000+` начинает быть похожим на обычный градиентный спуск. Имеет смысл взять `batch_size = 1000`. Судя по графикам, при этом значении достигается высокая точность и алгоритм относительно быстро сходится.

Выводы

Для наилучшего качества нужно взять `step_alpha = 1, step_beta = 1, weights = 0, batch_size = 1000`

2.5 Эксперимент 5

В этом эксперименте нужно сравнить два подхода для обучения — стохастический градиентный спуск и обычный. Обучим каждый алгоритм с лучшими параметрами и на одних данных. Заметим, что одна эпоха стохастического градиентного спуска может быть сопоставлена с одной итерацией обычного градиентного спуска. Посмотрим на получившийся график качества:

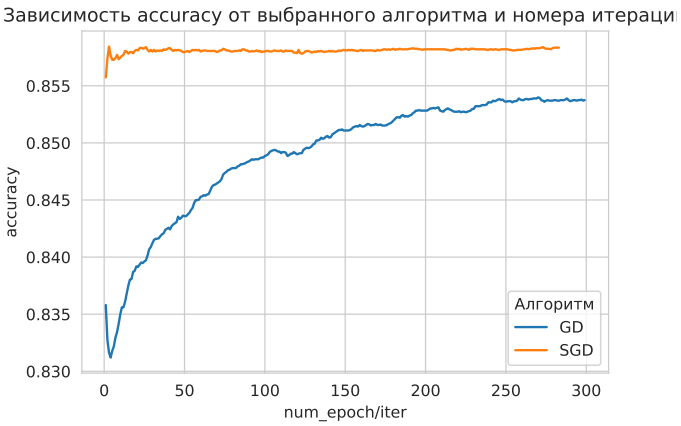


Рис. 8: Сравниваем GD и SGD

По графику видно, что SGD гораздо быстрее сходится к оптимальным значениям и его точность несколько выше, чем у обычного GD. В дальнейших экспериментах будем пользоваться именно стохастическим вариантом из-за скорости сходимости и высокого качества.

2.6 Эксперимент 6

Лемматизация позволяет объединить слова в разных формах в одно слово – это экономит признаковое пространство, не оставляя в нем похожих слов. Посмотрим, поможет ли лемматизация улучшить качество модели и ускорить обучение. Ниже приведена сравнительная таблица.

	Количество признаков	Время выполнения	accuracy
Без лемматизации	91695	48 сек	0.8481
С лемматизацией	85340	46 сек	0.8494

Таблица 1: Исследование лемматизации

В данном случае лемматизация немного ухудшила прогноз, несмотря на ожидаемый положительный эффект. Есть предположение, что этот эффект скомпенсировался тем, что некоторые специфичные для токсичных комментариев слова, могли потерять свое негативное значение и, таким образом, привести к неверной классификации. Стоит отметить, что количество признаков уменьшилось на 5-6 тысяч, что не является существенным в данном случае, поэтому время выполнения почти не изменилось. Преобразование проводилось с `CountVectorizer(min_df=1e-05)`.

2.7 Эксперимент 7

Посмотрим, как качество классификации зависит от параметров векторизации. В этом эксперименте сравнивается разные стратегии векторизации текстов – `BagOfWords` и `Tfidf`, а также посмотреть на качество в зависимости от параметров `min_df` и `max_df`. Для начала сравним алгоритмы `BagOfWords` и `Tfidf`:

	`BagOfWords`	`Tfidf`
accuracy	0.8437	0.8173

Таблица 2: Сравнение `BagOfWords` и `Tfidf`

Согласно таблице 2, `Tfidf` работает ощутимо хуже `BagOfWords`. Есть предположение, что это связано со спецификой подсчета значений для весов слов. В обсуждении Википедии слишком много уникальных слов, которым придается большой вес, в то время как оскорбительных и нецензурных слов не очень много, но они встречаются в различных документах намного чаще. Далее `Tfidf` мы использовать не будем.
Исследуем теперь зависимость качества от параметров `min_df` и `max_df`. Для каждого построим таблицу:

min_df	Количество признаков	accuracy
1e-5	91695	0.8475
1e-4	15848	0.8438
1e-2	446	0.7751

Таблица 3: Исследование параметра `min_df`

max_df	Количество признаков	accuracy
0.3	14246	0.8448
0.1	14236	0.8517
0.05	14208	0.8569
0.001	10852	0.5642

Таблица 4: Исследование параметра `max_df`

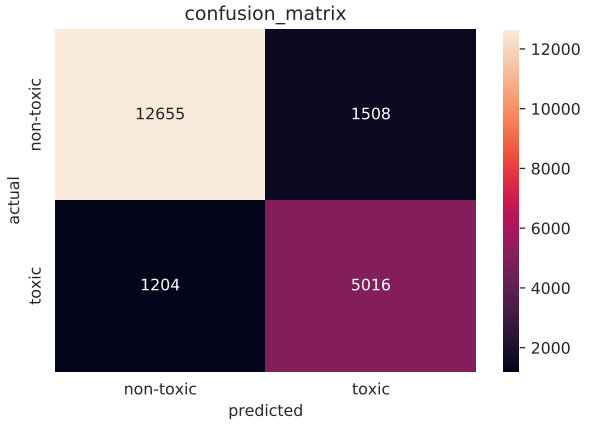
Параметр `min_df` отвечает за редкие слова, а `max_df` за частые.

Заметим, что для высоких значений `min_df` качество классификации заметно ухудшается. Это связано с тем, что при векторизации будут удаляться относительно редкие, но специфичные для токсичных комментариев слова. Оптимальным выбором тут будет значение `1e-4` – мы получаем не слишком большое число весов и приемлемое качество.

Параметр `max_df` будет благоприятно влиять на классификацию при значении 0.05. Таким образом, слова, встречающиеся чаще, чем в 5% документов, будут удалены Слишком низкие значения этого параметра уже существенно ухудшают качество классификации.
Оптимальные параметры: `max_df = 0.05, min_df=1e-4`.

2.8 Эксперимент 8

В итоге, на тестовой выборке получаем качество `0.8582`. `293` объекта наш алгоритм не смог классифицировать, присвоив им ответ 0. Это произошло из-за того, что комментарии состояли полностью из символов, очень редких слов или стоп слов – после чистки такие комментарии превратились в пустые списки. Удалим их при построении `confusion_matrix`.



Заметим, что на нетоксичных комментариях наш классификатор ошибается чуть чаще. В целом качество у данной модели далеко от идеального.

Теперь попробуем взглянуть на примеры с ошибочной классификацией и разобраться, почему так произошло.

Текст комментария	Истинный ответ	Предсказание алгоритма
== Worst Mainstream Movie, EVER!!! == \n Well, in the past decade.) Well, DB: Evolution was probably worse.... D	False	True
==Man Gaga== \n Lets be honest here if your born a man you are a man no offence to her if she was a lady id go her in a heartbeat but unfortunately she has a cock!:0 (UTC	True	False
If you carry on posting bullshit then I'm afraid that I will have to ban your IP from accessing wikipedia in the future. \n\n Thanks, Matt.	False	True
No Homo, his tities look kinda bigg, it looked fake	True	False

Таблица 5: Примеры ошибочной классификации

Из рассмотренных вариантов примерно можем примерно выделить ситуации, в которых классификатор ошибается, разберем ошибочные комментарии по порядку:

1. В тексте присутствую слова с негативным окрасом «Worst» и «worse», однако, сам комментарий не является оскорбительным, скорее его можно назвать критическим. Таким образом, комментарии с негативными словами наш классификатор скорее отнесет к токсичным. Это объясняется моделью кодирования текста - `BagOfWords`. Модель смотрит только на состав предложения, не беря во внимание контекст и коллокации.

2. В данном случае комментарий является язвительным и ироничным, однако, из-за отсутствия негативных слов и прямых оскорблений, наш классификатор не смог корректно предсказать его класс. Таким образом, классификатор плохо справляется с задачей, если в тексте отсутствуют прямые оскорбления или нецензурные слова.

3. Данный комментарий похож на первый, несмотря на присутствие в нем нецензурных слов, сам по себе он не является токсичным. Этот комментарий является скорее предупреждением.

4. В этом случае слова из комментария сами по себе не являются токсичными, однако в этом контексте предложение является непристойным, ситуация похожа на второй случай.

Можем заметить, что все ошибки нашего классификатора объединяет одно – алгоритм кодировния BagOfWords практически не сохраняет информацию о контексте, в котором употребляются слова, а также не сохраняет информацию об их взаимосвязи. Для избежания подобных ошибок можно попробовать изменить метод кодирования текста.

3 Вывод

В данной работе был построен линейный классификатор, а также разработаны и настроены методы градиентного спуска для его обучения. Рассматривалась задача классификации текстов – в нашем случае токсичных комментариев. В ходе экспериментов были подобраны параметры для алгоритма градиентного спуска и стохастического градиентного спуска. Были рассмотрены разные аспекты обработки и кодирования текстов, например, выяснилось, что в нашем случае лемматизация не приносит никаких улучшений в классификации и даже немного ухудшает ее. В конце удалось добиться точности – `0.8582`. По итогу были также выявлены ограничения алгоритма и разобраны примеры, на которых он ошибается.

Список литературы

[1] http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/

4 Приложение

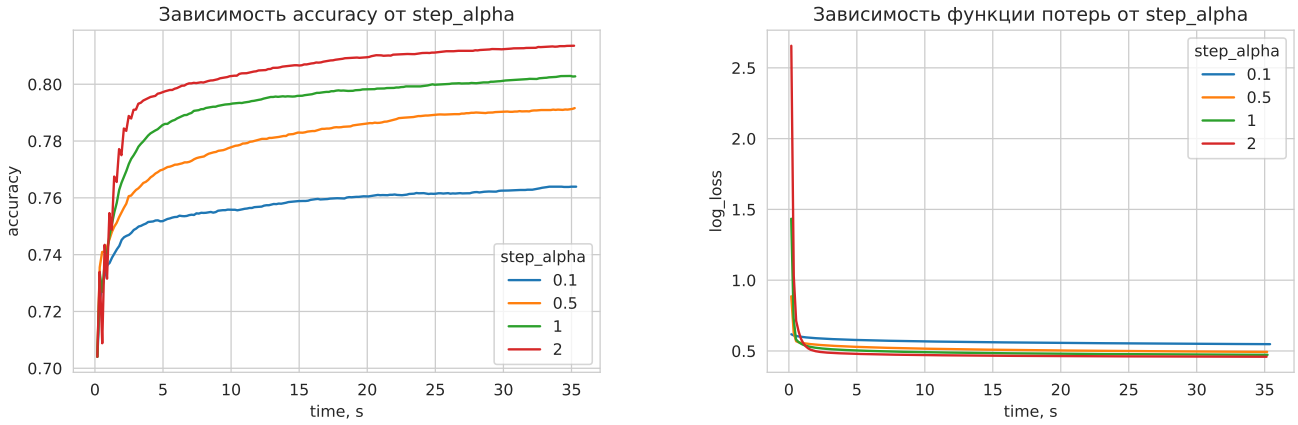


Рис. 9: К эксперименту 3.

Ниже на графике нормальные веса смещены на `0.001`, чтобы графики немного отличались

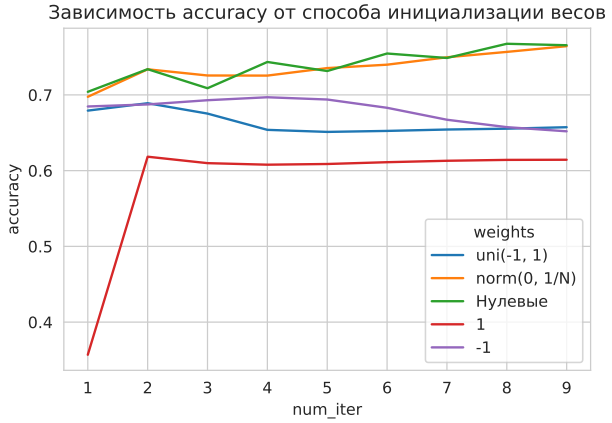


Рис. 10: К эксперименту 3.