Отчет о выполненной работе «Применение линейных моделей для определения токсичности комментария»

Федоров Илья Сергеевич
курс «Практикум на ЭВМ» ММП ВМК МГУ

20 ноября 2019 г.

Постановка задачи

В задании требовалось реализовать логистическую регрессию с помощью метода градиентного спуска и применить её для классификации комментариев: является ли комментарий токсичным или нет. Помимо этого требовалось выполнить серию экспериментов для исследования качества классификации в зависимости от параметров и различных технологий преобразования данных.

Теоретические основы

Напомним, что в логистической регрессии используется следующая функция потерь:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}\log\left(1 + e^{-y_i\langle x_i w\rangle}\right)$$

Здесь n - число объектов в обучающей выборке X, x_i - i-й объект, $y_i \in \{-1, 1\}$ - класс i-го объекта, w - вектор весов. Отметим, что мы предполагаем, что в множество признаков добавлен константный единичный признак, чтобы не писать отдельно свободный член b.
С целью борьбы с переобучением используемая нами модель будет использовать l_2 - регуляризацию. С её учетом функция потерь примет вид:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}\log\left(1 + e^{-y_i\langle x_i w\rangle}\right) + \frac{\lambda}{2}\|w\|^2$$

С помощью метода матричного дифференцирования, найдем градиент этой функции:

$$dL(w) = \frac{1}{n}\sum_{i=1}^{n} d\log\left(1 + e^{-y_i\langle x_i w\rangle}\right) + \frac{\lambda}{2}d\langle w, w\rangle = \langle -\frac{1}{n}\sum_{i=1}^{n}\frac{y_i}{1 + e^{y_i\langle x_i w\rangle}}x_i + \lambda w, dw\rangle$$

Следовательно, градиент функции потерь равен

$$\nabla L(w) = -\frac{1}{n}\sum_{i=1}^{n}\frac{y_i}{1+e^{y_i\langle x_i w\rangle}}x_i + \lambda w$$

Также существует [1] обобщение логистической регресии на случай многих классов - мультиномиальная регрессия. В этом случае строится k линейных моделей $\alpha_1(x), \alpha_2(x), \ldots, \alpha_k(x)$, где k - число классов, и модель $\alpha_i(x)$ выдает оценку принадлежности объекта x к i-му классу. Эти оценки можно перевести в вероятностное распределение с помощью функции

$$\text{softmax}(x_1, \ldots, x_k) = \left(\frac{e^{x_1}}{\sum_{i=1}^{k} e^{x_i}}, \ldots, \frac{e^{x_k}}{\sum_{i=1}^{k} e^{x_i}}\right)$$

Вероятность принадлежности объекта x к j-му классу тогда можно выразить как:

$$\mathbb{P}(y = j|x) = \frac{e^{\langle w_j, x\rangle}}{\sum_{i=1}^{k} e^{\langle w_i, x\rangle}}$$

Применив метод максимального правдоподобия, мы придем к следующей функции ошибки (с учетом l_2 - регуляризации):

$$Q(X, w) = -\frac{1}{n}\sum_{i=1}^{n}\log \mathbb{P}(y_i|x_i) + \frac{\lambda}{2}\sum_{i=1}^{k}\|w_i\|_2^2 \to \min_{w_1,\ldots,w_k}$$

Поскольку градиентом в этом случае будет матрица, найдем «градиент» по фиксированному вектору w_m с помощью формул матричного дифференцирования.

$$\frac{dQ}{dw_m} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{e^{\langle w_m, x_i\rangle}}{\sum_{j=1}^{k} e^{\langle w_j, x_i\rangle}} - [y_i = m]\right)x_i + \lambda w_m$$

Заметим, что в случае $k = 2$, задача мультиномиальной логистической регрессии сводится к бинарной логистической регрессии. Действительно, в этом случае

$$\mathbb{P}(y = 1|x) = \frac{e^{\langle w_1, x\rangle}}{e^{\langle w_1, x\rangle} + e^{\langle w_2, x\rangle}} = \frac{1}{1 + e^{-\langle w_1 - w_2, x\rangle}}$$

$$\mathbb{P}(y = 2|x) = \frac{e^{\langle w_2, x\rangle}}{e^{\langle w_1, x\rangle} + e^{\langle w_2, x\rangle}} = \frac{1}{1 + e^{\langle w_1 - w_2, x\rangle}} = 1 - \mathbb{P}(y = 1|x)$$

Как видим, модель приняла вид классической логистической регрессии с вектором весов $w_1 - w_2$. Теперь обратим внимание на функцию ошибки для мультиномиальной регрессии. Исключим из неё слагаемое, отвечающее за регуляризацию, явно выпишем вероятности (как мы их расписали выше), и будем считать, что классы закодированы числами $+1$ и -1, а также вектор весов w_1 соответствуется модели α_1, оценивающую вероятность принадлежность объекта к классу $+1$. Получим выражение, в точности совпадающее с функцией ошибки для классической логичестической регрессии без регуляризации с вектором весов $w_1 - w_2$. Отметим, что в случае использования

регуляризации (во всяком случае, l_2 - регуляризации) задача мультиномиальной логистической регрессии не эквивалентна классической логистической регрессии с регуляризацией, в том смысле, что решения соответствующих оптимизационных задач могут отличаться. Для демонстрации этого см. приложенный jupyter notebook с названием counterexample.ipynb. К сожалению, мне не удалось найти статьи со строгим доказательством этого наблюдения, но оно, вероятно, существует.

Замечание. Хотя в указанных выше формулах это не учтено, на практике не рекомендуется [2] учитывать в регуляризации вес, отвечающий смещению (тот самый, который соответствует константному единичному признаку). Это вполне логично: зачем нам штрафовать модель за большой сдвиг? В нашей задаче данные расположены в окрестности нуля, и это не так важно, однако в реализации эта рекомендация все-таки учтена, что дает небольшой выигрыш в точности.

Для численного решения оптимизационной задачи будем использовать градиентный спуск:

$$f(w_{n+1}) = f(w_n) - \frac{\alpha}{n^\beta}\nabla f(w_n)$$

В дальнейшем параметры α и β будем обозначать step_alpha и step_beta.

Эксперименты

Градиентный спуск

В ходе предварительных экспериментов было установлено, что для проведения дальнейших исследований при преобразовании комментариев в слова с помощью класса sklearn.feature_extraction.text.CountVectorizer оптимально выбрать параметр данного класса min_df = 0.0001. Этот параметр отвечает за количество слов в словаре: будут проигнорированы те слова, которые встречаются в менее чем 0.01% документах. Такое значение позволяет иметь в наборе данных достаточно признаков для качественной классификации, в то же время сохраняя достаточно высокую скорость обучения (если брать в словарь абсолютно все слова, то его размер составит более 80 тысяч слов, а при установленно min_df = 0.0001 - около 16 тысяч). Однако в конце приведенного исследования мы также более подробно рассмотрим влияние данного параметра на различные характеристики модели.

Перебором были обнаружены достаточно хорошие параметры «базовой» модели: регуляризация отключена, step_alpha=2.5, step_beta=0.0, tolerance=1e-8. На 5000 итераций данная модель имеет качество accuracy на тестовой выборке около 0.886. Стоит отметить, что классы в обучающей, и в тестовой выборке не являются сбалансированными: токсичными являются приблизительно 30% объектов. Поэтому помимо доли правильных ответов мы также будем исследовать показатель ROC-AUC, который более точно отражает качество классификации в случае несбалансированных классов. Наша базовая модель оценивается ROC-AUC в 0.945, что также является достаточно хорошим показателем.

Проведем более детальное исследование зависимости качества классификации с точки зрения метрик accuracy и ROC-AUC от параметров модели step_alpha и step_beta. Будем перебирать эти параметры на двумерной сетке: step_alpha будет изменяться от 0.1 до 5.0 с шагом 0.5, step_beta от 0.0 до 2.0 с шагом 0.3, регуляризация отключена, а для ускорения обучения уменьшим число итераций до 700. Такие данные удобно представить в виде heatmap.

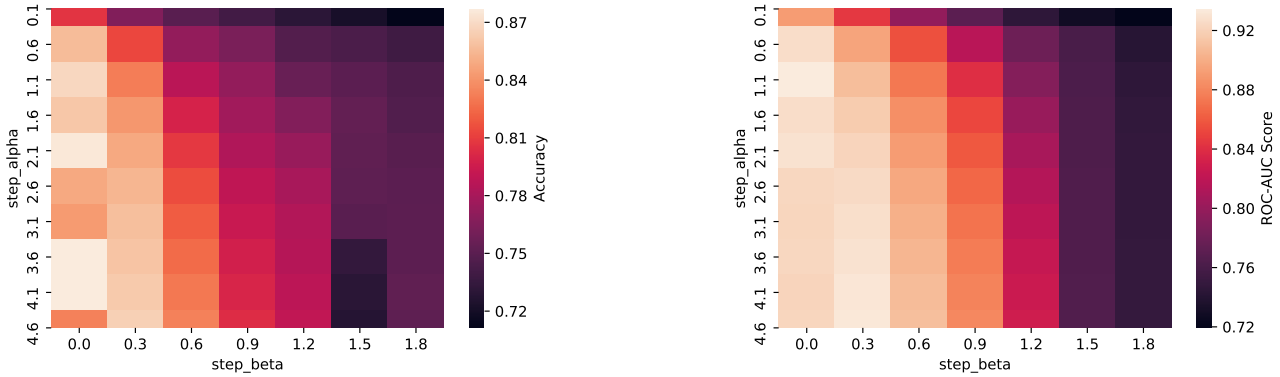


Рис. 1: Метрики качества классифкации в зависимости от параметров модели

Из этих heatmap можно сделать вывод, что качество повышается с ростом step_alpha и уменьшается с ростом step_beta. Несложно понять, почему увеличение step_beta ведет к ухудшению качества: коэффициент перед градиентом начинается слишком быстро стремиться к нулю, и градиентный спуск просто не успевает сойтись к оптимуму, поэтому нет смысла исследовать показатели метрик для ещё больших значений step_beta. Кроме того, отметим, что качество классификации моделей, для которых step_beta $= 0$, хуже, чем для моделей, использующих step_beta $= 0.3$. Более того, для моделей, не учитывающих в коэффициенте перед градиентом номер итерации, значение метрик в зависимости от step_alpha изменяется достаточно резко. Поскольку мы наблюдаем повышение качества с ростом step_alpha, рассмотрим качество модели на сетке, в которой step_alpha изменяется уже от 5.0 до 10.0 с шагом 0.5, а step_beta в тех же границах.

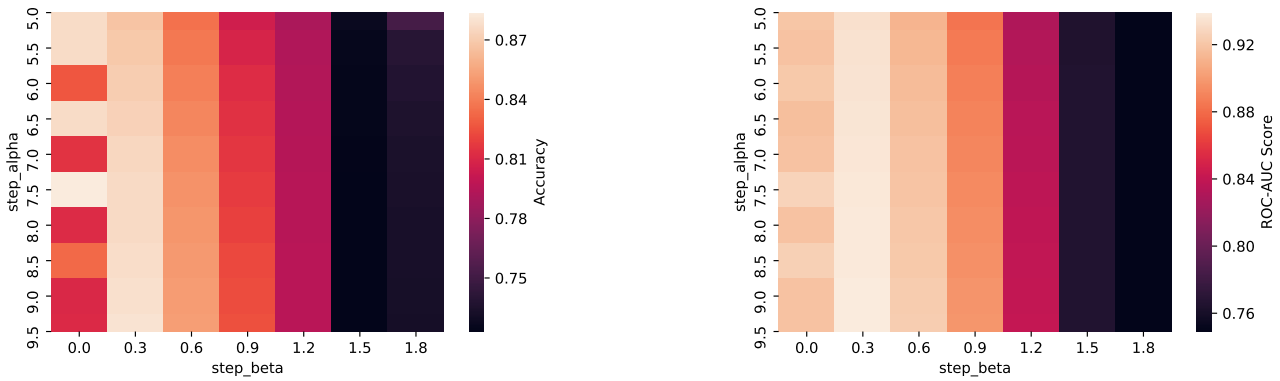
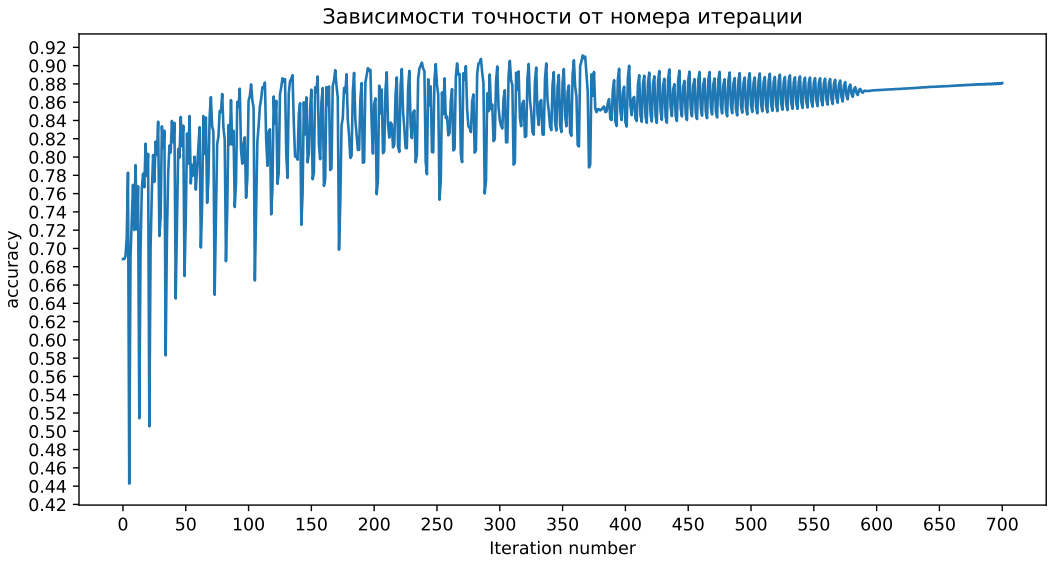


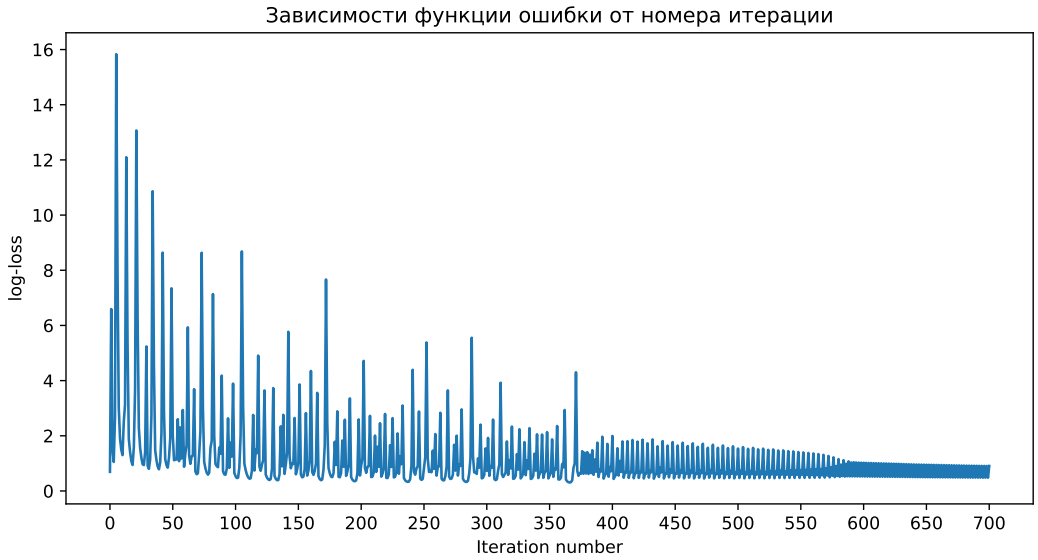
Рис. 2: Метрики качества классифкации в зависимости от параметров модели

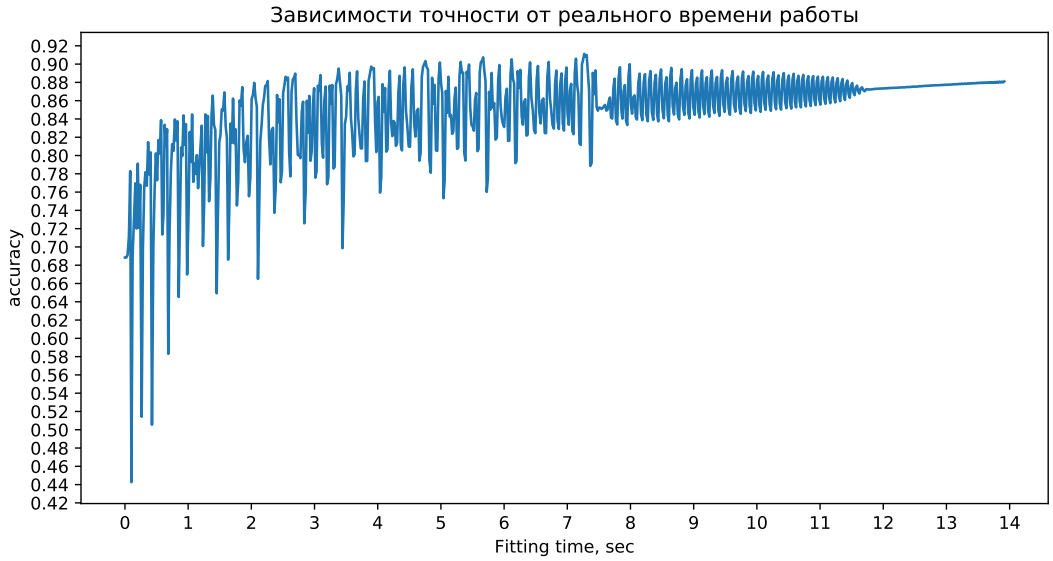
Как видим, дальнейшее увеличение параметра step_alpha не способствует существенному повышению качества классификации. По рис. 2 можем оценить оптимальное значение этого параметра примерно в 7.0, для step_beta = 0.3.

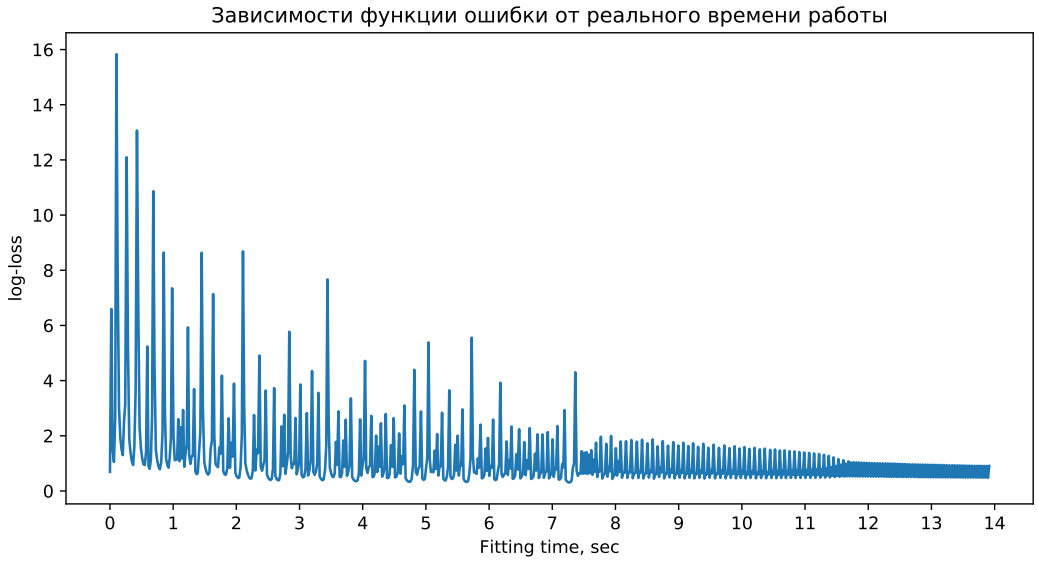
Исследуем теперь следующие характеристики модели в зависимости от этих же параметров: зависимость значения функции потерь и точности от времени работы и от номера итерации. Нет смысла рассматривать колоссальное количество графиков для каждой из построенных моделей, посмотрим лишь на графики, отвечающие моделям, для которых параметры достаточно сильно отличаются между собой.

Рассмотрим для начала описанные выше зависимости для модели, не учитывающей номер итерации (step_beta = 0), при step_alpha = 5.



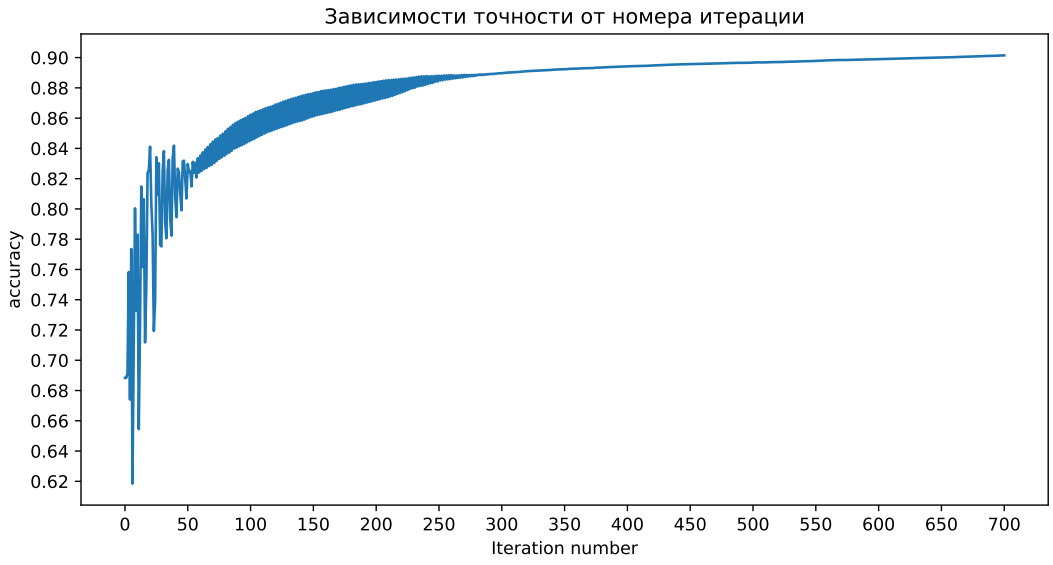

Зависимости функции ошибки от номера итерации
log-loss
Iteration number
0
2
4
6
8
10
12
14
16
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700

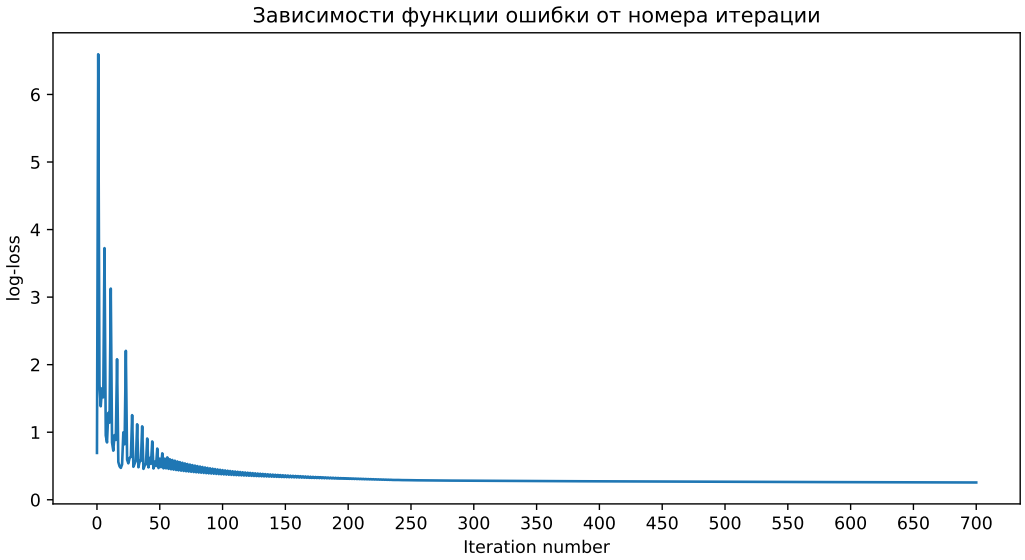

Зависимости точности от реального времени работы
accuracy
Fitting time, sec
0.42
0.44
0.46
0.48
0.50
0.52
0.54
0.56
0.58
0.60
0.62
0.64
0.66
0.68
0.70
0.72
0.74
0.76
0.78
0.80
0.82
0.84
0.86
0.88
0.90
0.92
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

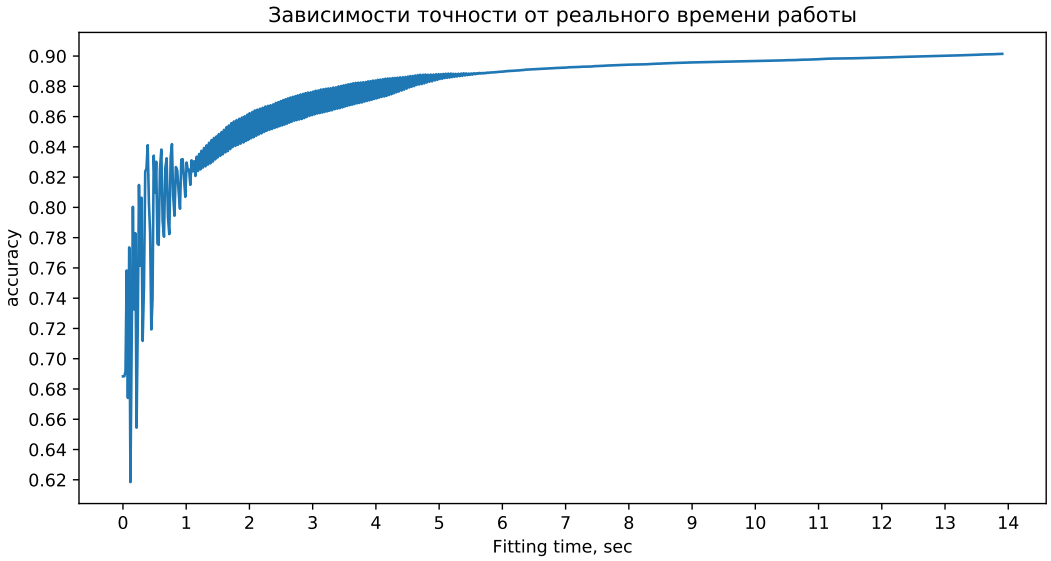


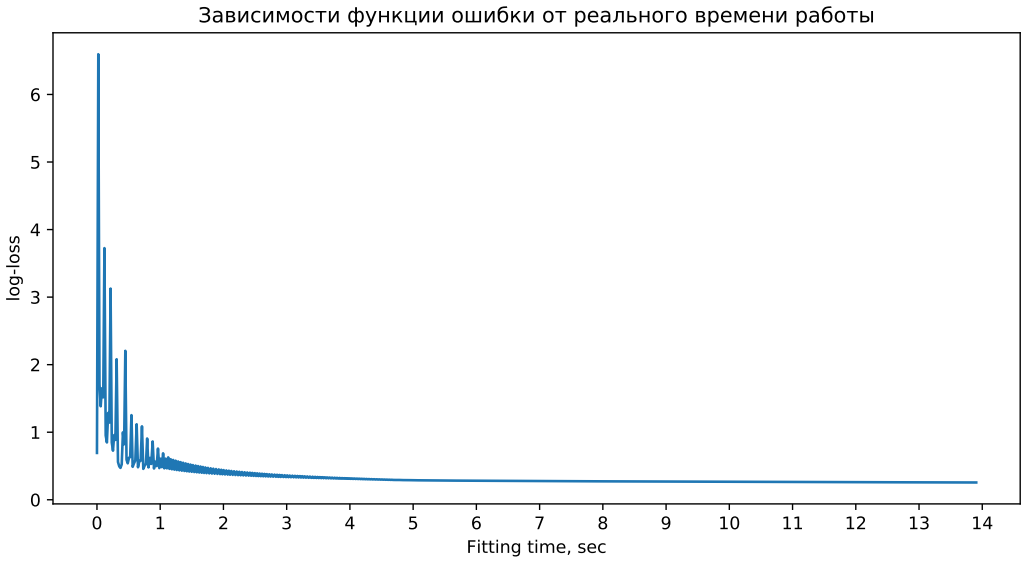
Из этих графиков можно сделать вывод, что если не уменьшать коэффициент перед градиентом в зависимости от номера итерации, то решение получается крайне неустойчивым: сходимость хоть и есть, но из-за того что шаг не уменьшается, очень медленная.

Теперь рассмотрим графики для подобранной по heatmap'ам оптимальной модели: step_alpha=5.0, step_beta=0.3.



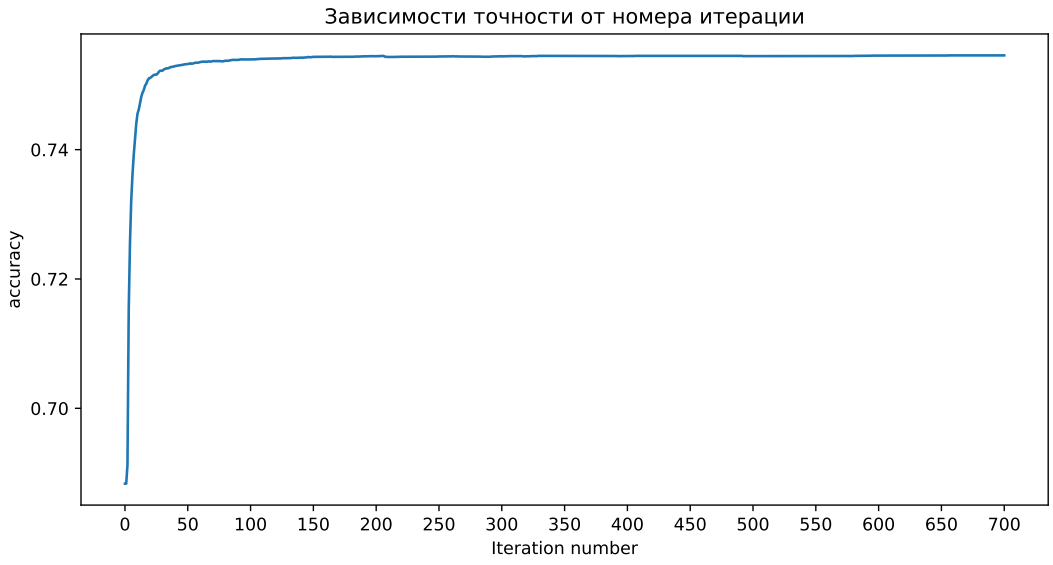

Зависимости функции ошибки от номера итерации
log-loss
Iteration number
0
1
2
3
4
5
6
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700

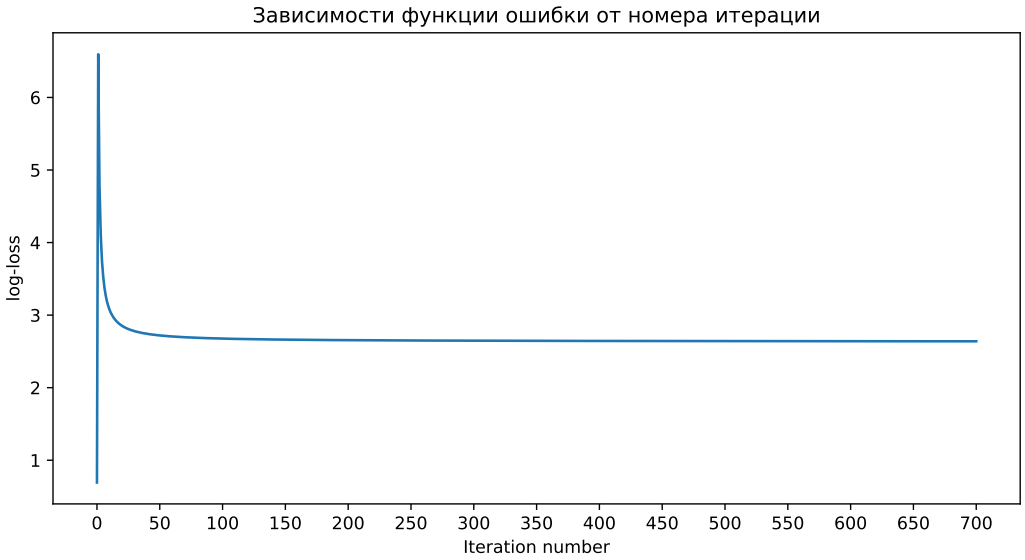

Зависимости точности от реального времени работы
accuracy
Fitting time, sec
0.62
0.64
0.66
0.68
0.70
0.72
0.74
0.76
0.78
0.80
0.82
0.84
0.86
0.88
0.90
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

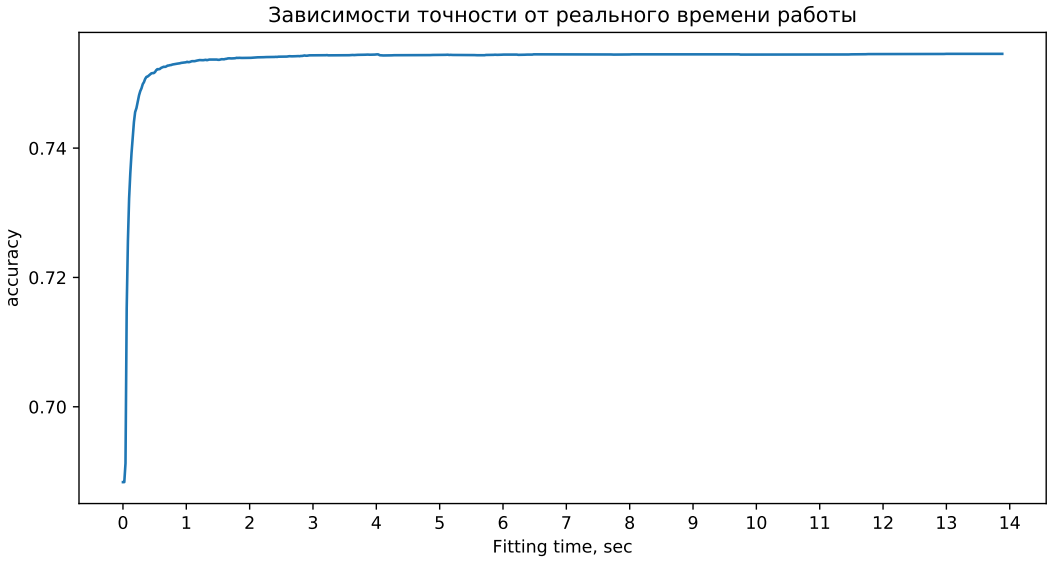


Получаем вполне ожидаемый результат: модель стала гораздо более устойчивой, сходимость к оптимальному решению достаточно быстрая.

Наконец, рассмотрим графики характеристик модели, у которой коэффициент перед градиентом уменьшается пропорционально квадрату номера итерации: step_alpha=5.0, step_beta=2.0.



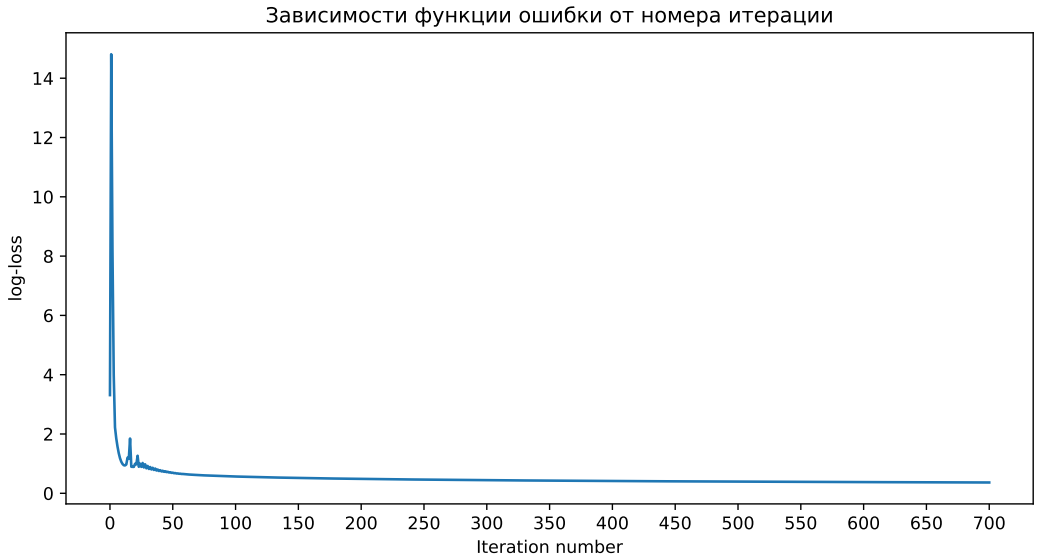

Зависимости функции ошибки от номера итерации
log-loss
Iteration number
1
2
3
4
5
6
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700


Зависимости точности от реального времени работы
accuracy
Fitting time, sec
0.70
0.72
0.74
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

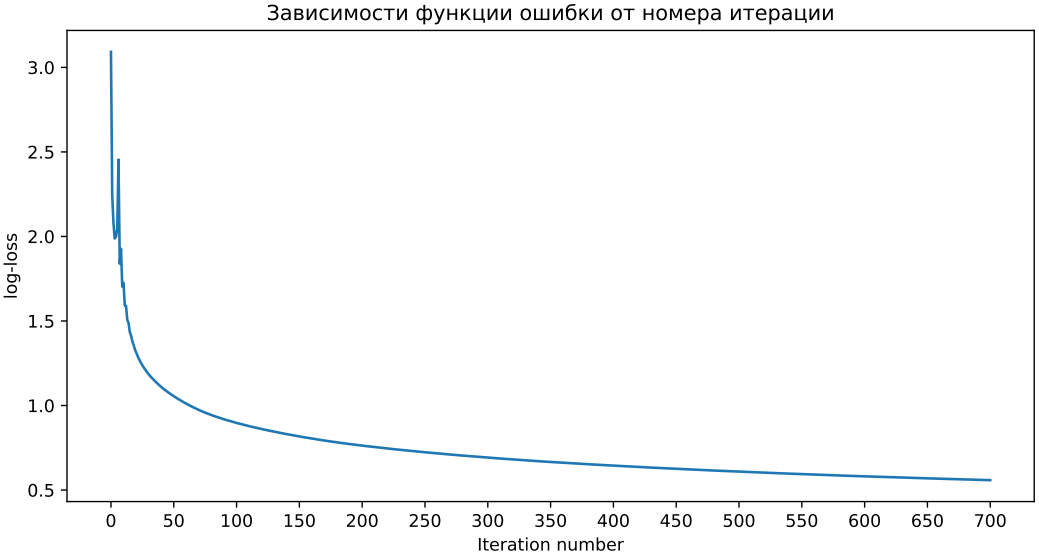
Как видим, графики очень быстро стабилизируются, но на точке, которая не является оптимальной. Это можно понять, сравнив значения функции ошибки и точности для этой модели и оптимальной модели. Значение log-loss для модели с квадратичным убыванием коэффициента перед градиентом стабилизируется на числе, чуть меньшем 3, в то время как оптимальная модель стабилизирует это значение примерно на 0.5.

Исследуем теперь, как изменяются эти графики в зависимости от начального приближения. Во всех рассматриваемых выше моделях начальное приближение вектора весов w_0 было нулевое. Будем использовать модель с параметрами step_alpha=5.0, step_beta=0.3. Попробуем семплировать начальное приближение из различных вероятностных распределений, и смотреть, как будет изменяться график зависимости значения функции ошибки от номера итерации.

Для начала попробуем равномерное распределение: $w_0 \sim U[-1, 1]$. Получим следующий график:



Заметим, что на первом шаге градиентного спуска ошибка сильно увеличилась, и лишь потом устремилась к нулю. Это может указывать на то, что начальное положение было достаточно далеко от оптимальной точки, и алгоритму пришлось идти к ней через точки с большим значение функции ошибки. Качественной причиной этому может служить то, что в нашей модели веса совершенно точно не будут распределены равномерно, ведь, очевидно, большинство слов в комментариях будут играть нейтральный характер, поэтому должны иметь вес, близкий к нулю. Из эти соображений попробуем семплировать w_0 из стандартного нормального распределения $w_0 \sim \mathcal{N}(0, 1)$. Получим следующий график:



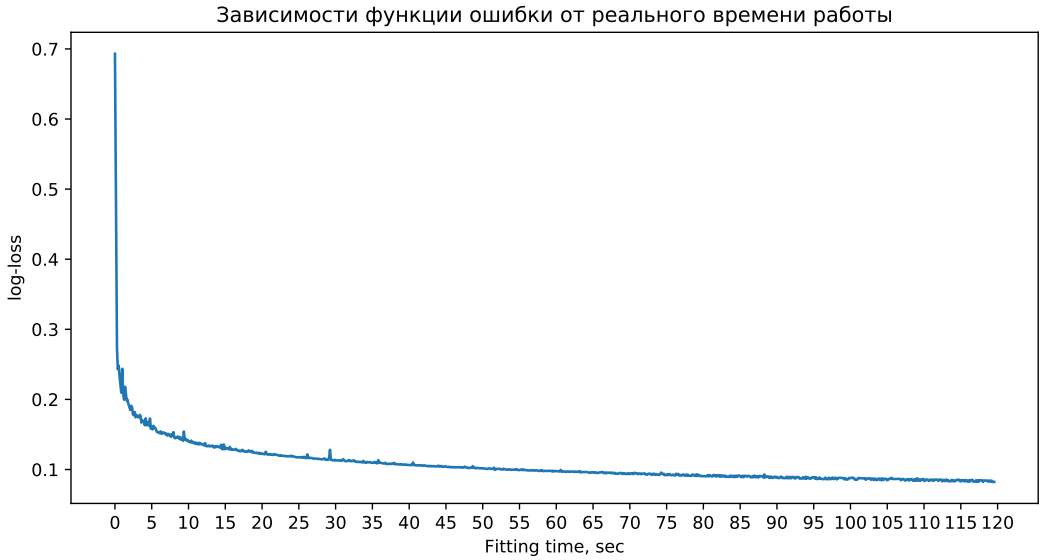
Как видим, наше априорное предположение значительно улучшило поведение градиентного спуска в течение его первых итераций.

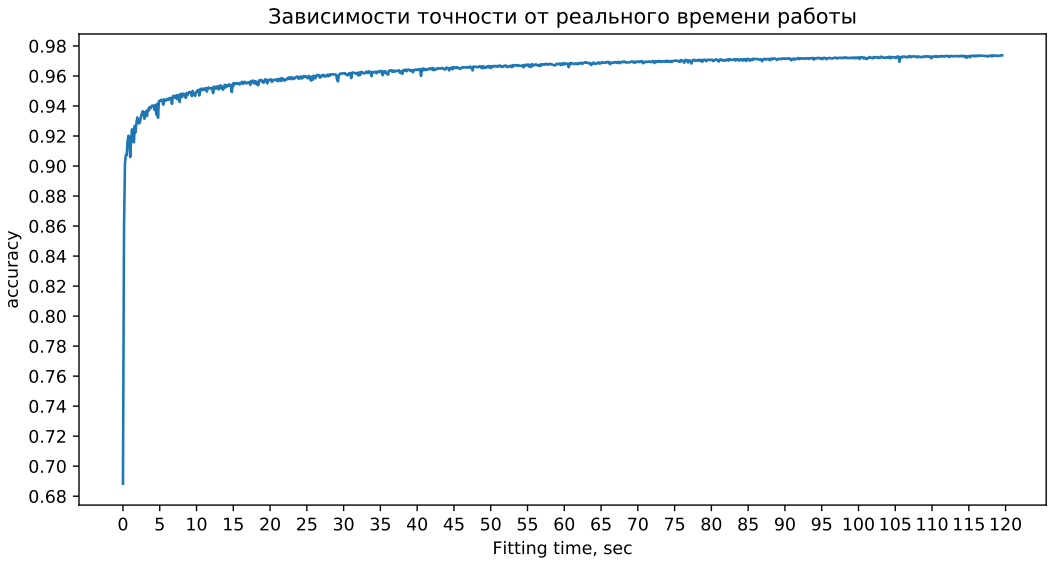
Стохастический градиентный спуск

Исследуем поведение стохастического градиентного спуска от размера батча. Стоит заметить, что коэффициент перед градиентом меняется после каждого батча, а не после каждой эпохи, поэтому в дальнейшем мы так же, как и для обычного градиентого спуска, рассмотрим влияние параметров step_alpha и step_beta на качество модели. Однако пока что будем использовать полученные выше параметры step_alpha = 5.0, step_beta = 0.3. Поскольку количество эпох будет неизменно (1000), нас интересует зависимость точности и функции ошибки от реального времени выполнения метода. Возьмем в качестве базовой модели SGD с размером батча в 5000 объектов. Получим следующие результаты:

Отметим, что графики стали более плавные, в сравнении с моделью, использующей

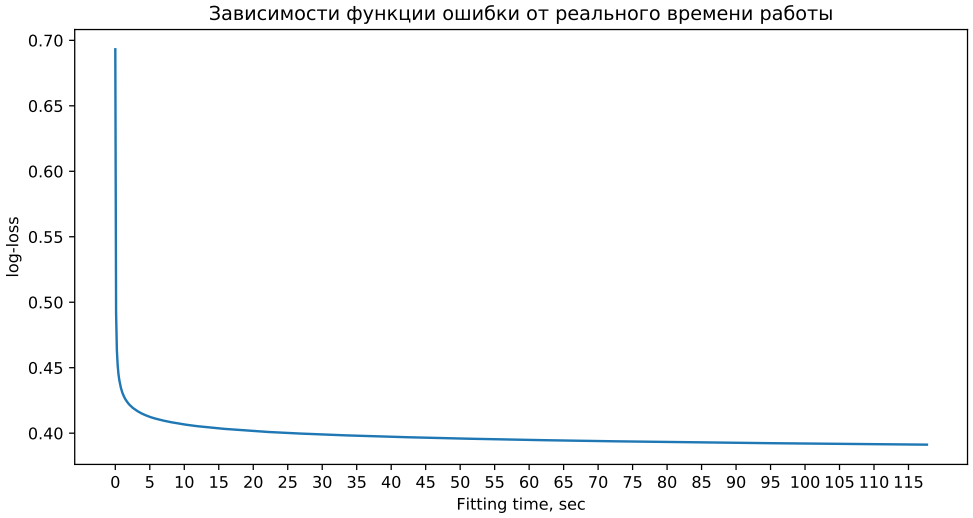
градиентный спуск с параметрами step_alpha=5.0, step_beta=0.3. Это связано с тем, что коэффициент перед градиентом уменьшается на каждом батче, поэтому модель не может сильно перешагивать через точку минимума. В то же время скорость убывания не квадратичная (как в примере, который мы рассмотрели выше), поэтому с течением времени модель сходится к оптимуму, а не останавливается на половине пути из-за слишком быстро убывающего шага. Кроме того, ошибка достаточно монотонно стремится к нулю, мы не наблюдаем резких скачков в начале, как это было в случае обычного градиентого спуска. Причиной этому является как описанное выше, так и тот факт, что внутри каждой эпохи мы делаем несколько шагов по каждому батчу (а, как известно из математической статистики, лучше измерить некоторую величину много раз и усреднить, чем доверять одному точному измерению). Попробуем теперь изменить размер батча до 100 объектов. Получим следующие графики:





Время обучения значительно увеличилось. Этому способствует несколькоф факторов. Во-первых, мы тратим время на перемешивание выборки на каждом эпохе и индексациях по ней. Во-вторых, обработка каждого батча соответствует матричным операциям, которые эффективно выполняются с помощью NumPy, а в случае стохастического градиентного мы поставляем с помощью медленного в питоне цикла for небольшие матрицы (эффективнее векторизованно работать с большими матрицами, а не с множеством маленьких). Далее, немного повысилась точность на обучающей выборке: если для SGD с размером батча в 5000 лучшая точность на обучении была около 0.935, SGD с размером батча в 100 имеет лучшую точность, равную 0.973. Несмотря на это, качество на тестовой выборке почти не изменилось: точность осталась равной около 0.886.

Поскольку с уменьшением размера батча мы будем получать все большее суммарное количество итераций, нужно помнить о коэффициенте перед градиентом, который будет стремиться приближаться к нулю. К примеру, если бы мы выбрали коэффициент step_beta = 1.0, то получили бы следующие результаты:



В этом случае, как мы и ожидали, коэффициент перед градиентом быстро стал слишком маленьким, тем самым не дав модели дойти до оптимума.

Если же мы будем увеличивать батч, то поведение модели будет стремиться к поведению классического градиентного спуска, ведь если размер батча станет равен размеру обучающей выборке, то мы и получим стандартный градиентный спуск. Время работы будет, исходя из описанных выше соображений, сокращаться.

Зафиксируем размер батча в 5000 объектов и исследуем зависимость качества классикации от параметров step_alpha и step_beta, как мы делали это для обычного градиентного спуска. Будем перебирать ту же сетку, что и в прошлый раз.

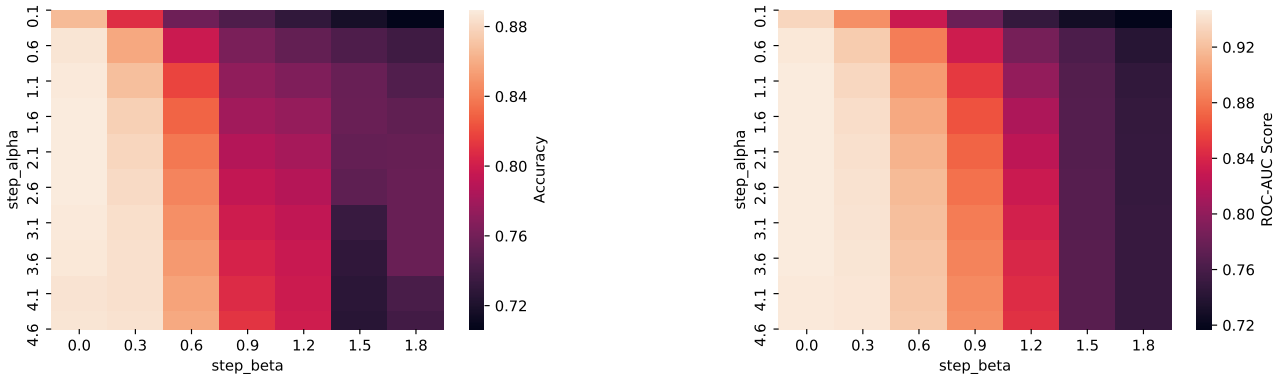


Рис. 3: Метрики качества классифкации в зависимости от параметров модели (SGD)

Эти heatmap'ы похожи на те, что мы видели ранее, однако все же есть несколько отличий. Во-первых, качество классифкации немного выше (тона чуть более светлые, чем для классического градиентного спуска), а также модели, использующие step_beta=0 также стали показывать хорошее качество классифкации. Это можно попробовать объяснить тем, что, поскольку мы меняем вектор весов не небольшие по модулю значения (на каждом батче), то нет необходимости постоянно уменьшать коэффициент. В классическом же градиентном спуске модуль градиента может оказаться достаточно большим, и, как мы видели ранее, модель будет перешагивать через оптимальную точку. Как и в прошлый раз, видим, что при значения step_alpha близких 5.0, качество становится выше, поэтому посмотрим, что происходит дальше.

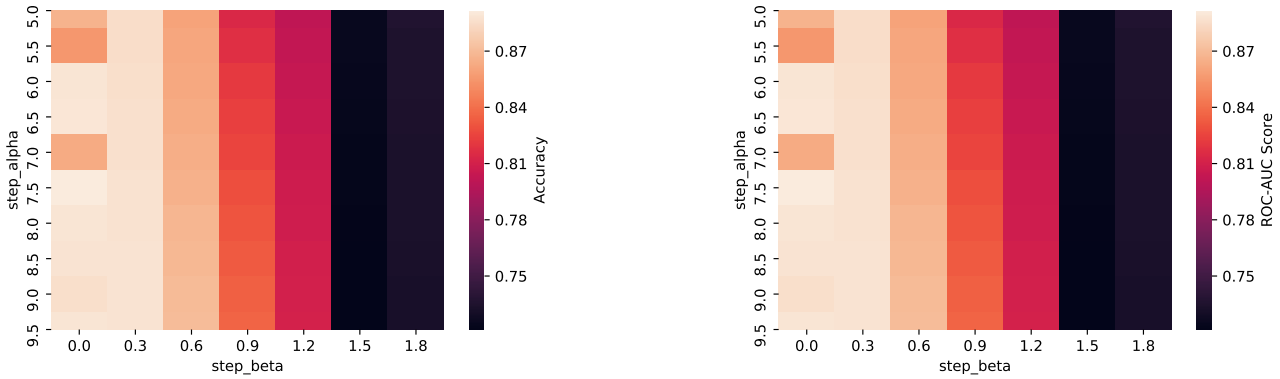


Рис. 4: Метрики качества классифкации в зависимости от параметров модели (SGD)

Как и для классического градиентного спуска, качество перестает сильно изменяться при увеличении step_alpha до диапазона (5.0, 10.0).

Выше мы отметили, что стохастический градиентный спуск стал показывать лучшие, в сравнении с классическим градиентным спуском, результаты при step_beta $= 0$. Посмотрим, как выглядят график функции ошибок для такой модели. Параметр step_alpha выберем равным 5.

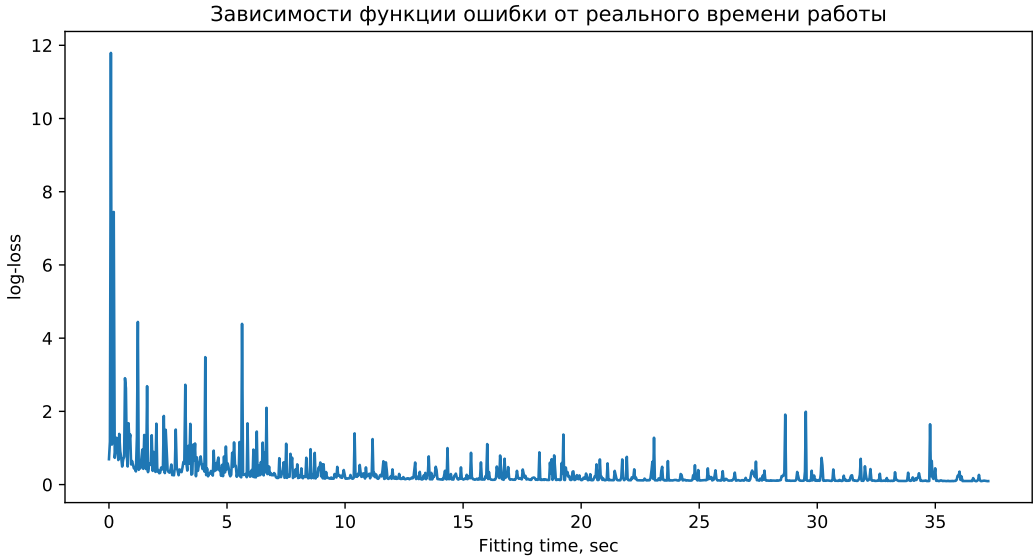


График стал стабильнее чем график модели стандартного градиентного спуска при таких же параметрах.

Сравнение GD и SGD

Фактически, в предшествующем пункте мы уже провели сравнение этих двух методов. Подведем некоторые итоге уже сказанному. Во-первых, стохастический гра-

диентный спуск является обобщением градиентного спуска, которое сходится к нему при увеличении размера батча до размера всей выборки. Тем самым, этот алгоритм является более гибким и настраиваемым. Во-вторых, при небольшом размере батча стохастический градиентный спуск может работать несколько стабильнее классического, особенно, если коэффициент перед градиентом не уменьшается с каждой итерацией. Также мы можем измерять ошибку не каждую эпоху, а каждый батч, и останавливать алгоритм, когда будет достигнута сходимость. В связи с этим стохастический градиентный спуск может работать быстрее. Также стоит отметить одно из главных преимуществ SGD: при его использовании нет необходимости хранить всю обучающую выборку в оперативной (или видео) памяти. Это особенно важно при работе с большими данными, ведь в современном машинном обучении выборки достигают терабайтов, и такие объемы просто невозможно (и бессмысленно) одновременно загружать в ОЗУ.

Лемматизация

Попробуем применить лемматизацию к комментариям. Это преобразование приводит каждое слово в тексте к его нормальной форме. Будем использовать библиотеку nltk и функцию WordNetLemmatizer. Как и в прошлый раз, к полученному корпусу применим CountVectorizer и добавим константный единичный признак. Стоит заметить, что размерность признакового пространства понизилась. При фиксированном параметре min_df=0.0001 метода CountVectorizer для лемматизированного корпуса мы имеем 12979 признаков, в том время как без лемматизации их было 16051. Очевидно, что признаковое пространство понизило размерность, поскольку теперь некоторые слова, которые отличались до лемматизации, но являлись разными формами одного и того же слова, стали одним и тем же признаком, соответствующему нормальной форме этого слова. Будем использовать логистическую регрессию на основе классического градиентный спуска с следующими параметрами: step_alpha = 5.0, step_beta = 0.1, регуляризация отключена, 5000 итераций, tolerance=1e-8. Получим точность 0.896, roc-auc=0.9477. Таким образом, лемматизация повысила точность классификации на целый процент. Поскольку размерность признакового пространства сократилось, время обучения также уменьшилось.

Bag of words VS TF-IDF

До этого мы кодировали каждый комментарий вектором, в котором в i-й позиции стоит количество раз, которое i-е слово из словаря встретилось в этом комментарии. Существует также чуть более продвинутое кодирование текстов - TF-IDF. Эта модель учитывает популярность слова во всем корпусе. К примеру, слова "когда "это "из "или"и т.д., очевидно, не являются важными для классификации текста, поскольку несут лишь служебный характер, а не отражают смысл. Используем, как и раньше, параметр min_df = 0.0001. Посмотрим, какой способ кодирования слов даст лучший результат. Будем использовать подобранную выше модель: обычный

градиентный спуск, step_alpha=5.0, step_beta=0.1, регуляризация отключена, 5000 итераций.

Сначала не будем использовать лемматизацию. Получаем следующие результаты:

- Bag of words. Accuracy: 0.887
- TF-IDF Accuracy: 0.876

Теперь посмотрим на результаты с использованием лемматизации:

- Bag of words. Accuracy: 0.896
- TF-IDF Accuracy: 0.880

Таким образом, в нашем случае TF-IDF работает немного хуже.

Исследование зависимости качества от размерности признакового пространства

В предыдущих экспериментах мы использовали CountVectorizer с фиксированным параметром min_df=0.0001. Как уже было указано ранее, это значит, что в словарь не попадут слова, которые встретились в менее чем 0.01% комментариев. Существует также параметр max_df. Если он равен $x < 1.0$, то в словарь не попадут слова, который встречаются в более чем x * 100% документах. Таким образом, мы можем варьировать размер словаря, исключая слишком редкие и слишком частые слова. Проведем несколько экспериментов с этими параметрами. Используемая модель: классический градиентный спуск, step_alpha=5.0, step_beta=0.1, 5000 итераций, регуляризация отключена, лемматизация включена, tolerance=1e-8. Результаты можно видеть в следующей таблице:

min_df	**max_df**	**dim**	**Accuracy**	**ROC-AUC**
-	-	78338	0.884	0.946
0.0001	-	12979	0.896	0.948
-	0.0001	65359	0.699	0.607
0.0001	0.6	12977	0.890	0.948
0.001	-	3203	0.891	0.945
0.01	-	546	0.815	0.897

Таблица 1: Результаты экспериментов. dim - размерность признакового пространства

Из этих данных можно сделать несколько выводов. Во-первых, нет смысла брать в словарь все слова, потому что тогда признаков получается слишком много, модель можно переобучаться на шумовые признаки, находить случайные зависимости. Во-вторых, нельзя обучаться исключительно на редких словах (эксперимент 3). Модели сложно определить, является ли слово характерным для токсичных комментариев

или нет, если оно встретилось лишь 1 или 2 раза среди 50 тысяч комментариев, поэтому качество также падает. В-третьих, можно достаточно сильно понизить размерность признакового пространства, не так сильно потеряв в качестве (см. эксперименты 2 и 5). Наконец, нельзя слишком сильно понижать признаковое пространство, оставляя лишь самые популярные слова (см. последний эксперимент). Возможно, это бы работало чуть лучше, если бы классы были сбалансированы, однако в нашем случае модель не получает достаточно много важной информации.

Анализ ошибок

Рассмотрим подробне, какие ошибки допускает наилучшая из рассмотренных нами моделей: логистическая регрессия на основе градиентного спуска, step_alpha = 5.0, step_beta = 0.1, tolerance=1e-8, 5000 итераций, регуляризация отключена, лемматизация включена. Её матрица ошибок выглядит следующим образом:

	$\alpha(x) = +1$	$\alpha(x) = -1$
y=+1	5193	1050
y=-1	1102	13331

Таблица 2: Матрица ошибок. Класс +1 означает, что комментарий токсичный

В этом случае точность (precision) равна 0.831, а полнота 0.824. Рассмотрим примеры объектов, на которых модель ошиблась. К примеру, модель отнесла комментарий "dear god this site is horrible"к токсичным, хотя они таким не является. Вероятно, этот вывод был сделан из-за слова "horrible". Другим примером может служить комментарий "arabs are committing genocide in iraq but no protests in europe may europe also burn in hell". Несмотря на то что модель не отнесла этот комментарий к токсичным, она выдала для него вероятность токсичности 0.47, так что в реальной задаче стоило бы попробовать разные пороги для классификации (в нашей модели везде использовался порог 0.5). Наконец, наша модель классифицировала комментарий о сленге как токсичный. Вот его фрагмент: "i belive that is majorly true very much so okay so the good meaning is a female dog bitch it also stands for the name brittany fellows preceding unsigned comment added by etymology the word bitch is from the old norse bikkjuna meaning female of the dog of unknown orig". Совершенно очевидно, почему произошла эта ошибка.

Выводы

В данном отчете были представлены результаты экспериментов с логистической регрессией. Были рассмотрены теоретические основы этой модели (включая неочевидные их аспекты), были сравнены между собой методы градиентного спуска и стохастического градиентного спуска. На примере задачи классификации комментариев мы также рассмотрели некоторые методы работы с текстовыми данными, включая

модели Bag of words, TF-IDF, а также метод лемматизации. Наконец, мы посмотрели на примеры объектов, на которых классификатор ошибается.

Список литературы

[1] Попов Артём. Краткий конспект лекции «Линейные модели классификации». https://github.com/mmp-practicum-team/mmp_practicum_fall_2019/blob/master/09_linear_and_text/linear_model.pdf, 2018.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007.