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

Практикум на ЭВМ 2021/2022

Отчёт по практическому заданию №2

Работу выполнила:
Козлова Ольга
Группа: 317

Москва
2021

Содержание

1. Введение

Необходимо написать реализацию линейного классификатора на языке программирования Python. Задача классификатора - определять токсичность комментариев, взятых из раздела обсуждений английской Википедии. Для улучшения качества работы надо провести предварительную обработку текста. Для обучения линейного классификатора необходимо вывести формулу градиента функции потерь и провести эксперименты для исследования зависимости поведения градиентного спуска и стохастического градиентного спуска от различных параметров (шага, начального приближения и т.д.). После нахождения лучших параметров градиентного спуска надо попробовать улучшить качество обучения за счет обработки документов.

2. Теоретическая часть

2.1. Градиент функции потерь для бинарной логистической регрессии

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

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

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

2.2. Градиент функции потерь для многоклассовой логистической регрессии

$$Q(X,w_1,...,w_K) = -\frac{1}{l}\sum_{i=1}^{l}\log P\left(y_i \mid x_i\right) = -\frac{1}{l}\sum_{i=1}^{l}\log\frac{\exp\left(\left\langle w_i,x_i\right\rangle\right)}{\sum_{k=1}^{K}\exp\left(\left\langle w_k,x_i\right\rangle\right)})$$

$$dQ(X,w1,...,w_K) = -\frac{1}{l}\sum_{i=1}^{l}\frac{\sum_{k=1}^{K}\exp\left(\left\langle w_k,x_i\right\rangle\right)}{\exp\left(\left\langle w_{y_i},x_i\right\rangle\right)}\left(\frac{\exp\left(\left\langle w_{y_i},x_i\right\rangle\right)\left\langle x_i,dw_j\right\rangle\mathbb{I}[y_i=j]}{\sum_{k=1}^{K}\exp\left(\left\langle w_k,x_i\right\rangle\right)} -\right.$$

$$\left. - \frac{\exp\left(\left\langle w_{y_i},x_i\right\rangle\right)\exp\left(\left\langle w_j,x_i\right\rangle\right)\left\langle x_i,dw_j\right\rangle}{\left(\sum_{k=1}^{K}\exp\left(\left\langle w_k,x_i\right\rangle\right)\right)^2}\right) = -\frac{1}{l}\sum_{i=1}^{l}\left\langle x_i,dw_j\right\rangle\left(\mathbb{I}[y_i=j] - \frac{\exp\left(\left\langle w_j,x_i\right\rangle\right)}{\sum_{k=1}^{K}\exp\left(\left\langle w_k,x_i\right\rangle\right)}\right)$$

$$\nabla_{w_j}Q(X,w_1,...,w_K) = -\frac{1}{l}\sum_{i=1}^{l}x_i\left(\mathbb{I}[y_i=j] - \frac{\exp\left(\left\langle w_j,x_i\right\rangle\right)}{\sum_{k=1}^{K}\exp\left(\left\langle w_k,x_i\right\rangle\right)}\right)$$

2.3. Эквивалентность мультиноминальной регрессии для двух классов и бинарной регрессии

$$w_0 = w, w_1 = -w$$

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

$$= -\frac{1}{l}\sum_{i=1}^{l} y_i \langle x_i, dw\rangle \left(1 - \frac{1}{1+\exp\left(2y_i \langle -w, x\rangle\right)}\right) = -\frac{1}{l}\sum_{i=1}^{l}\left(\frac{y_i \langle x_i, dw\rangle}{1+\exp\left(2y_i \langle w, x\rangle\right)+1}\right)$$

$$\nabla Q(X,w) = \frac{1}{l}\sum_{i=1}^{l}\frac{-y_i x_i}{\exp(2y_i \langle w, x_i\rangle)+1}$$

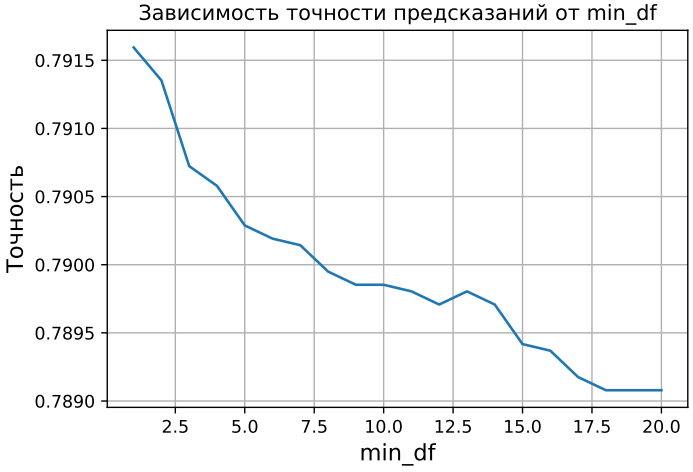
Заметим, что оптимизации будут эквивалентны, так как минимизация по аргументу $f(x)$ эквивалентна минимизации $2f(x)$. Таким образом, минимизация функкции потерь бинарной регрессии эквивалентна минимизации функции потерь многоклассовой регрессии, если сделать замену $w' = 2w''$ (w' - веса бинарной регрессии, w'' - веса многоклассовой регрессии).

$$\nabla Q(X,2w') = \frac{1}{l}\sum_{i=1}^{l}\frac{-y_i x_i}{\exp(y_i \langle 2w', x_i\rangle)+1} = \nabla Q(X,w'') = \frac{1}{l}\sum_{i=1}^{l}\frac{-y_i x_i}{\exp(2y_i \langle w'', x_i\rangle)+1}$$

3. Эксперименты

3.1. Предварительная обработка данных

Для начала привести все слова в комментариях к нижнему регистру и заменить все символы, не являющиеся цифрами и буквами, на пробелы, чтобы не было увеличения количества различных слов из-за специальных символов и регистра. С помощью `sklearn.feature_extraction.text.CountVectorizer` преобразуем обучающую и тестовую выборку к разреженным матрицам, где в позиции `(i, j)` находится количество вхождений слова с номером j в документ i. У конструктора `CountVectorizer` есть параметр `min_df`, отвечающей за минимальное количество документов, в которое должно входить слово j, чтобы попасть в разреженную матрицу.

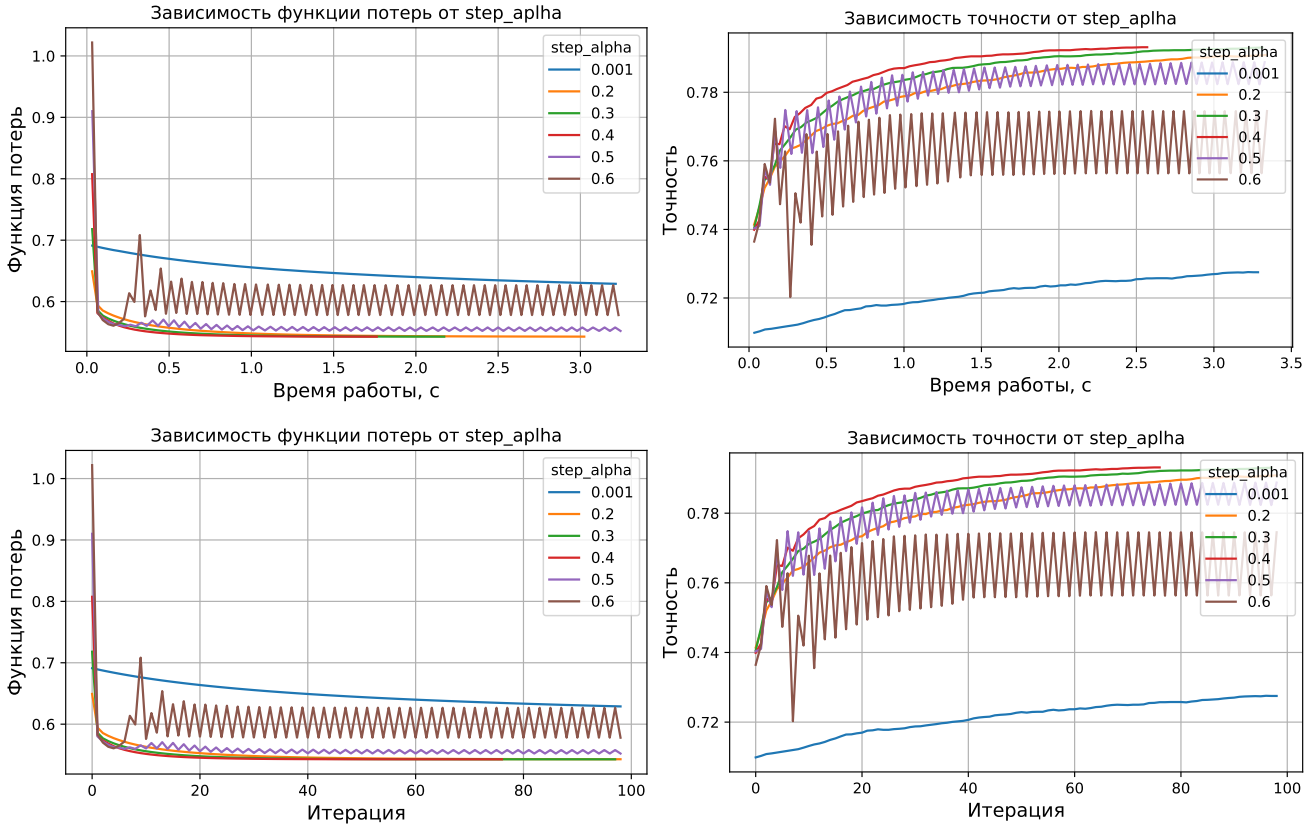


Как видно из графика точность обучения на тестовой выборке резко падает с увеличением `min_df`, поэтому везде далее будет использоваться разрежанная матрица, созданная с параметром `min_df=1` (считалось с параметрами `step_alpha=0.1, step_beta=0.1, tolerance=1e-6, max_iter=1000`).

3.2. Исследование градиентного спуска

Для начала исследуем параметры `step_alpha`, `step_beta` из коэффициента `learning rate` $\eta_k = \alpha/k^\beta$.

Рисунок 3.1. Исследование поведения градиентного спуска в зависимости от параметра step_alpha

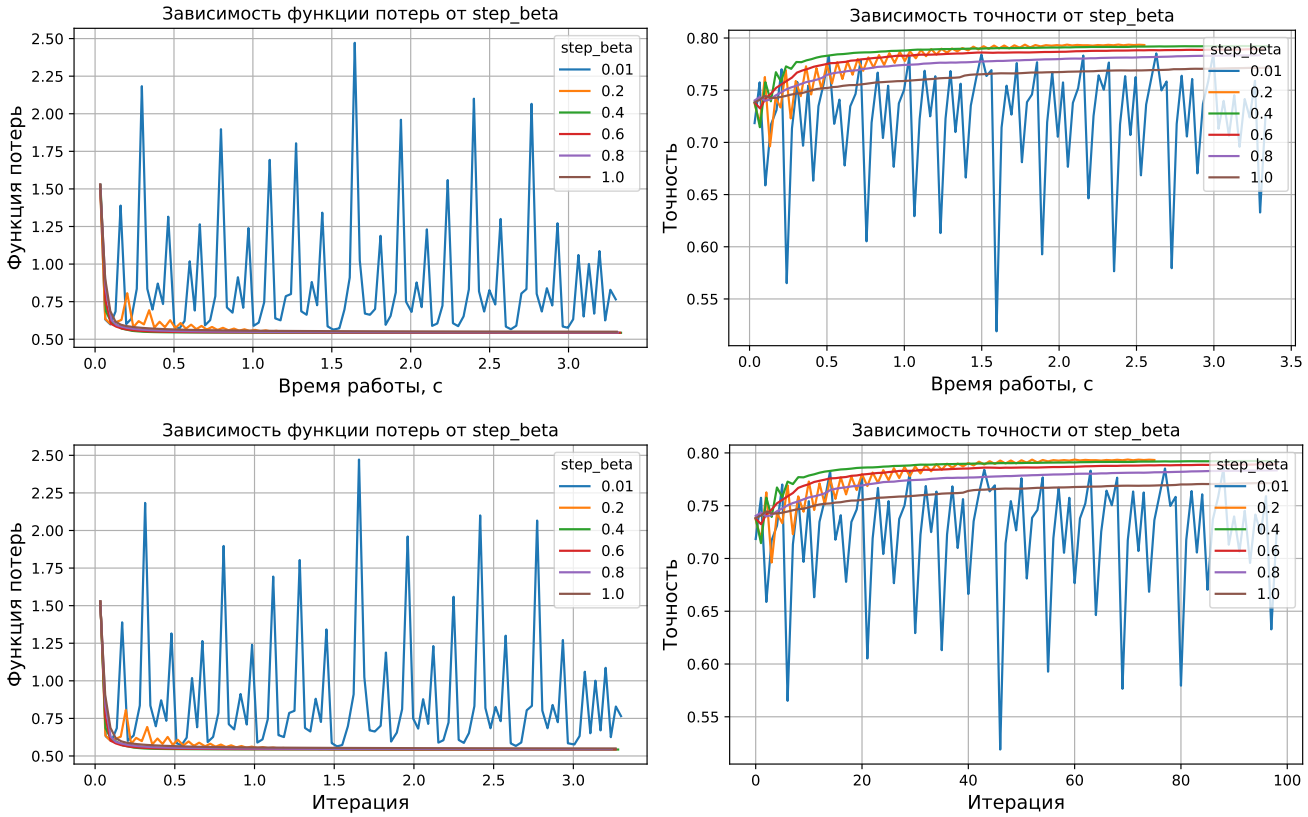


Значения остальных параметров: `step_beta=0, tolerance=1e-6, max_iter=100, l2_coef=0.1`

Видно, что графики с зависимостью времени работы практически не отличается от графиков с зависимостью от итерации. Можно заметить, что для больших значений `step_alpha` значение точности и функции потерь колеблется, из-за того что, спускаясь про градиенту, мы "проскакиваем" минимум функции. Для более маленьких значений модель дольше обучается и дольше достигает значения `tolerance`.

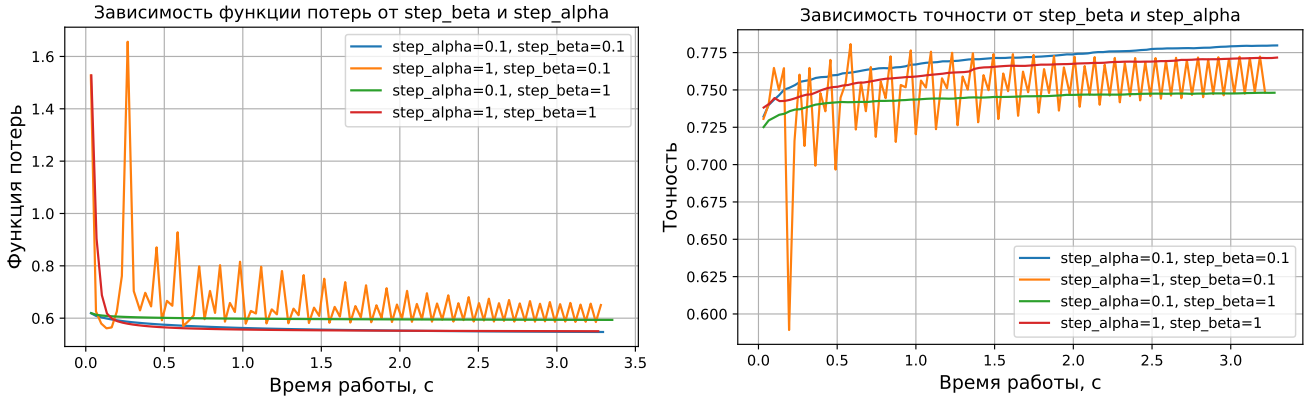
Для маленьких значений `step_beta` есть небольшие колебания из-за того, что коэффициент learning rate медленно убывает. Из графика для точности видно, что если значения `step_beta` будут достаточно маленькими, но не слишком, качество качество на тестовой выборке будет выше. Это связано с тем, что для больших значений вклад градиента в обучение слишком быстро уменьшается.

Рисунок 3.2. Исследование поведения градиентного спуска в зависимости от параметра `step_beta`



Значения остальных параметров: `step_alpha=1, tolerance=1e-6, max_iter=100, l2_coef=0.1`

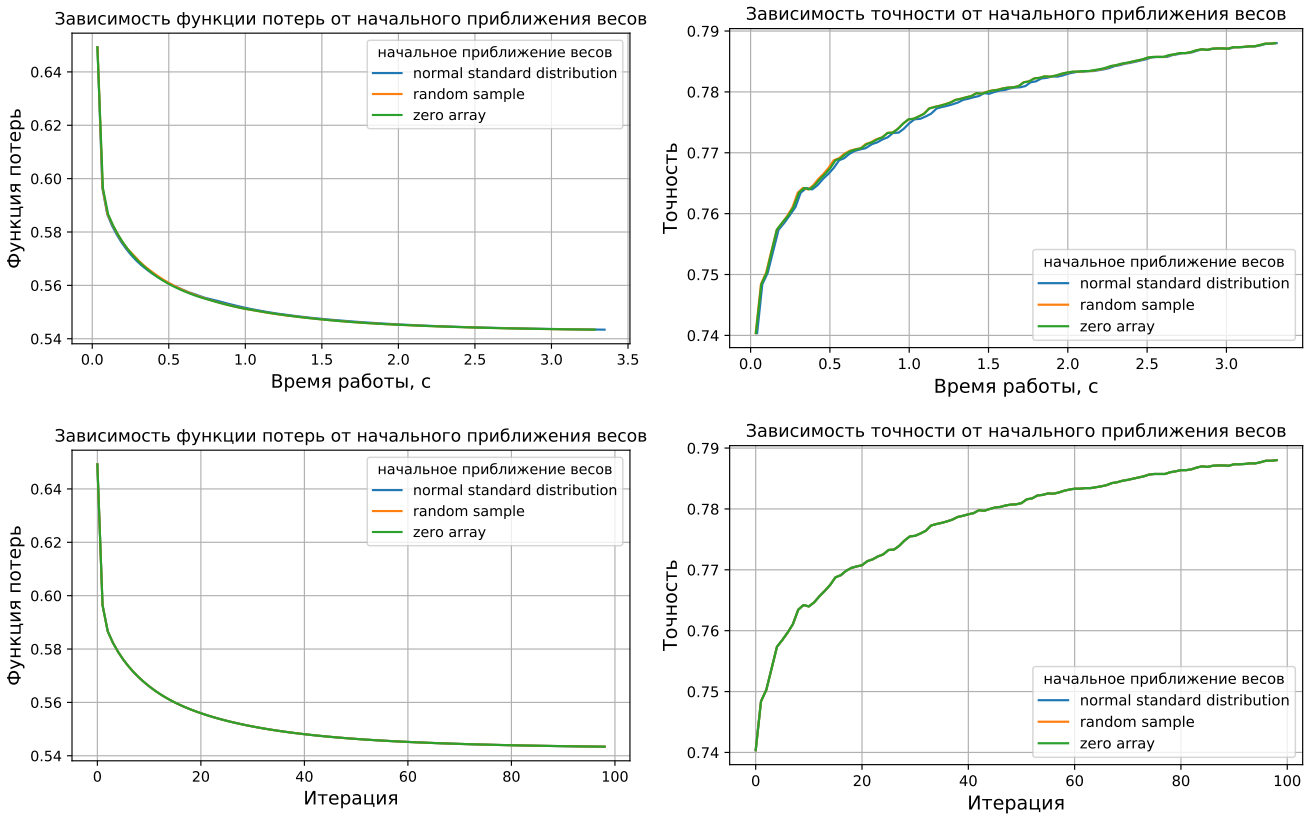
Рисунок 3.3. Исследование поведения градиентного спуска в зависимости от параметров step_alpha и `step_beta`



Значения остальных параметров: `tolerance=1e-6, max_iter=100, l2_coef=0.1`

При большом значении `step_alpha` и маленьком `step_beta` сначала есть колебания, но через большое количсетво итераций кривая выровняется, так как learning rate будет уменьшаться. Для маленького значения `step_alpha` и большого `step_beta` вклад градиента слишком быстро станет маленьким и модель не сможет хорошо обучиться. В остальных случаях функция потерь будет уменьшаться почти одинаково, однако точность будет лучше для маленьких значений `step_alpha` и `step_beta`.

Рисунок 3.4. Исследование поведения градиентного спуска в зависимости от начального приближения весов



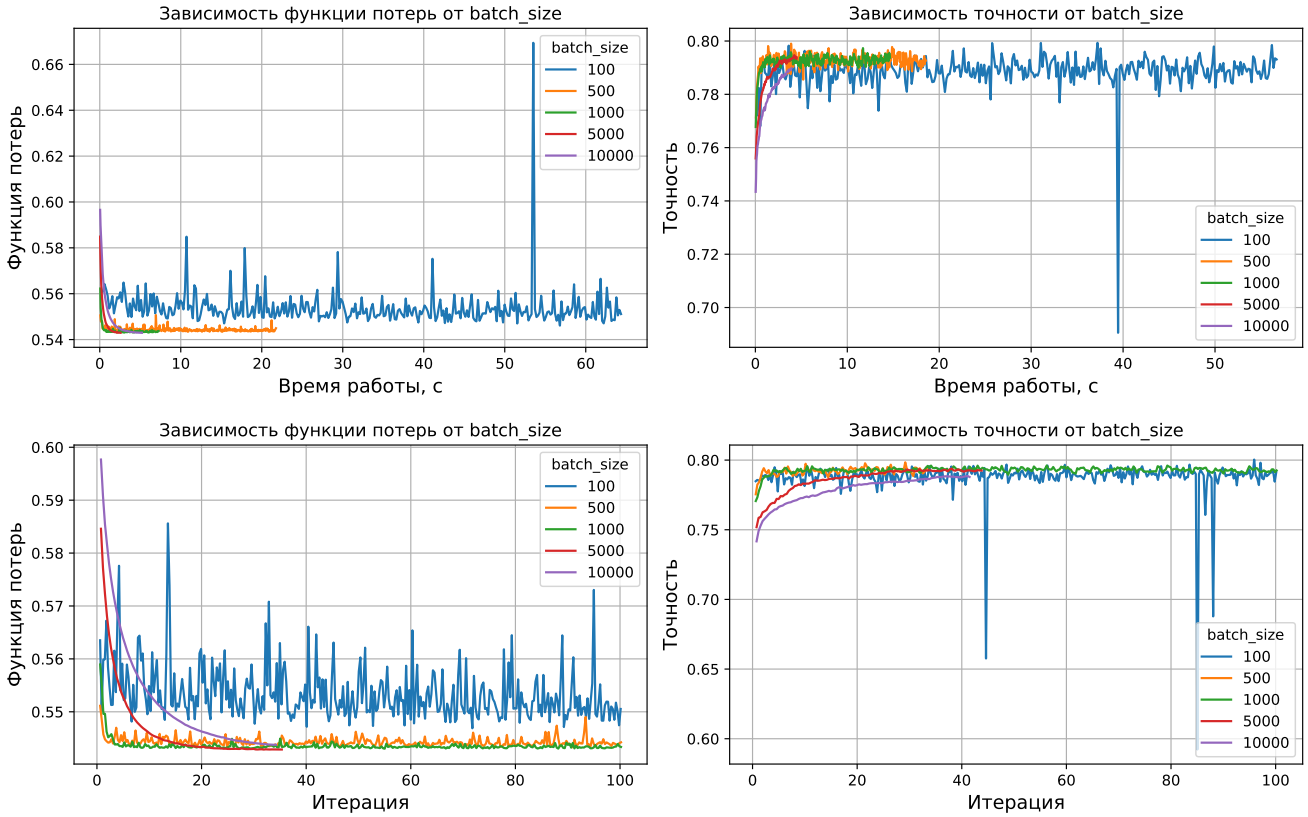
Значения остальных параметров: `step_alpha=0.2, step_beta=0.1, tolerance=1e-6, max_iter=100, l2_coef=0.1`

От начального приближения весов не зависит качество обучения.

3.3. Исследование стохастического градиентного спуска

Сначала исследуем зависимость от параметра `batch_size`. Из-за того, что приходится заново искать градиент на каждой подвыборке размера `batch_size` для маленьких значений метод работает заметно медленнее, чем для больших. При этом для больших значений `batch_size` мтеод начинает себя вести как обычный градиентный спуск.

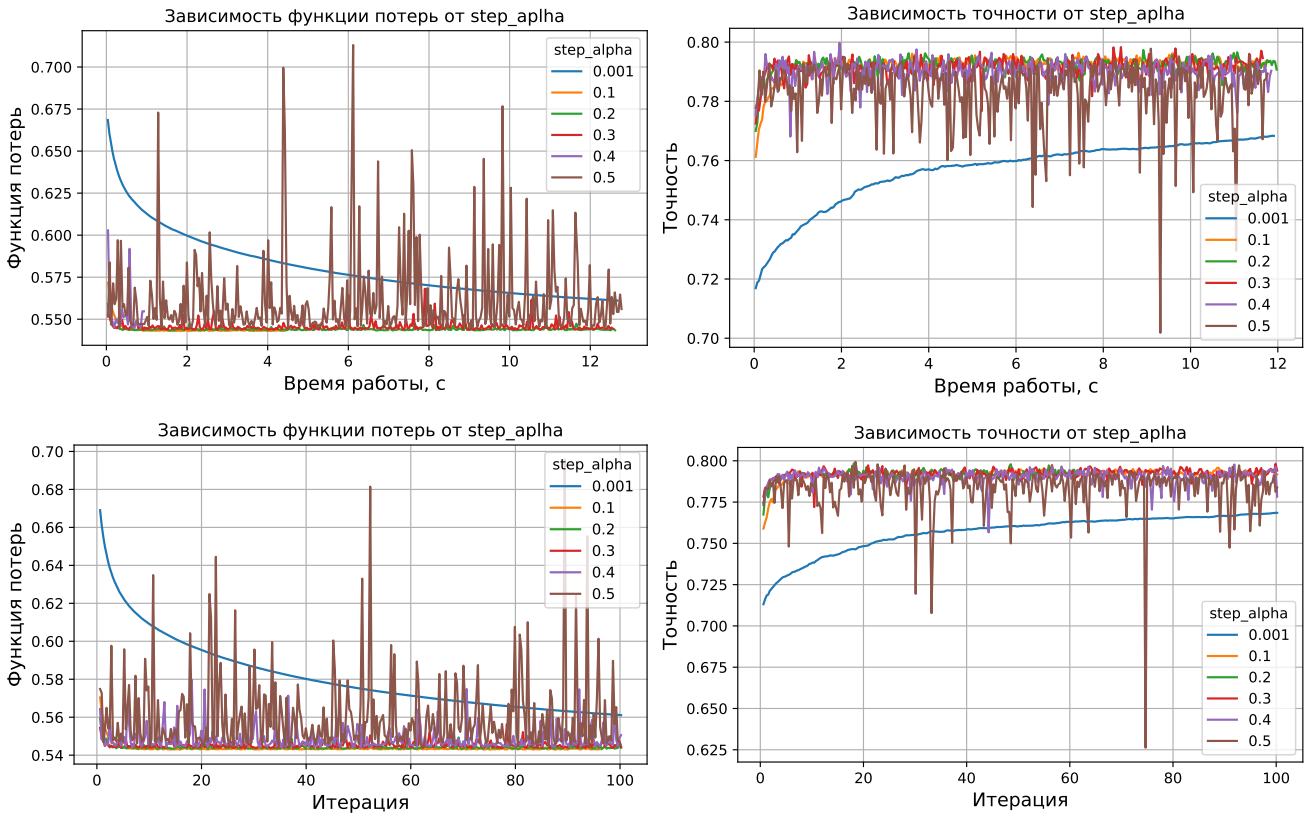
Рисунок 3.5. Исследование поведения стохастического градиентного спуска в зависимости от параметра `batch_size`



Значения остальных параметров: `step_alpha=0.1, step_beta=0.1, tolerance=1e-6, max_iter=100, l2_coef=0.1`

Везде далее будем использовать `batch_size=2000`, чтобы с одной стороны метод не стал совсем похожим на обычный градиентный спуск, с другой - чтобы не так долго обучался.

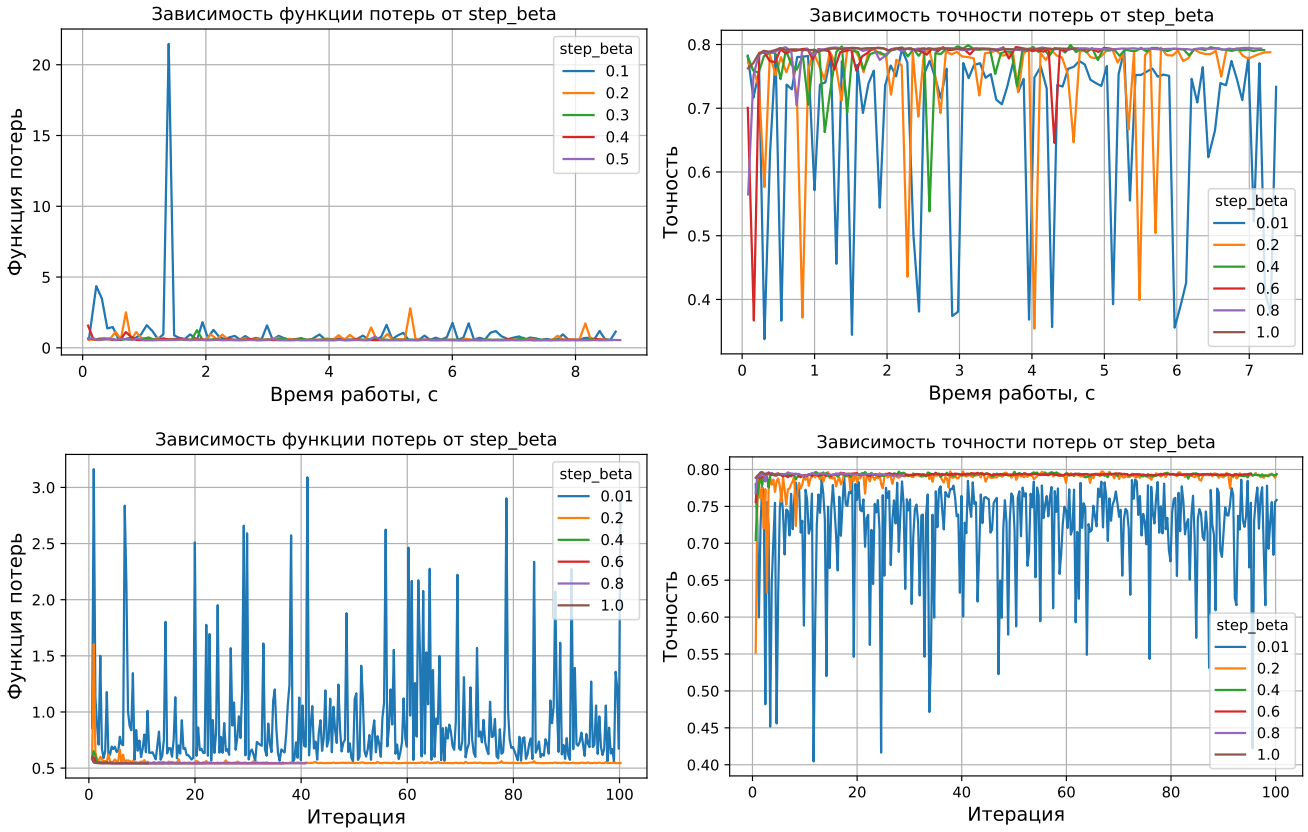
Рисунок 3.6. Исследование поведения стохастического градиентного спуска в зависимости от параметра `step_alpha`



Значения остальных параметров: `batch_size=2000, step_beta=0.1, tolerance=1e-6, max_iter=100, l2_coef=0.1`

При совсем маленьких значениях параметра `step_alpha` нет колебаний из-за того что спуск слишком медленный. При больших значениях `step_alpha` сильно больше колебаний чем при маленьких, и из-за того что `step_beta=0`, эти колебания не затухнут.

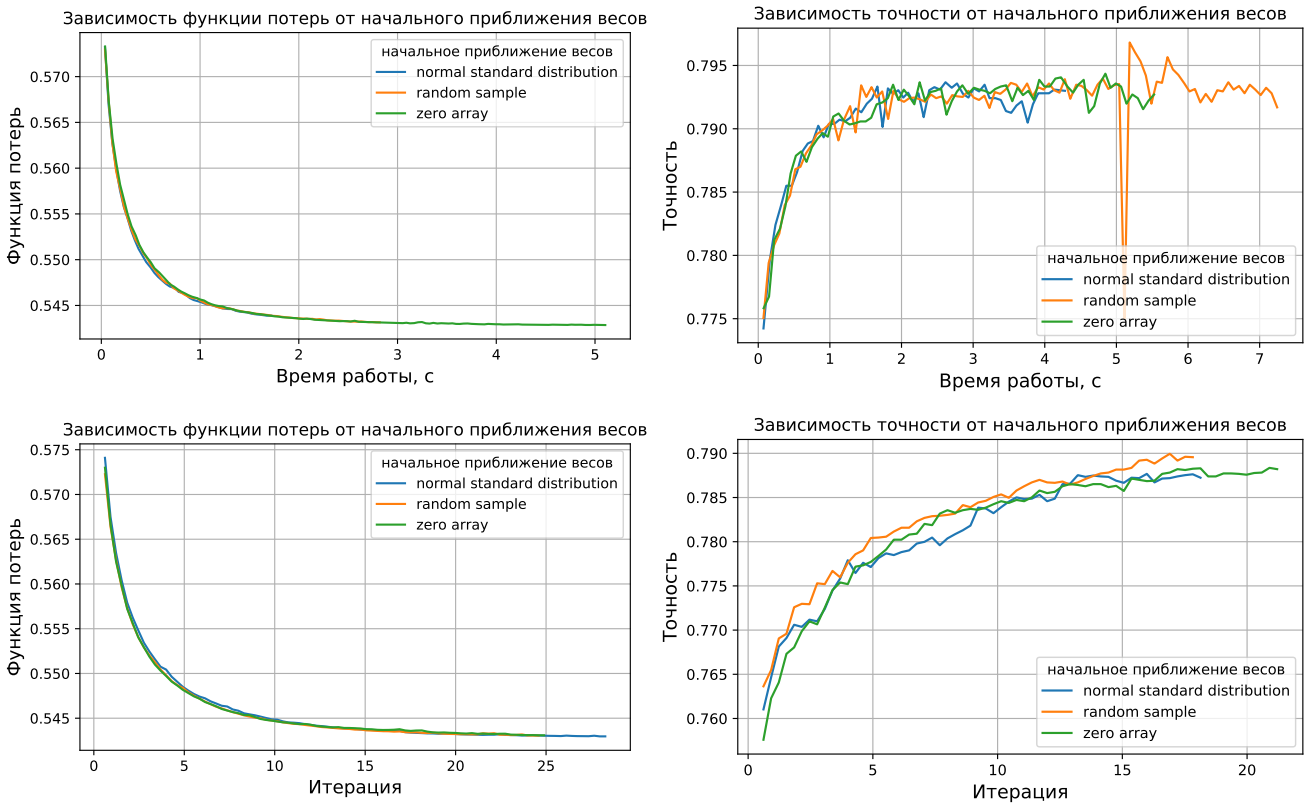
Рисунок 3.7. Исследование поведения стохастического градиентного спуска в зависимости от параметра `step_beta`



Значения остальных параметров: `batch_size=2000, step_alpha=1, tolerance=1e-6, max_iter=100, l2_coef=0.1`

При маленьких параметра `step_beta` колебания очень медленно затухают, при этом модель будет слишком плохо обучаться при больших значений, так как вклад градиента будет слишком быстро убывать.

Рисунок 3.8. Исследование поведения стохастического градиентного спуска в зависимости от начального приближения весов



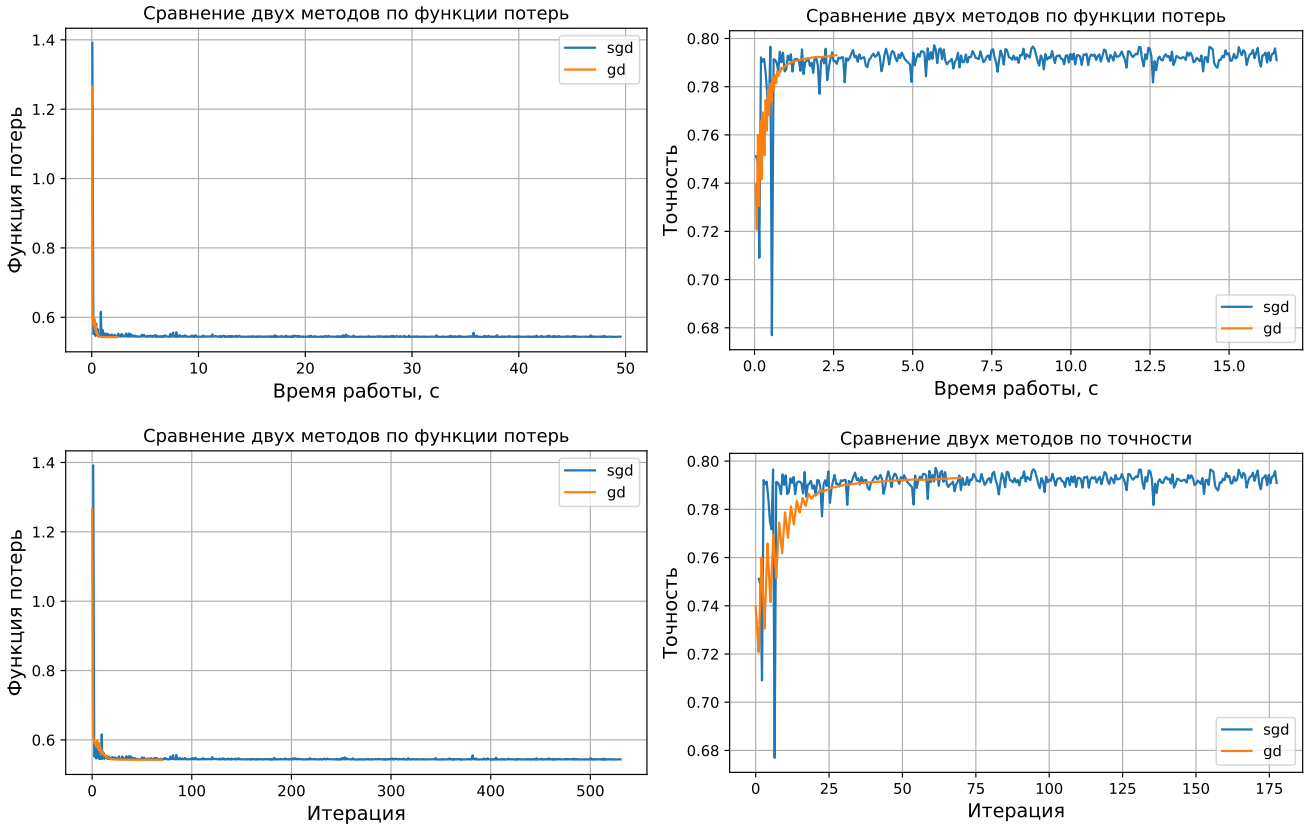
Значения остальных параметров: `batch_size=2000, step_alpha=0.1, step_beta=0.4 tolerance=1e-6, max_iter=100, l2_coef=0.1`

Как и в предыдущем разделе в основном результаты не сильно отличаются, только на рандомно сгенирированном сэмпле точность чуть получше.

3.4. Сравнение градиентного спуска и стохастического градиентного спуска

Видно, что на стохастическом спуске модель дольше обучается, при этом точность сильно колеблется, и итоговое точность на градиентном спуске оказывается выше средней точности на стохастическом. Поэтому для дальнейших экспериментов будет использоваться градиентный спуск.

Рисунок 3.9. Сравнение градиентного спуска и стохастического градиентного спуска

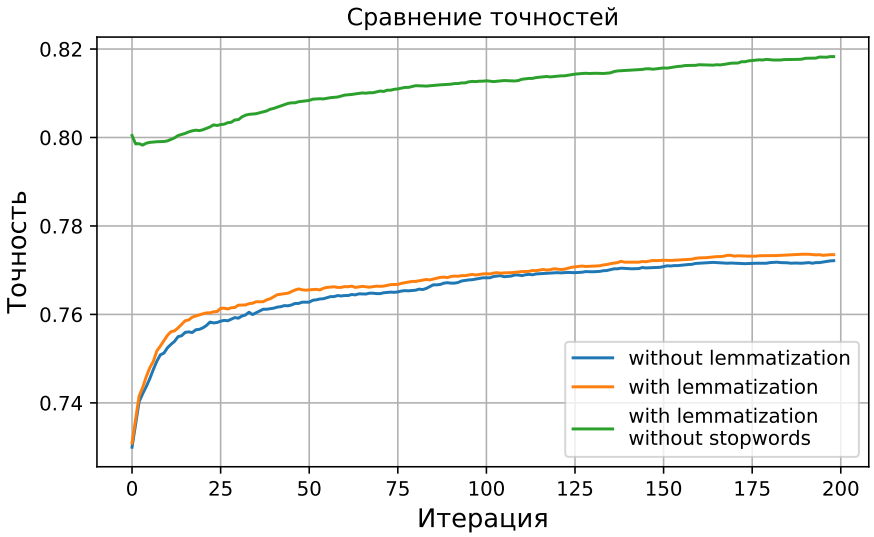


Значения остальных параметров: `step_alpha=0.8, step_beta=0.2, tolerance=1e-6, max_iter=2000, l2_coef=0.1, batch_size=200`

3.5. Лемматизация и удаление стоп-слов

Стоп слова увеличивают размерность, но при этом не несут в себе содержательной информации о токсичности комментария, поэтому при их удалении точность значительно возросла. Лемматизация немного снижает размерность без потери исходного смысла предложений, поэтому она тоже улучшает немного точность прогноза.

Рисунок 3.10. Зависимость точности от обработки документов



Остальные параметры градиентного спуска: `step_alpha=0.1, step_beta=0.4, tolerance=1e-6, max_iter=200, l2_coef=0.1`

	default	lemmatization	stop-words
размерность	89368	82991	82851
время работы, с	6.95	6.71	5.22

Таким образом, наилучшим методом является градиентный спуск с лемматизацией и удалением стоп-слов.

3.6. Анализ ошибок

Частые ошибки определения токсичности комментария:

- Есть токсичные слова и не токсичные: `'dear god this site is horrible', 'eek but shes cute in an earthy kind of way cant sing for shit though thanks for giving me an unhappy memory'`
- Слова не имеющие смысла: `'puwersa ng masa', 'aapn bhtla aanand jhala'`
- Потеря нейтрального смысла слов, из-за часто встречающегося токсичного использования: `'i am not joking', 'black mamba it is ponious snake of the word and but it not kills many people but king cobra kills many people in india'`

Модель посчитала токсичный комментарий нетоксичным 1124 раза, а наоборот - 2472 раза. Чаще всего ошибается из-за использования не английских слов и смеси в одном предложении слов разных категорий.

4. Заключение

Получилось достаточно хорошо обучить модель для распознавания токсичности комментариев. Для улучшения модели были использованы различные обработки текстов и исследовано поведение градиентного спуска. Лучший результат по точности на тестовой выборке - 0.826.

5. Литература

- https://stackoverflow.com/
- Документации библиотек numpy, matplotlib, sklearn и др.
- Шаблон отчета в LaTeX