Московский Государственный Университет

Градиентные методы обучения линейных моделей

Выполнил: Курцев Д.В.
Группа: 317

Факультет Вычислительной математики и кибернетики
Кафедра Математических методов прогнозирования

Ноябрь 2021

Введение

В данном задании была реализована и разобрана логистическая регрессия с методом градиентного спуска (GD) и стохастического градиентного спуска (SGD). Её суть заключается в следующем.

Для начала разберём задачу бинарной классификации. Т.е. метки класса могут принимать всего 2 значения. Для определённости будем считать, что $y_i \in \{-1, +1\}$. Пусть у нас есть матрица объектов-признаков $X \in \mathbb{R}^{n \times d}$ с их значениями классов $\{y_1, ..., y_n\}$. Сразу же будем считать, что среди признаков есть константа. Тогда линейная модель классификации определяется следующим образом:

$a(x) = sign(\langle x, w \rangle)$, где w - вектор весов, который "подбирает" модель, $w \in \mathbb{R}^d$.

В логистической регрессии используется следующая функция ошибки на объекте $\mathcal{L}(x_i, w) = log(1 + e^{-y_i \langle x_i, w \rangle})$. Обозначим за $M_i = y_i \langle x_i, w \rangle$. Величина M_i называется *отступом (margin)*. Тогда получим следующий функционал потерь на всей выборке X, который мы будем минимизировать:

$$Q(X, w) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(x_i, w) = \frac{1}{n} \sum_{i=1}^{n} log(1 + e^{-M_i}) \to \min_w$$

Чтобы уменьшить переобучение, обычно к такому функционалу добавляют слагаемое, зависящее только от весов модели. В данной работе мы воспользовались L_2 регуляризацией и рассмотрели следующий функционал:

$$Q(X, w) = \frac{1}{n} \sum_{i=1}^{n} log(1 + e^{-M_i}) + \frac{\lambda}{2} \|w\|^2 \to \min_w$$

Известно что, градиент указывает направление наибольшего возрастания функции. Поэтому, чтобы сойтись к минимуму, необходимо идти в сторону антиградиента. Для этих целей найдём градиент нашего функционала:

$$dQ = \frac{1}{n} \sum_{i=1}^{n} dlog(1 + e^{-M_i}) + \frac{\lambda}{2} d\|w\|^2 = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{1 + e^{-M_i}} \, de^{-y_i \langle x_i, w \rangle} +$$

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

$$\frac{1}{n}\sum_{i=1}^{n}\frac{-y_i}{1+e^{M_i}}\,\langle x_i,dw\rangle) + \lambda\langle w,dw\rangle = \langle -\frac{1}{n}\sum_{i=1}^{n}\frac{y_i}{1+e^{M_i}}x_i + \lambda w, dw\rangle$$

Таким образом, получим, что градиент нашего функционала равен:

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

Теперь рассморим задачу мультиномиальной регрессии. В этом случае строится k линейных моделей $a_1(x),...,a_k(x), a_j = sign(\langle w_j,x\rangle)$. Где каждая модель даёт оценку принадлежности объекта к определённому классу. Вероятность j-го класса можно выразить как:

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

Тогда по методу максимального правдоподобия функционал ошибки принимает следующий вид:

$Q(X,w) = -\frac{1}{n}\sum\limits_{i=1}^{n} log(\mathbb{P}(y_i|x_i)) + \frac{\lambda}{2}\sum\limits_{i=1}^{k}\|w_i\|^2 \to \min\limits_{w}$

Найдём $\frac{dQ}{dw_m}$

$$dQ = -\frac{1}{n}\sum_{i=1}^{n} d\,log\frac{e^{\langle w_{y_i},x_i\rangle}}{\sum\limits_{j=1}^{k}e^{\langle w_j,x_i\rangle}} + \lambda\langle w_m,dw_m\rangle = -\frac{1}{n}\sum_{i=1}^{n}(d\,log\,e^{\langle w_{y_i},x_i\rangle} -$$

$$-d\;log\sum_{j=1}^{k}e^{\langle w_j,x_i\rangle}) + \lambda\langle w_m,dw_m\rangle \;=\; -\tfrac{1}{n}\sum_{i=1}^{n}(d\;\langle w_{y_i},x_i\rangle \;-\; \frac{de^{\langle w_m,x_i\rangle}}{\sum\limits_{j=1}^{k}e^{\langle w_j,x_i\rangle}} +$$

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

$$\Rightarrow \quad \nabla_{w_m} Q = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{e^{\langle w_m, x_i \rangle}}{\sum\limits_{j=1}^{k} e^{\langle w_j, x_i \rangle}} - \mathbb{1}\{y_i = m\} \right) x_i + \lambda w_m$$

Покажем, что задача многоклассовой логистической регрессии сводится к бинарной. Пусть у нас k = 2. Тогда:

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

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

где мы обозначили $w = w_1 - w_2$. Теперь переобозначим классы числами +1 и -1 и посмотрим на функционал без регуляризации:

$$Q(X, w) = -\frac{1}{n} \sum_{i=1}^{n} log(\mathbb{P}(y_i|x_i)) = -\frac{1}{n} \sum_{i=1}^{n} [log(\frac{e^{\langle w_1, x_i \rangle}}{e^{\langle w_1, x_i \rangle} + e^{\langle w_2, x_i \rangle}}) \mathbb{1}\{y_i = 1\}$$

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

$$+ log(\frac{1}{1 + e^{\langle w_1 - w_2, x_i \rangle}}) \mathbb{1}\{y_i = -1\}] = \frac{1}{n} \sum_{i=1}^{n} log(1 + e^{-y_i \langle w, x_i \rangle})$$

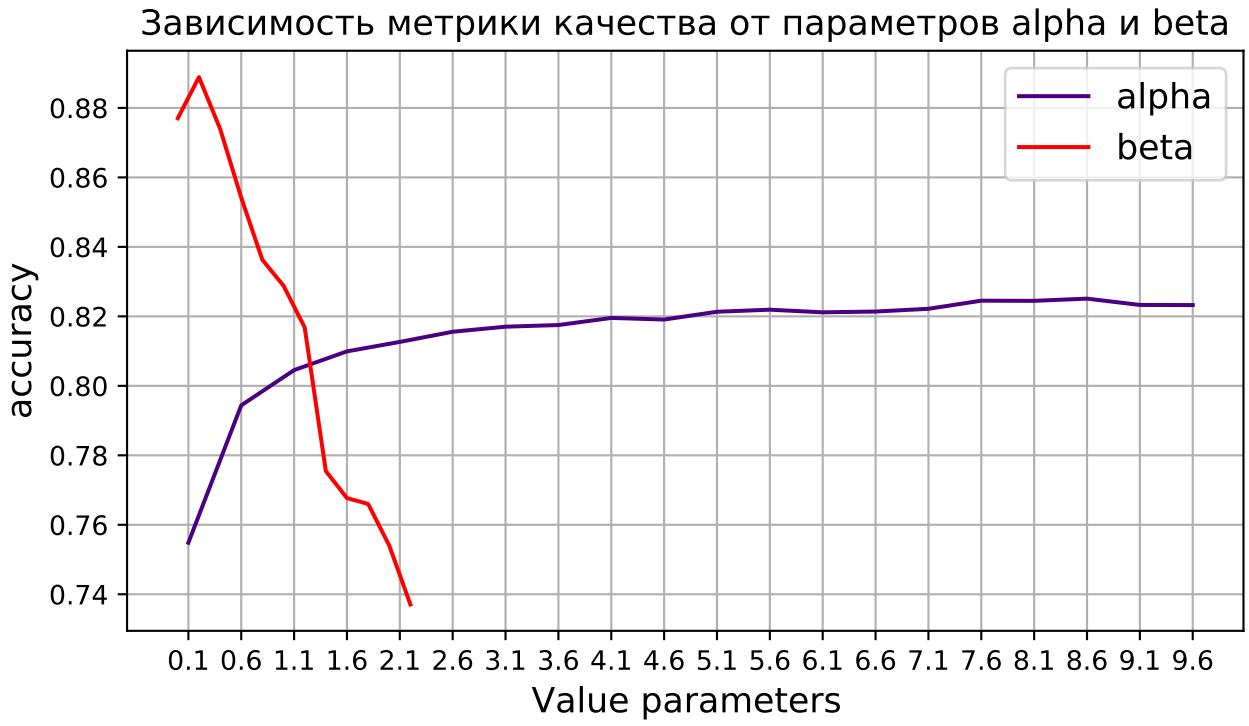
Таким образом, мы свели задачу мультиномиальной классификации к бинарной.

Как мы уже ранее отметили, вектор весов w на каждой итерации изменяется в сторону антиградиента функционала: $w^{(k+1)} = w^{(k)} - \eta_k \nabla Q$, где $\eta_k = \frac{\alpha}{k^\beta} > 0$, а α и β - гиперпараметры алгоритма.

Проанализируем как работают линейные модели в зависимости от разных гиперпараметров. Рассмотрим задачу классификации комментариев на токсичные и нет. Все эксперименты проводились на датасете, включающим в себя 52 тыс. объектов обучающей и 20.5 тыс. объектов тестовой выборки.

Градиентный спуск (GD)

Для начала произведём предварительную обработку текста. Приведём все тексты к нижнему регистру. Заменим в тексте все символы, не являющиеся буквами и цифрами, на пробелы. Далее разобьём случайным образом обучающую выборку на валидационную и обучающую в отношении 3:7. Запустим модель для нескольких различных параметров с нулевым начальным приближением и проанализируем, какие значения показывают наилучшее качество. Посмотрим на среднее значение метрики accuracy в зависимости от разных α и β.



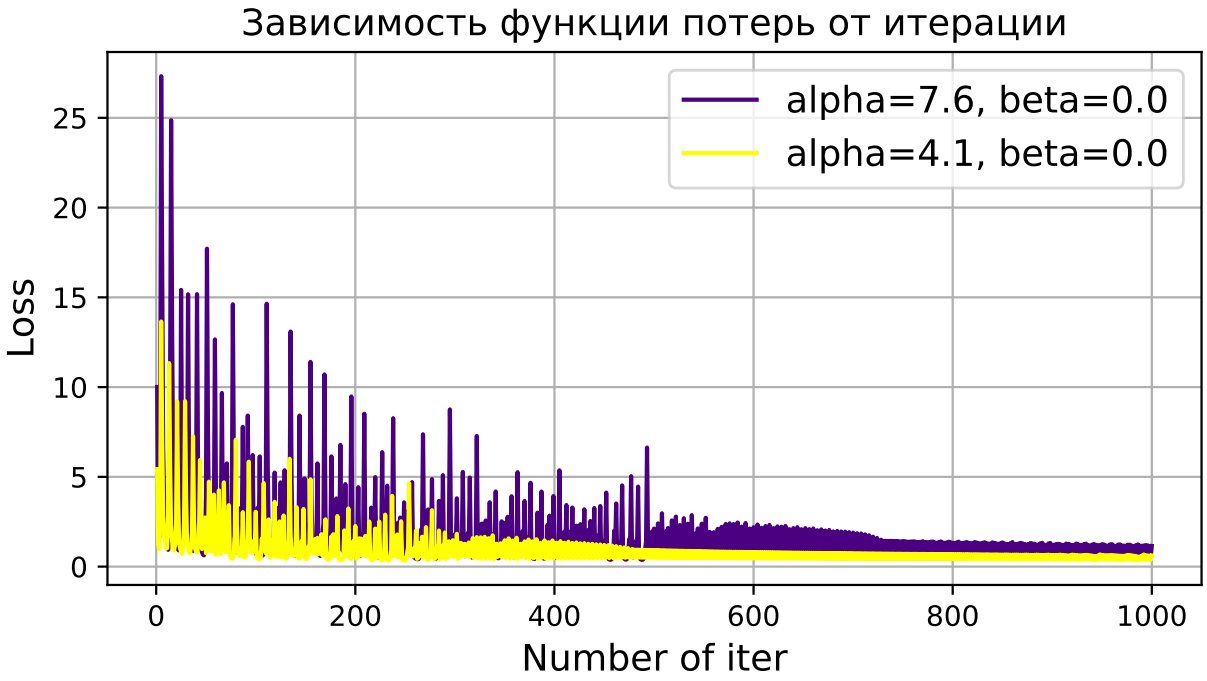
GD, $w = 0$

Видим, что алгоритм работает намного лучше при не очень

больших значениех параметра β. Это можно объяснить тем, что при довольно больших значениях β параметр η_k быстро падает и начинает стремиться к нулю. Т.е. можно сказать, что мы практически перестаём обновлять веса, поэтому качество модели уменьшается. Примерно тот же вывод можно сделать о α, т.к. при его низких значениях η_k довольно мало, и мы просто не успеваем достичь экстремума функционала.

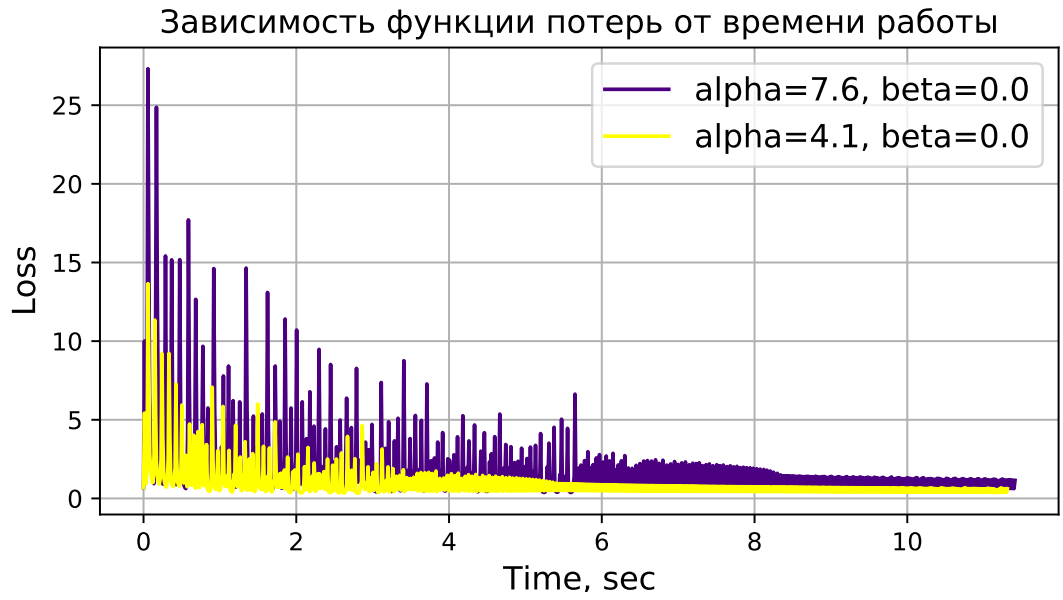
Наша модель показала наилучшее качество(0.9036) при параметрах $\alpha = 7.6, \beta = 0.2$.

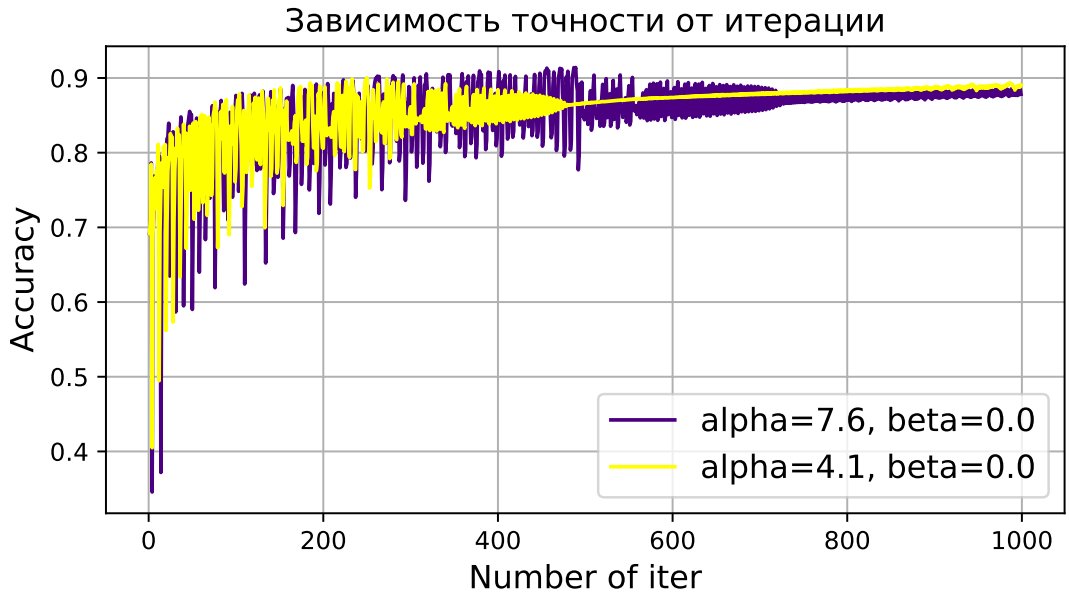
Исследуем поведение градиентного спуска в зависимости от разных значений параметра α. Для этого построим различные графики зависимости функции потерь и accuracy от итерации и времени работы модели.

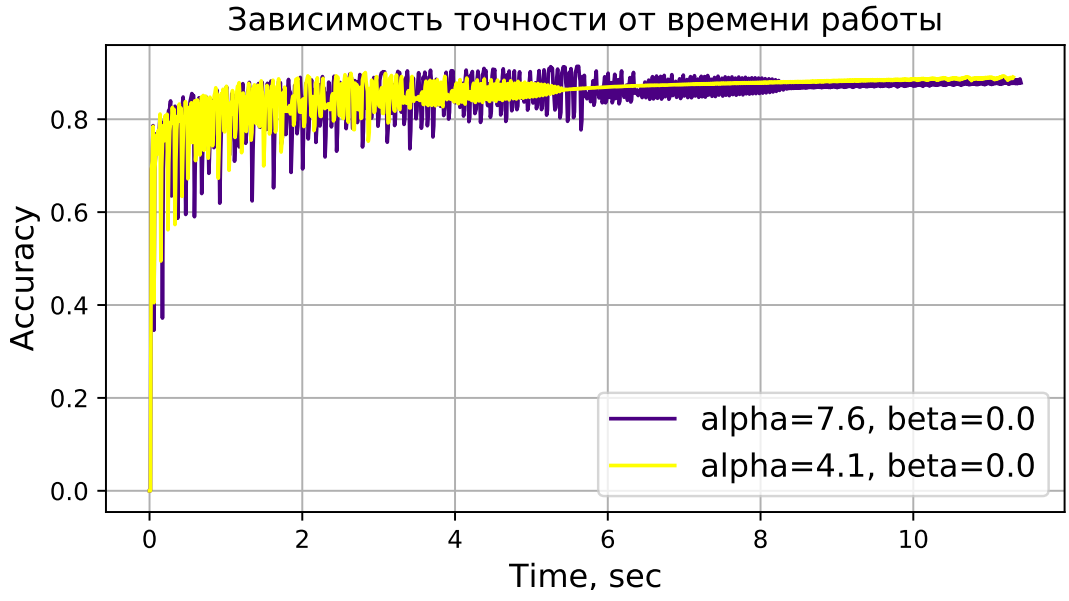


GD, $w = 0$

Заметим, что при меньшем значении α, графики функционала и точности быстрее выходят на плато. Это можно объяснить тем, что значения η_k становятся меньше, поэтому мы перестаём сильно перескакивать через оптимальное значение весов. Что влечёт за собой более ровные графики. Отметим, что графики зависимостей от времени и итерации абсолютно идентичны. Поэтому всюду да-

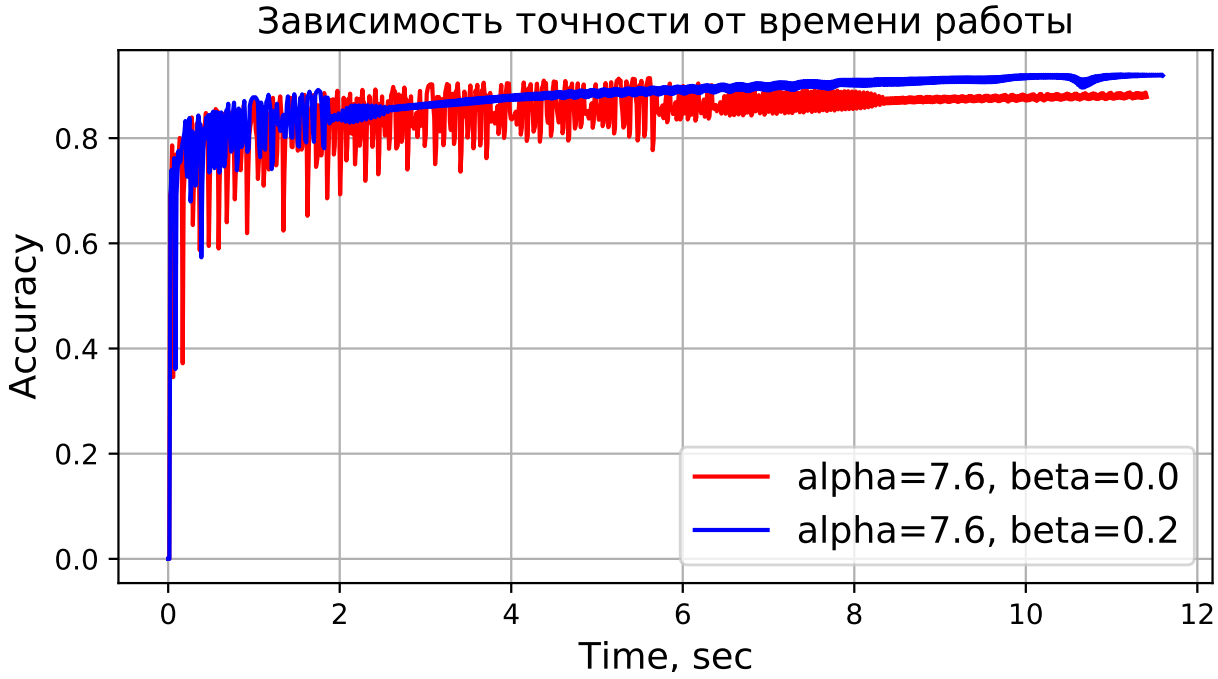

Зависимость функции потерь от времени работы
alpha=7.6, beta=0.0
alpha=4.1, beta=0.0
Loss
Time, sec


Зависимость точности от итерации
alpha=7.6, beta=0.0
alpha=4.1, beta=0.0
Accuracy
Number of iter


Зависимость точности от времени работы
alpha=7.6, beta=0.0
alpha=4.1, beta=0.0
Accuracy
Time, sec

лее мы будем приводить лишь по 2 графика (точность от времени и функции ошибки от итерации или наоборот). С полным набором графиков можно ознакомиться в Приложении 1.

Посмотрим теперь, как влияет значение β на поведение графиков.

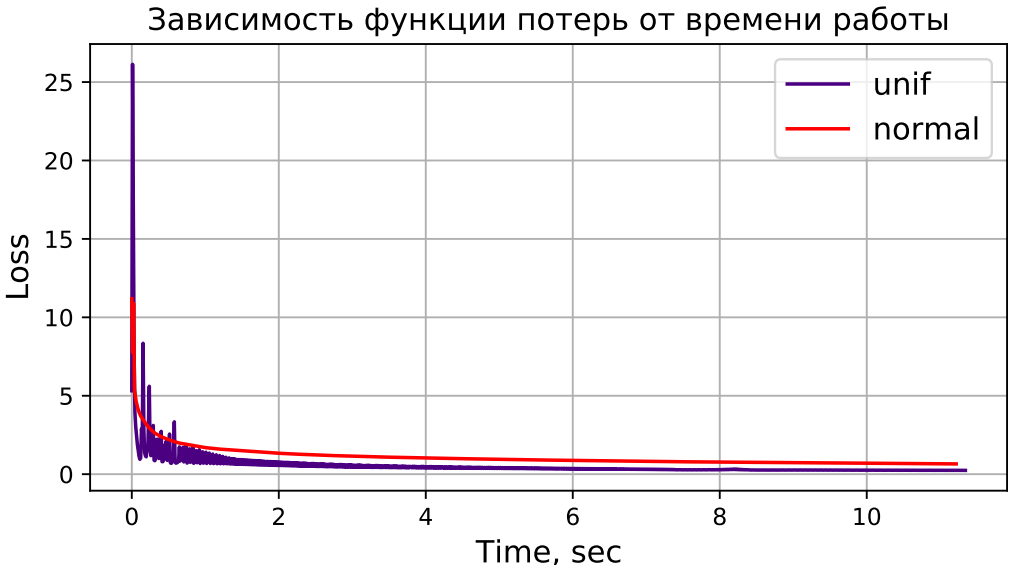


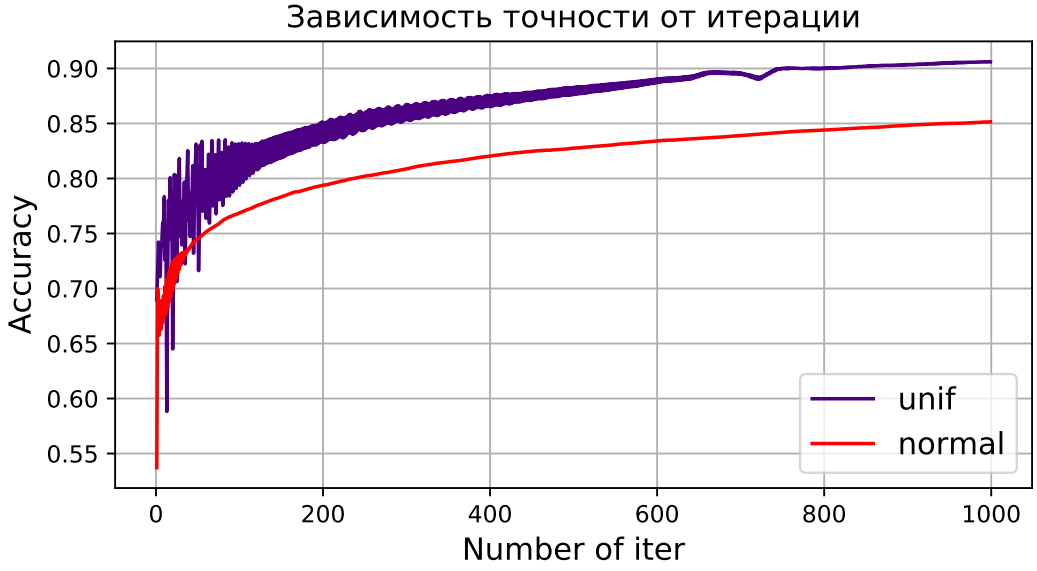
GD, $w = 0$

Нетрудно увидеть, что $\beta = 0$, плохо подходит для задачи. Так

как при таком значении параметра η_k вообще не изменяются. И получается так, что значения весов постоянно перескакивают через оптимум. Поэтому мы наблюдаем сильно изрезанные красные графики, которые так и не сходятся к оптимальному значению. При $\beta = 0.2$ η_k с каждой итерацией уменьшаются, поэтому функция потерь и точность быстро достигают своих значений и перестают изменяться. Что мы и видем на рисунках: синие графики гораздо быстрее выходят на плато.

Всюду до этого мы начинали обучение с инициализацией весов нулями. Посмотрим, что произойдёт, если мы в качестве начального приближения возьмём веса из равномерного распределения $U[-1, 1]$ или из нормального $\mathcal{N}(0, 1)$. Возьмём $\alpha = 7.6, \beta = 0.2$.



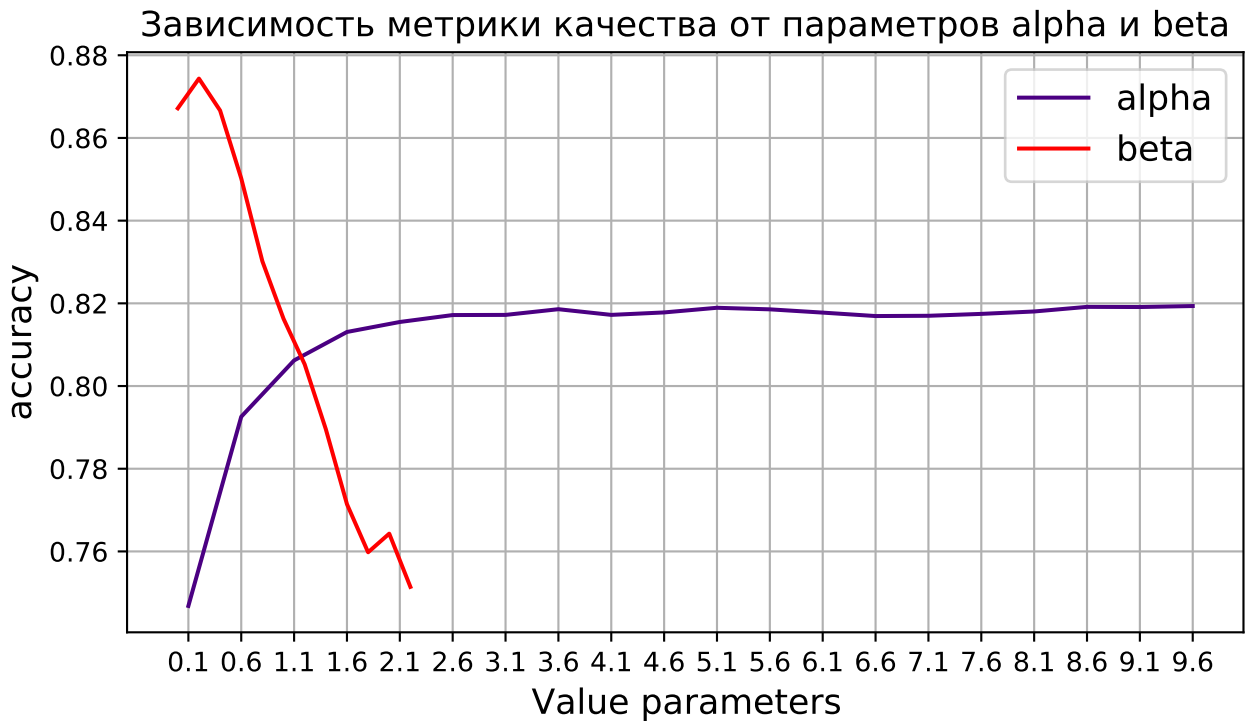


Видим, что если взять w_0 из нормального распределения, то графики гораздо быстрее выходят на плато и перестают меняться. А если взять из равномерного, то графики очень сильно напоминают случай константного приближения. Это можно объяснить тем, что веса в нашей модели скорее всего не будут абсолютно одинаковыми или равномерно распределёнными, т.к. некоторые признаки играют более важную роль, некоторые меньшую.

Отметим, что если $w_0 \sim U[-1, 1]$, то точность на валидационной выборке равна 0.8879, а если $w_0 \sim \mathcal{N}(0, 1)$, то 0.8415. Что хуже, чем при $w_0 = 0$. Поэтому далее будем запускать GD при $\alpha = 7.6, \beta = 0.2, w_0 = 0$.

Стохастический гадиентный спуск (SGD)

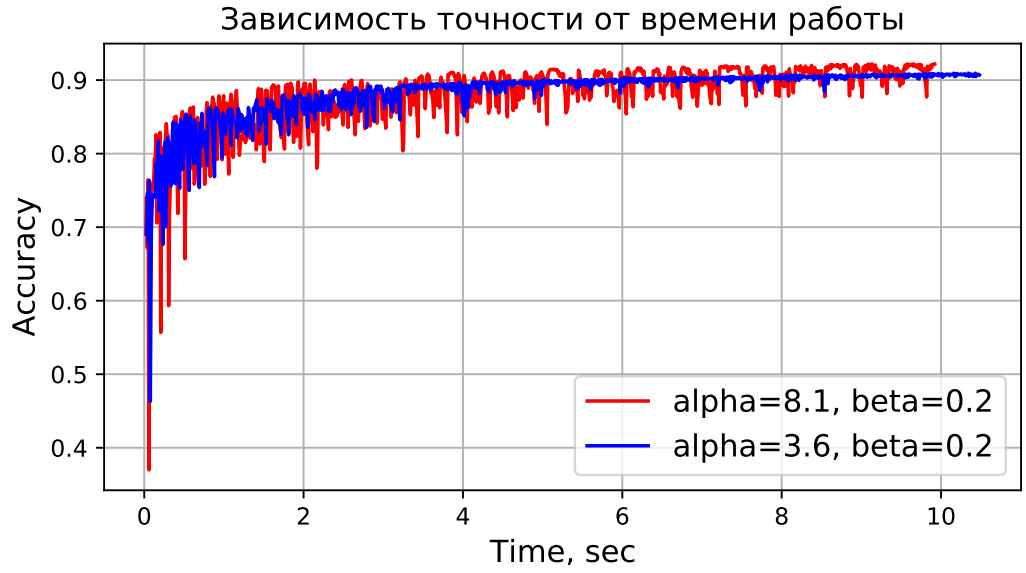
Теперь проанализируем работу стохастического градиентного спуска. Аналогично предыдущему пункту запустим модель при $w_0 = 0$ для разных α и β и выберем наилучшие параметры. $batch\_size$ возьмём равным 100.



SGD, $w = 0, batch\_size = 100$

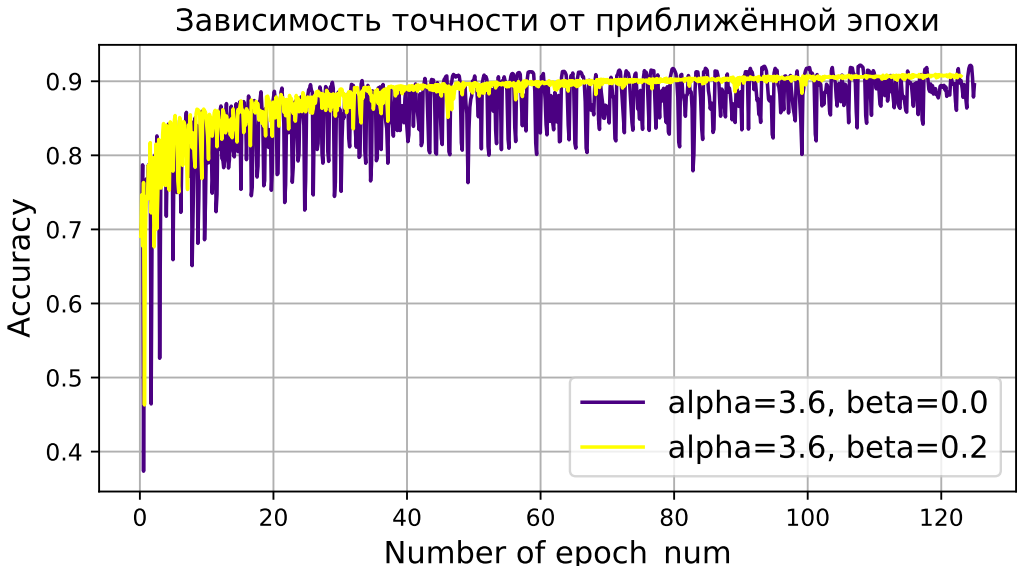
Видим, что график получился примерно таким же. Но в данном случае точность чуть снижается. Наилучшими параметрами являются $\alpha = 3.6, \beta = 0.2$ с accuracy $= 0.8826$. Качество немного упало. Это происходит из-за того, что на каждой итерации мы считаем градиенты лишь на небольшой части выборки. Зато это даёт значительный выигрыш во времени работы (всего около 4 секунд, против 11.5)

Аналогично градиентному спуску исследуем зависимости при различных значениях параметров α и β. Чтобы графики получились более информативными будем чаще обновлять историю, пожертвовав временем работы алгоритма. С полным набором графиком можно ознакомиться в Приложении 2. $batch\_size = 5000$.



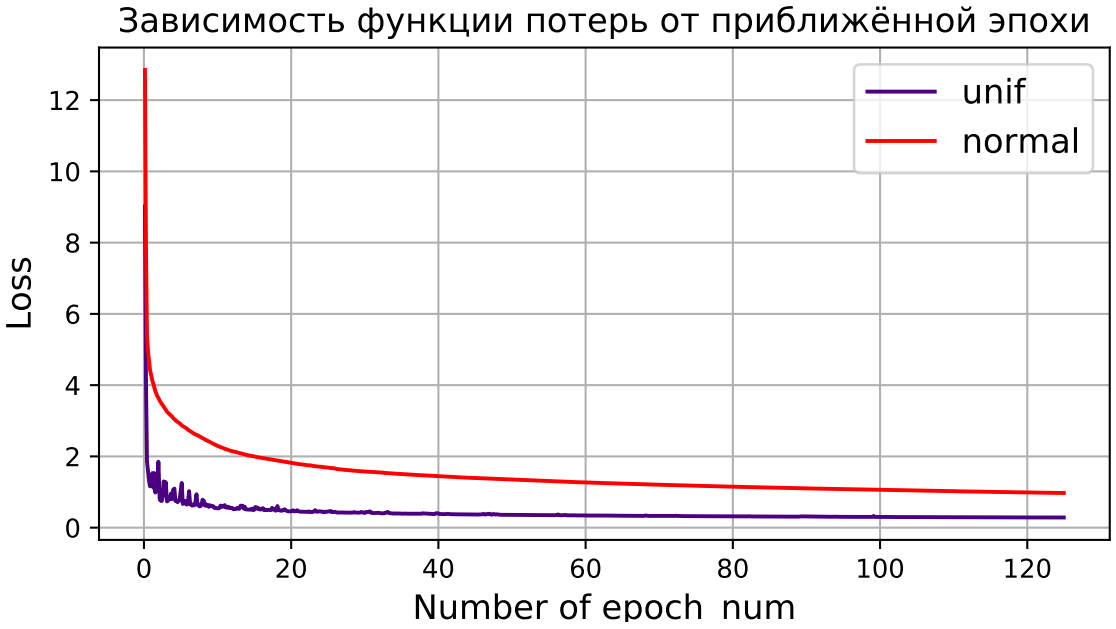
Заметим, что графики при разных α сильно напоминают графики модели GD. Отметим, что график функции ошибки гораздо быстрее выходит на плато, чем соответствующий ему график GD. Он становится более стабильным. Это можно объяснить тем, что в SGD параметр η_k понижается намного быстрее, чем в градиентном спуске. Так как обновление весов происходит после каждого батча и за итерацию SGD считается именно проход по одному батчу, а не по всей выборке. Поэтому мы довольно быстро перестаём переходить через оптимум и начинаем к нему сходиться. Однако отметим, что при большом значении α график accuracy довольно сильно изрезан.

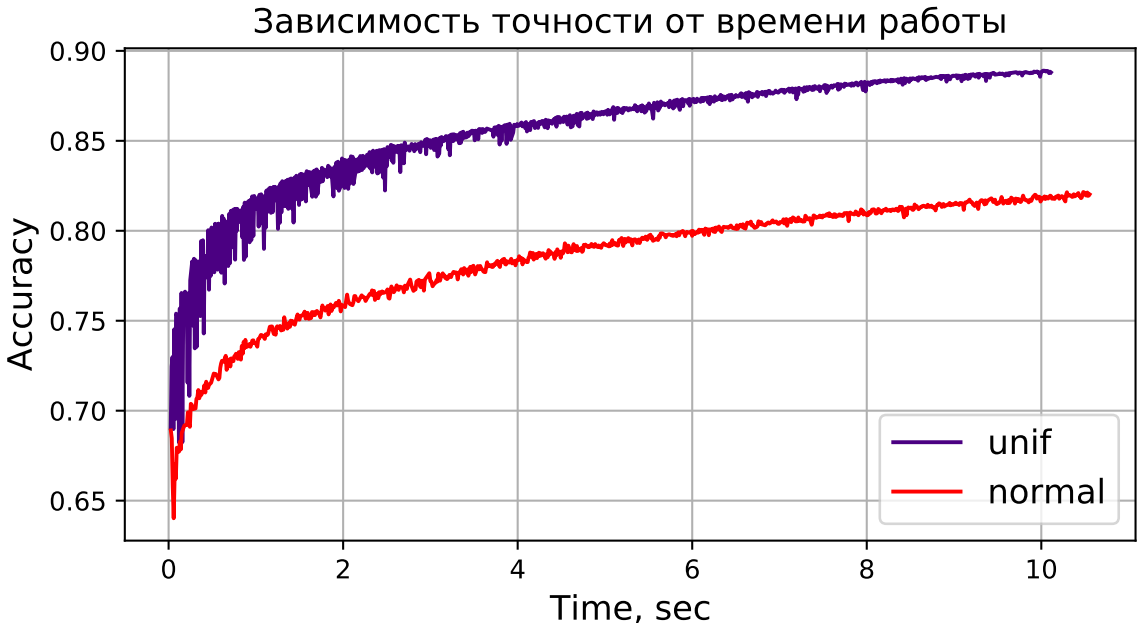
Посмотрим на поведение графиков в зависимости от β.



Аналогичные выводы (GD) можно сделать и для этих графиков. Отметим, что значение $\beta = 0.2$ по-прежнему является наилучшим, а $\beta = 0$ вообще непригодным, так как алгоритм так и не сошёлся к оптимуму, хоть и стал очень близок к нему

Так же, как и для GD, исследуем результаты модели, если мы в качестве начального приближения возьмём веса из равномерного распределения $U[-1, 1]$ или из нормального $\mathcal{N}(0, 1)$. Зафиксируем $\alpha = 3.6, \beta = 0.2, batch\_size = 100$.



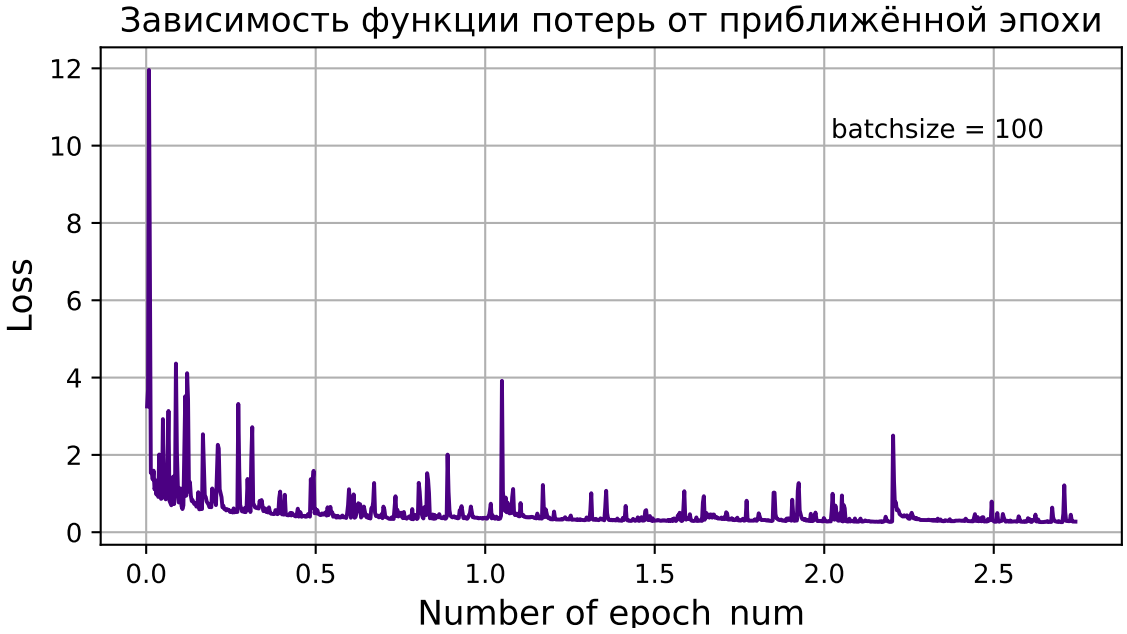


SGD, $\alpha = 3.6, \beta = 0.2, batch\_size = 5000$

Заметим, что поведение алгоритма аналогично GD. Отметим,

что если $w_0 \sim U[-1, 1]$, то точность на валидационной выборке равна 0.8739, а если $w_0 \sim \mathcal{N}(0, 1)$, то 0.8135. Что хуже, чем при $w_0 = 0$. Поэтому далее будем запускать SGD при $\alpha = 3.6, \beta = 0.2, w_0 = 0$.

Теперь исследуем работу SGD в зависимости от размера батча. Так как приближённые номера эпох в зависимотсти от $batch\_size$ имеют разную размерность, для наглядности графики от номера эпохи приведём на разных картинках. Чтобы чуть ускорить алгоритм, будем реже записывать историю.



SGD, $\alpha = 3.6, \beta = 0.2, w_0 = 0$

Заметим, что при увелечинии размера батча графики функции потерь и точности ведут себя более сглаженно. Так как на каждой итерации мы смотрим на большее число объектов, что позволяет нам делать более объективные оценки вектора весов. Поэтому мы более стабильно сходимся к оптимуму. Однако, время работы алгортима увеличивается.

GD vs SGD

Сравним между собой стандартный и стохастический градиентный спуск. Сравнения будем проводить на наилучших параметрах, выбранных в предыдущих пунктах.

	%	**Time, sec**
	90.36	11.6

GD

batch_size	%	**Time, sec**
100	87.70	4.59
1 000	90.23	5.48
10 000	90.41	11.3

SGD

Таблица 1: $\alpha = 7.6, \beta = 0.2, w_0 = 0, max\_iter = 1000$

	%	**Time, sec**
	89.57	11.3

GD

batch_size	%	**Time, sec**
100	88.26	4.91
1 000	89.44	5.18
10 000	89.54	11.9

SGD

Таблица 2: $\alpha = 3.6, \beta = 0.2, w_0 = 0, max\_iter = 1000$

Видим, что при увелечении $batch\_size$ точность работы SGD увеличивается и догоняет GD. Вместе с этим возрастает и время работы алгоритма. И становится сопоставимым с временем, затраченным на GD. Так же роста точности можно добиться и увеличением количества итераций. Так, SGD при $\alpha = 7.6, \beta = 0.2, batch\_size = 100, max\_iter = 5000$ показывает точность равную 90.68%, а при $batch\_size = 10000$ - 90.91%. Таким образом, можно заключить, что SGD является более гибким и настраиваемым. Он гораздо быстрее сходится, при этом уступая в точности совсем немного, а иногда даже чуть-чуть выигрывает. Заметим так же, что он является более пригодным и по памяти. Так как на каждой итерации необходимо держать лишь $batch\_size$ элементов, а не всю выборку.

Лемматизация

Применим к нашей выборке алгоритм лемматизации. То есть приведём все слова к начальной форме. И так же поделим нашу выборку на обущающую и валидационную. Отметим, что до лемматизации признаковое пространство было равно 3737, а после 3204, что существенно меньше. Так происходит, потому что раньше в комментариях одни и те же слова имели различную форму (другое число, род) и поэтому считались, как разные.

Проверим, как меняется точность и время работы модели.

	$\alpha = 7.6$	$\alpha = 3.6$
%	89.87	89.77
t, sec	9.91	8.69

GD

	$\alpha = 7.6$	$\alpha = 3.6$
%	88.41	88.73
t, sec	4.4	4.42

SGD, $batch\_size = 100$

Таблица 3: $\beta = 0.2, w_0 = 0, max\_iter = 1000$

Заметим, что в большинстве случаев лемматизация улучшает качество работы алгоритма. Так как раньше некоторые слова имели разную форму представления и считались различными, поэтому их частота встречаемости в тексте была меньше. Что влекло за собой меньшие веса у значимых слов. Теперь, же частота важных слов возрастает, что даёт улучшение в качестве работы метода. Однако, как можно увидеть, бывают случаи, когда точность может чуть упасть. Возможно это происходит из-за того, что некоторые слова имеют несколько смыслов. Поэтому алгоритм лемматизации мог посчитать "плохое" слово за "хорошее" (и наоборот) и привести не к той форме.

Обратим внимание на то, что приведение слов к начальной форме позволяет сократить признаковое пространство. Благодаря этому время работы модели уменьшается.

Bag Of Words vs Tfidf

Всюду ранее мы пользовались $CountVectorizer$, который сопоставляет каждому токену частоту встречаемости этого токена в тексте документа. Существует ещё один способ векторизации - $TfidfVectorizer$. Он сопоставляет каждому токену частоту встречаемости этого токена в тексте документа умноженную на величину обратно пропорциональную частоте встречаемости токена во всех документах обучающей выборки. Он придаёт большую значимость словам, которые редко встречаются в остальных текстах, чем в одном из них.

Посмотрим какой из них лучше всего подходит для векторизации в нашей задаче. Исследования проведём на наилучших пара-

метрах, с лемматизацией и без неё.

	Без лем-ии	**С лем-ей**
%	90.36	89.87
t, sec	11.4	8.72

Count

	Без лем-ии	**С лем-ей**
%	87.04	87.57
t, sec	11.2	8.95

Tfidf

Таблица 4: GD $\alpha = 7.6, \beta = 0.2, w_0 = 0, max\_iter = 1000$

	Без лем-ии	**С лем-ей**
%	88.26	88.73
t, sec	4.53	4.44

Count

	Без лем-ии	**С лем-ей**
%	84.71	85.37
t, sec	4.51	4.59

Tfidf

Таблица 5: SGD $\alpha = 3.6, \beta = 0.2, w_0 = 0, batch\_size = 100$

Заметим, что $Tfidf$ проигрывает в качестве работы метода. Возможно, это связано с тем, что в комментариях люди используют самую обычную лексику. Поэтому все слова примерно одинаково распределены между классами.

Отметим, что время работы алгоритма практически никак не меняется. А размерность признакового пространства остаётся таким же. 3737 без лемматизации и 3204 с ней.

min_df, max_df

Всюду ранее мы использовали значение $min\_df = 0.001$. Этот параметр отвечает за то, что слова, имеющие частотность меньше указанного значения, не попадут в словарь. max_df отбрасывает слова, чья частотность попадания в тексты превышает данного значения. Таким образом, мы можем изменять признаковое пространство, убирая редкие или слишком частые слова.

Посмотрим, как данные параметры влияют на точность и время работы алгоритма.

min_df	**max_df**	**Accuracy, %**	**Time, sec**	**dim**
default	default	90.97	18.97	78339
0.0001	default	90.99	13.70	12980
0.001	default	89.87	11.7	3204
0.01	default	85.57	7.28	547
0.1	default	78.35	5.48	58
default	0.001	71.45	16.59	75136
default	0.01	84.92	17.8	77906
default	0.1	90.24	18.7	78291
default	0.5	90.30	18.53	78334

Таблица 6: GD, $\alpha = 7.6, \beta = 0.2, w_0 = 0, max\_iter = 1000$

Увидим, что с увеличением min_df наша точность сначала чуть увеличивается, а потом сильно падает. Это связано с тем, что мы убираем из выборки слова, характерные только для своего класса. То есть те, которые используются только в токсичных комментариях или наоборот. Поэтому мы не можем объективно оценивать принадлежность к тому или иному классу объектов из тестовой выборки. Зато заметим, что при росте min_df размерность признакового пространства сильно уменьшается, как и время работы метода.

Видно, что, напротив, увеличение параметра max_df позволяет повысить точность модели. Есть слова, которые могут встречаться в обоих классах, но в зависимости от контекста, они могут нести различную смысловую нагрузку, поэтому убирать их из выборки было бы неправильно. Отметим, что при росте max_df, размеры выборки увеличиваются, как и время, затраченное на работу алгоритма.

Анализ наилучшей модели

Ранее мы выбирали методы и гиперпараметры алгоритмов, улучшая качество работы на валидационной выборке. Теперь обучим модель на обучающей и проверим результат работы на тестовой.

Для начала посмотрим на работу обычного градиентного спуска с параметрами $\alpha = 7.6, \beta = 0.2, w_0 = 0$. Используем лемматизацию и векторицацию с помощью метода $CountVectorizer$ с параметрами $min\_df = 0.0001, max\_df = default$. Обучим модель на 5000 итерациях, чтобы получить более хорошую точность.

Качество классификации достигает 89.04%. Матрица ошибок выглядит следующим образом (+1 - значит, что комментарий токсичен):

	y_true = +1	**y_true = -1**
y_pred = +1	5410	1434
y_pred = -1	833	12999

Таблица 7: Матрица ошибок GD

Получим, что precision(точность) равно 0.7905, а recall(полнота) - 0.8666. Приведём примеры, на каких объектах ошибается наш алгоритм.

Наш алгоритм отнёс много положительных комментариев к токсичным (1434), из-за чего упала точность. Вот несколько таких комментариев. 'dear god this site be horrible', 'you be obviously nut just like this montel williams person i have never met a sane welshman', 'i will burn you to hell if you revoke my talk page access'. Они явно имеют негативный окрас, и вообще непонятно почему такие комментарии не являются токсичными...

Напротив, в таких комментариях, как 'you have my trust but trust me on this checkuser work will just eat you up burn you out and i guarantee you ll hate it seriously', 'you can t just say that s crap and remove it without gain a consensus you already know this base on your block history', 'agree it be stupid to say that any malfoy be albino', нет ничего токсичного. Наверное, модель отнесла такие комментарии к "плохим" из-за лексики, которая в них используются. Слова и выражения 'burn you out', 'hate', 'crap', 'stupid' и т.д. имеют отрицательный характер. Но контекст, котором они использованы, не имеет плохого смысла. К сожалению, нашему алгоритму это сложно выцепить...

Посмотрим теперь на комментарии, которые являются токсичными, но на которых наша модель даёт другой ответ. Такой пример, как 'if ya not still fu k u', возможно был воспринят нормальным, потому что в слове, которое всё портит, автор заменил одну букву на '*'. В связи с этим у нас появилось 2 разных слова, которых могло не быть в обучающей выборке и модель неправильно среагировала на такой комментраций. Ещё примерами являются 'shut up please this consensus be over' и 'oh hey fuckface i can go all day buddy'. Возможно, такие комментарии не были отнесены к токсичным из-за таких слов, как 'please', 'buddy', которые редко встречаются в негативных твитах. Отметим, что в первом из них (про consensus) наша модель максимально не уверена, т.к. выдаёт вероятность токсичности 48.65%, а во второму 40%. Так, что возможно стоит более внимательно смотреть на вероятности классификации.

Посмотрим теперь, как работает наилучшая модель SGD с $\alpha = 3.6$ и с теми же остальными параметрами.

Качество классификации достигает 88.67%. Матрица ошибок выглядит следующим образом (+1 - значит, что комментарий токсичен):

	y_true = +1	**y_true = -1**
y_pred = +1	5282	1381
y_pred = -1	961	13052

Таблица 8: Матрица ошибок GD

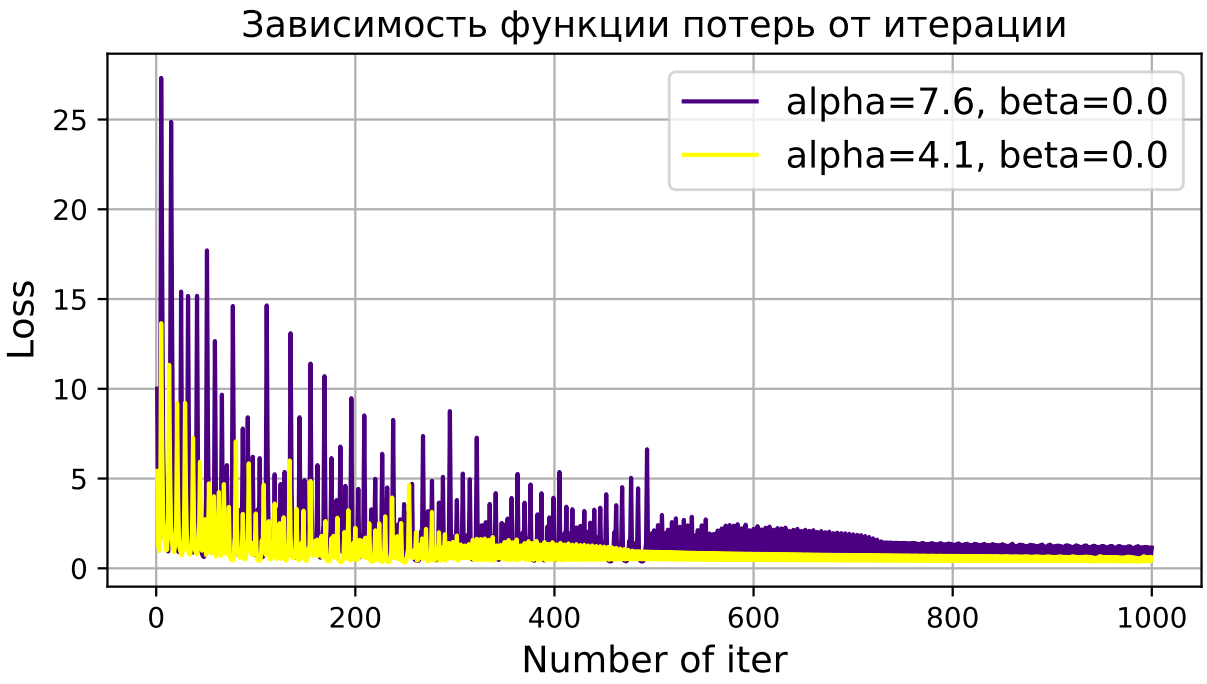
Получим, что precision(точность) равно 0.7927, а recall(полнота) - 0.8461. Качество и полнота немного упали, но точность чуть увеличилась. Мы не будем приводить примеры, т.к. они идентичны выше разобранным.

Заключение

В данной работе была подробно рассмотрена и разобрана логистическая регрессия для классификации токсичности комментариев. Были реализованы и проанализированы стандартный градиентный спуск и стохастический градиентный спуск. Были рассмотрены различные способы, улучшающие качество работы моделей (векторизация, лемматизация).

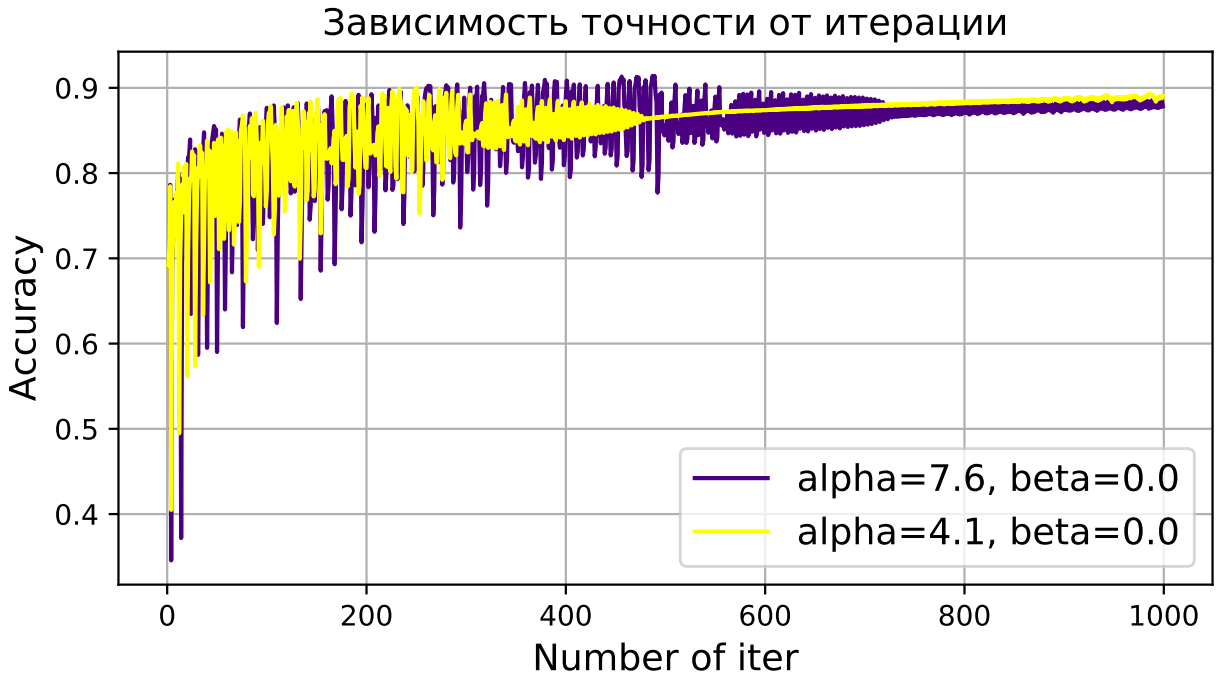
Приложение 1

В данном приложении приведены графики работы градиентного спуска

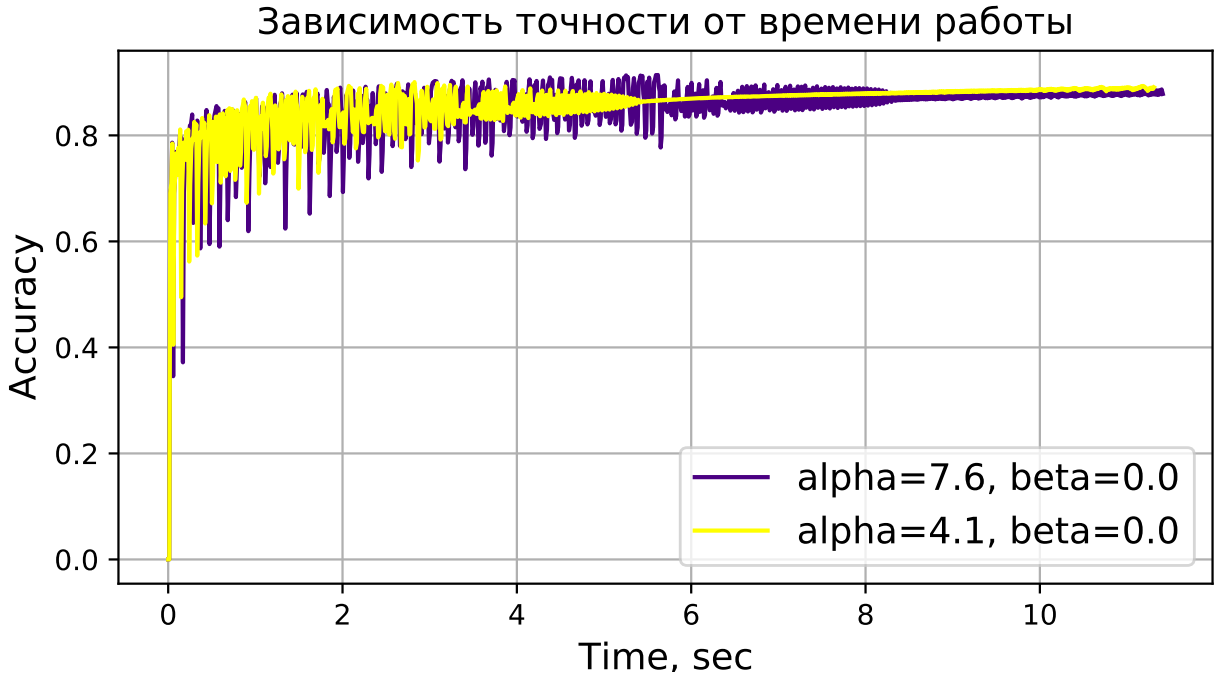


GD, $w = 0$

GD, $w = 0$


Зависимость точности от итерации
Accuracy
Number of iter
alpha=7.6, beta=0.0
alpha=4.1, beta=0.0

GD, $w = 0$


Зависимость точности от времени работы
Accuracy
Time, sec
alpha=7.6, beta=0.0
alpha=4.1, beta=0.0

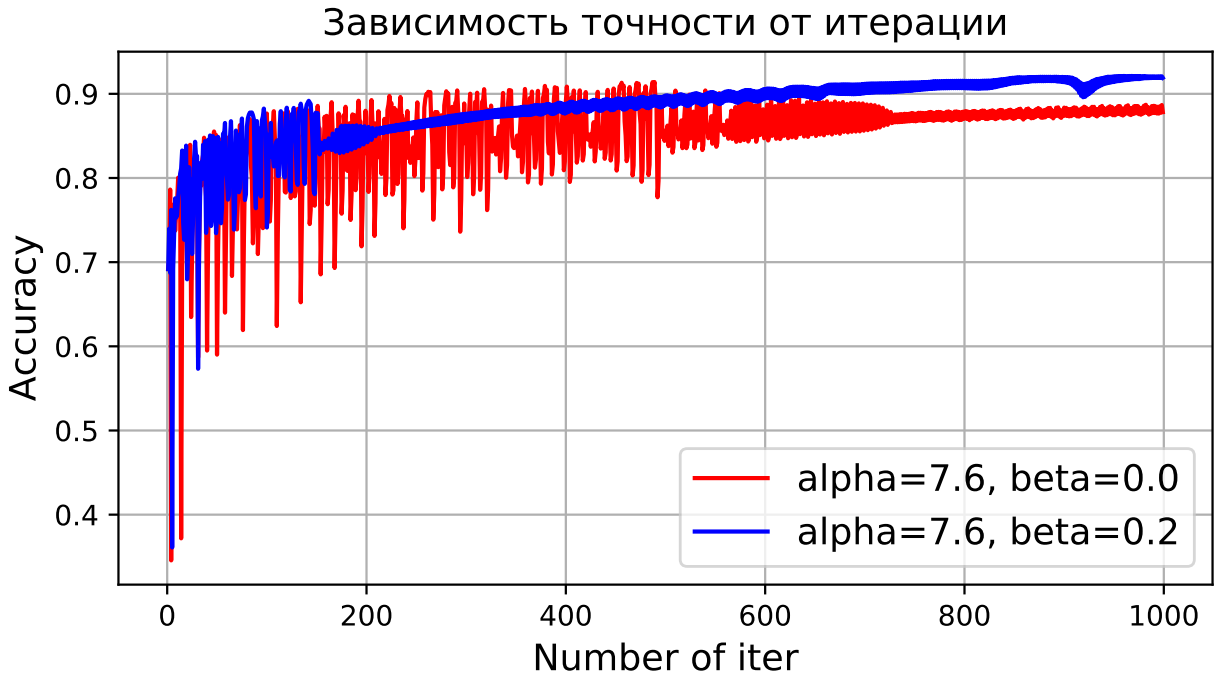
GD, $w = 0$


Зависимость функции потерь от итерации
alpha=7.6, beta=0.0
alpha=7.6, beta=0.2
Loss
Number of iter
0
5
10
15
20
25
0
200
400
600
800
1000

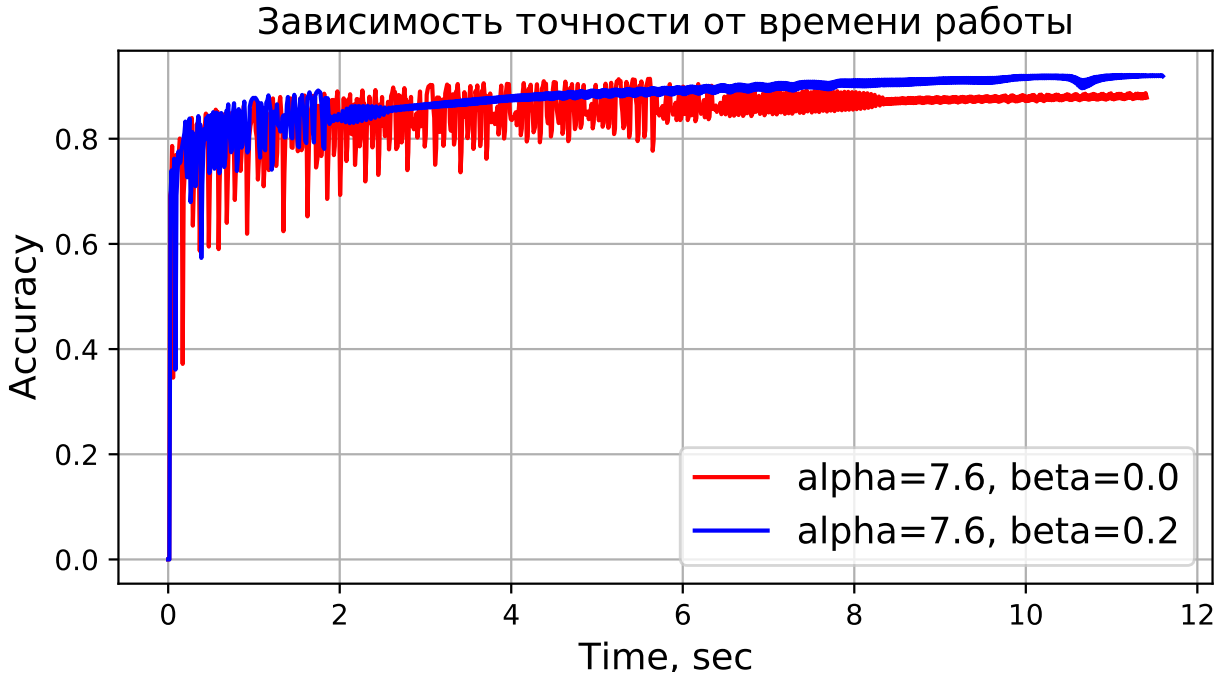
GD, $w = 0$


Зависимость функции потерь от времени работы
alpha=7.6, beta=0.0
alpha=7.6, beta=0.2
Loss
Time, sec
0
5
10
15
20
25
0
2
4
6
8
10
12

GD, $w = 0$

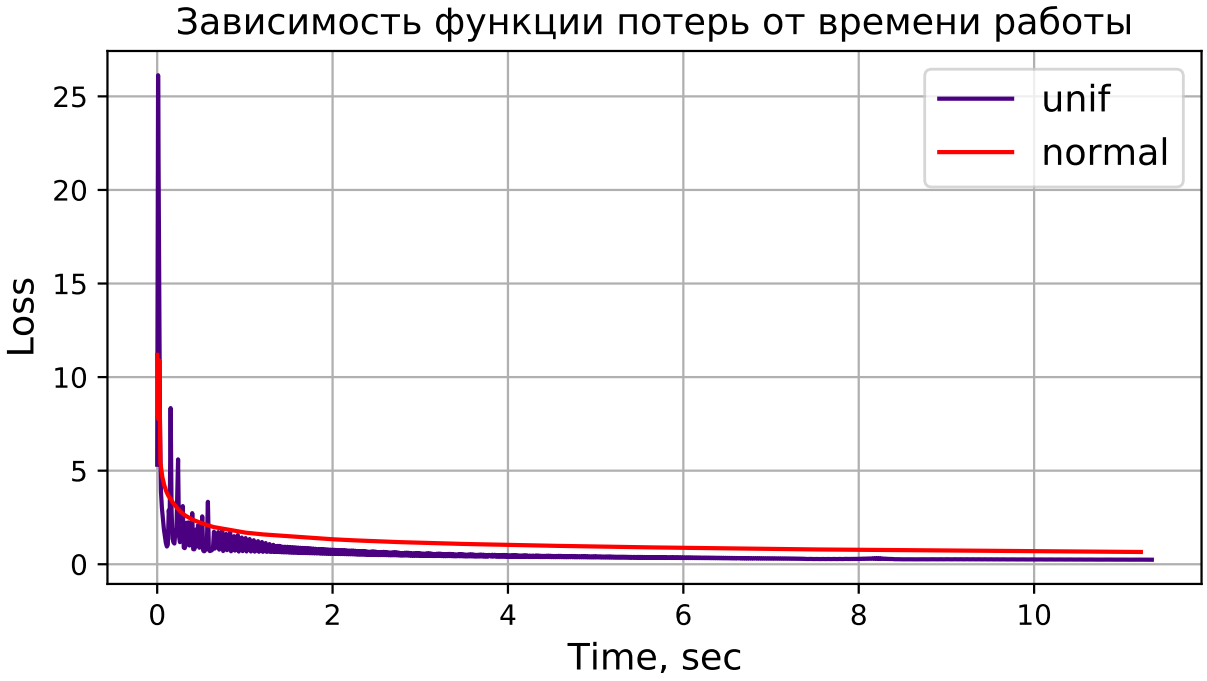

Зависимость точности от итерации
Accuracy
Number of iter
alpha=7.6, beta=0.0
alpha=7.6, beta=0.2

GD, $w = 0$

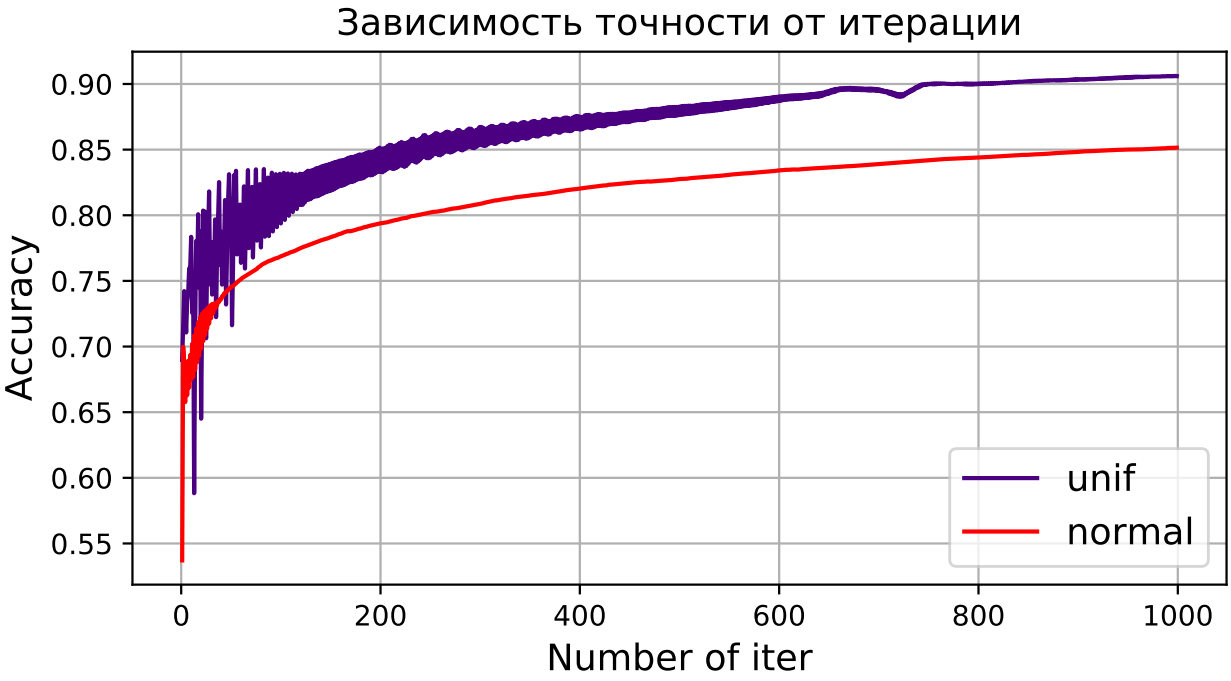

Зависимость точности от времени работы
Accuracy
Time, sec
alpha=7.6, beta=0.0
alpha=7.6, beta=0.2

GD, $w = 0$

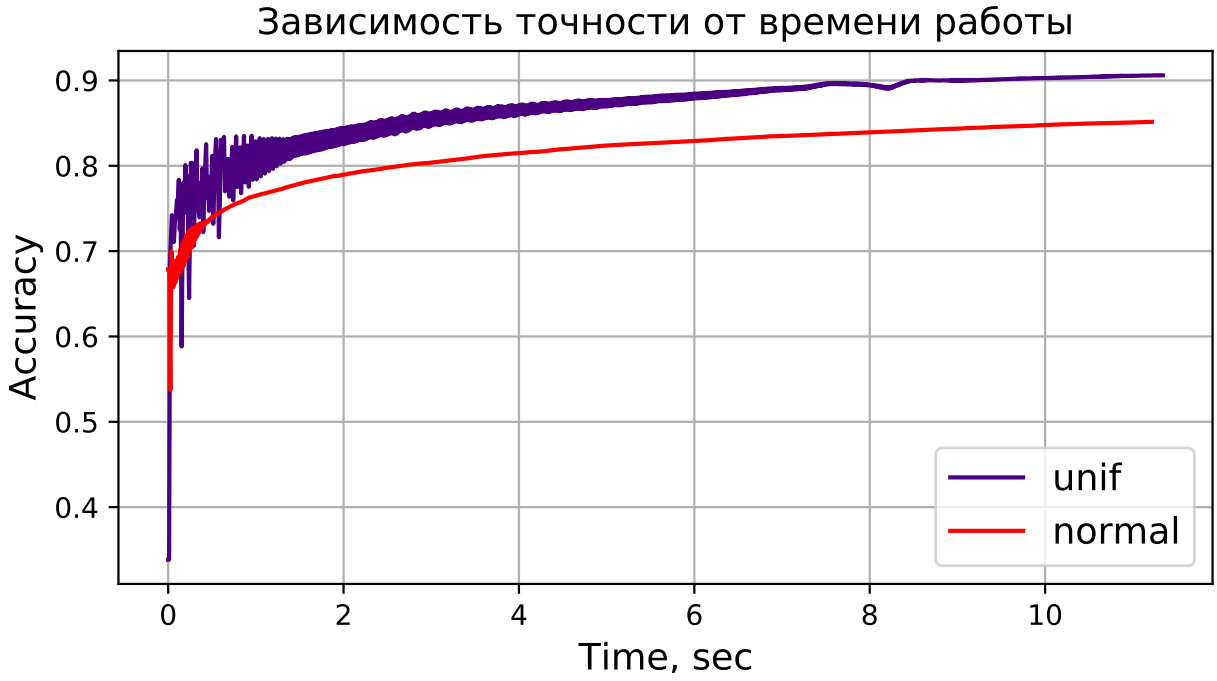
GD, $\alpha = 7.6, \beta = 0.2$



GD, $\alpha = 7.6, \beta = 0.2$



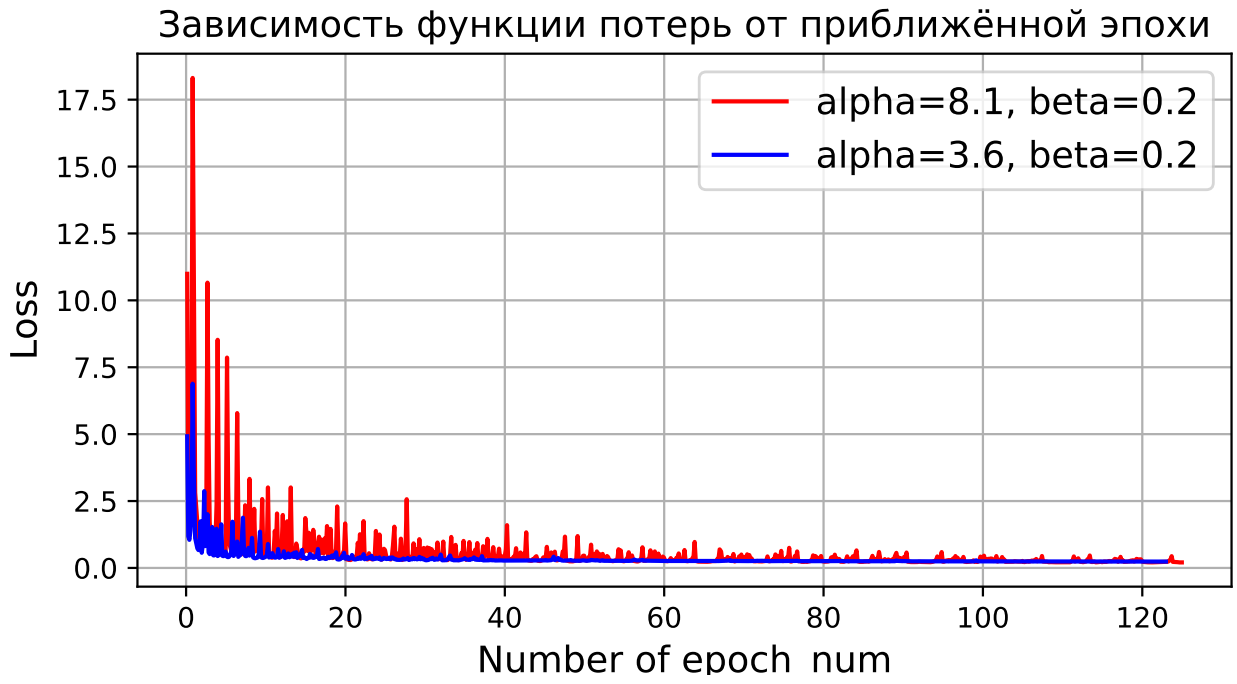
GD, $\alpha = 7.6, \beta = 0.2$



GD, $\alpha = 7.6, \beta = 0.2$

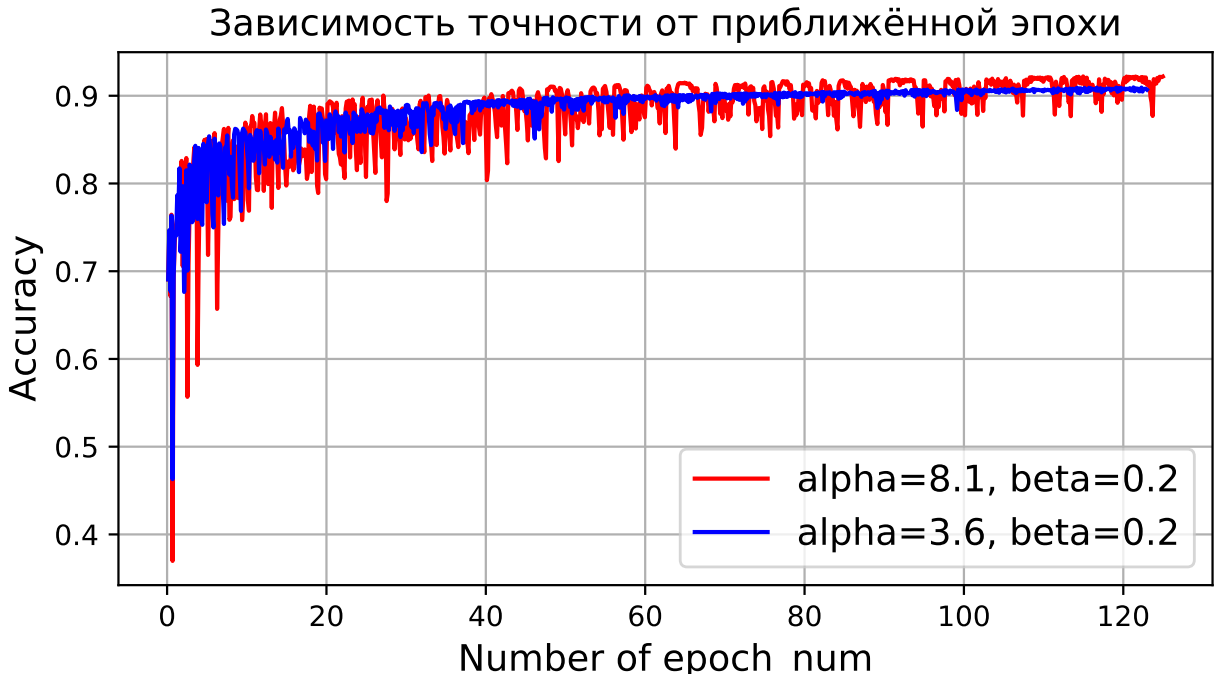
Приложение 2

В данном приложении приведены графики работы стохастического градиентного спуска

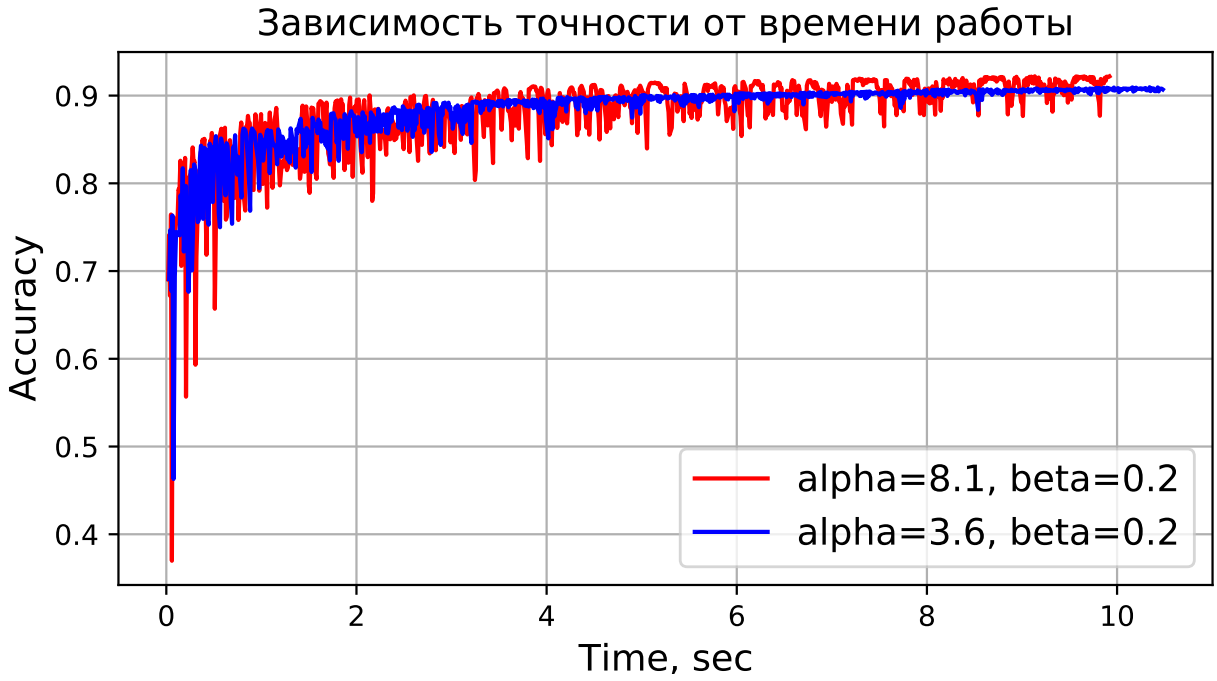


SGD, $w = 0, batch\_size = 5000$

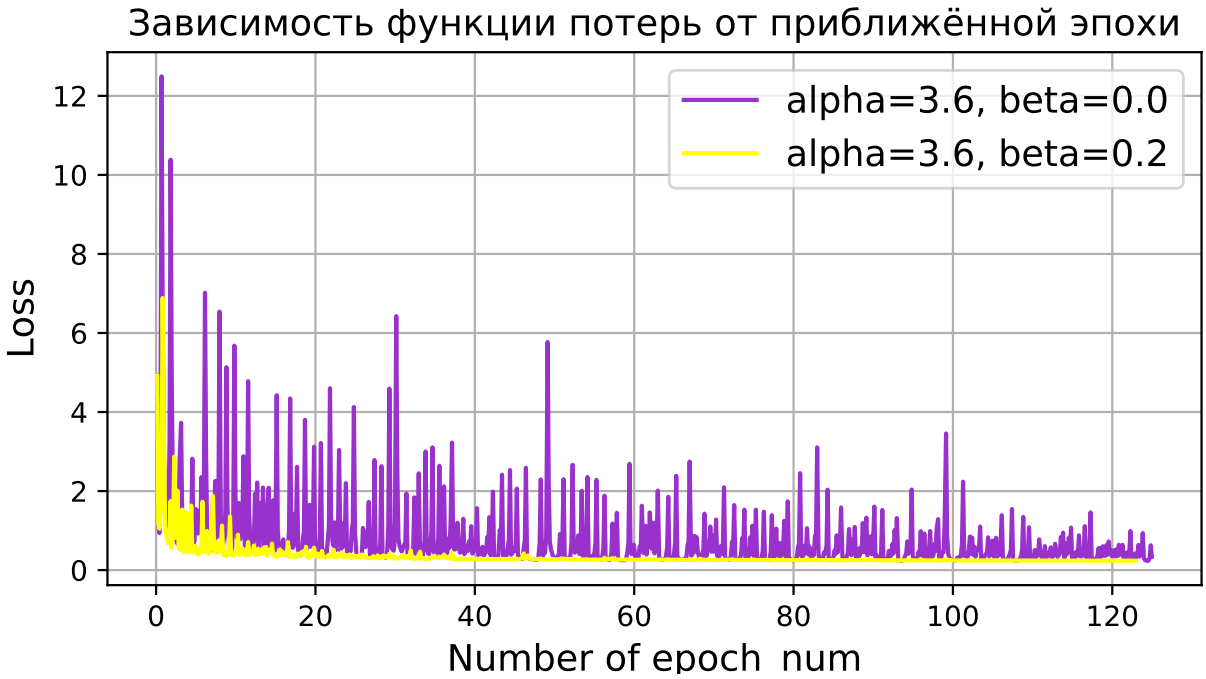
SGD, $w = 0, batch\_size = 5000$



SGD, $w = 0, batch\_size = 5000$

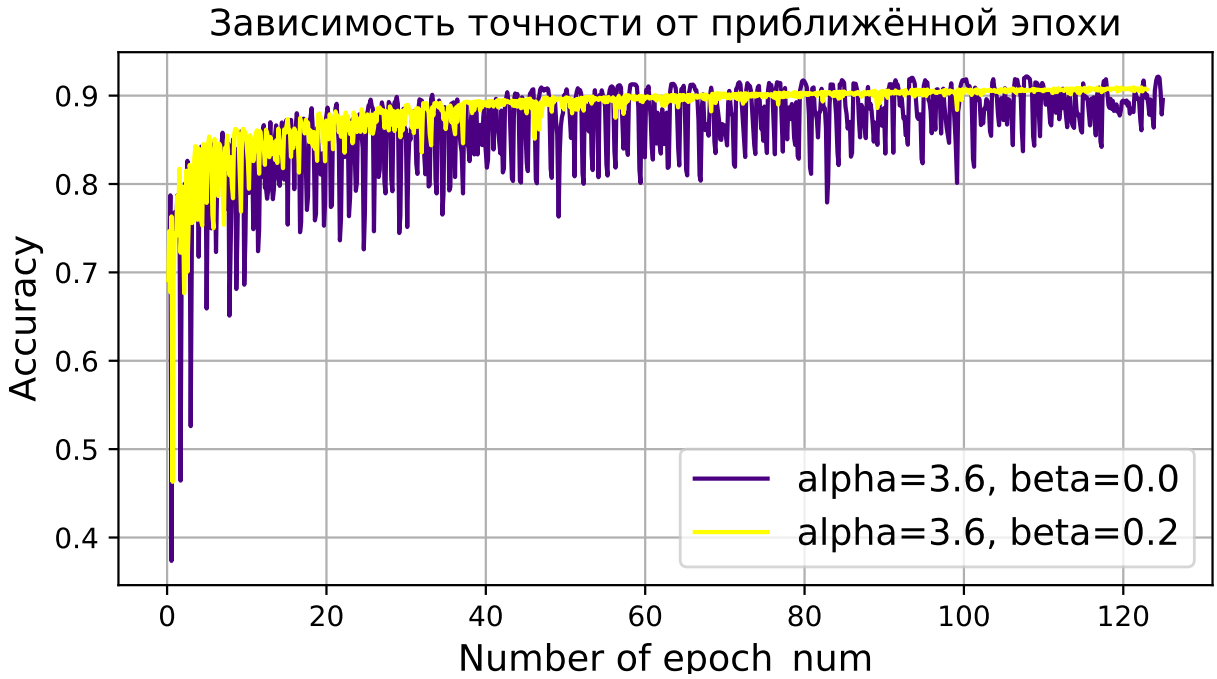


SGD, $w = 0, batch\_size = 5000$

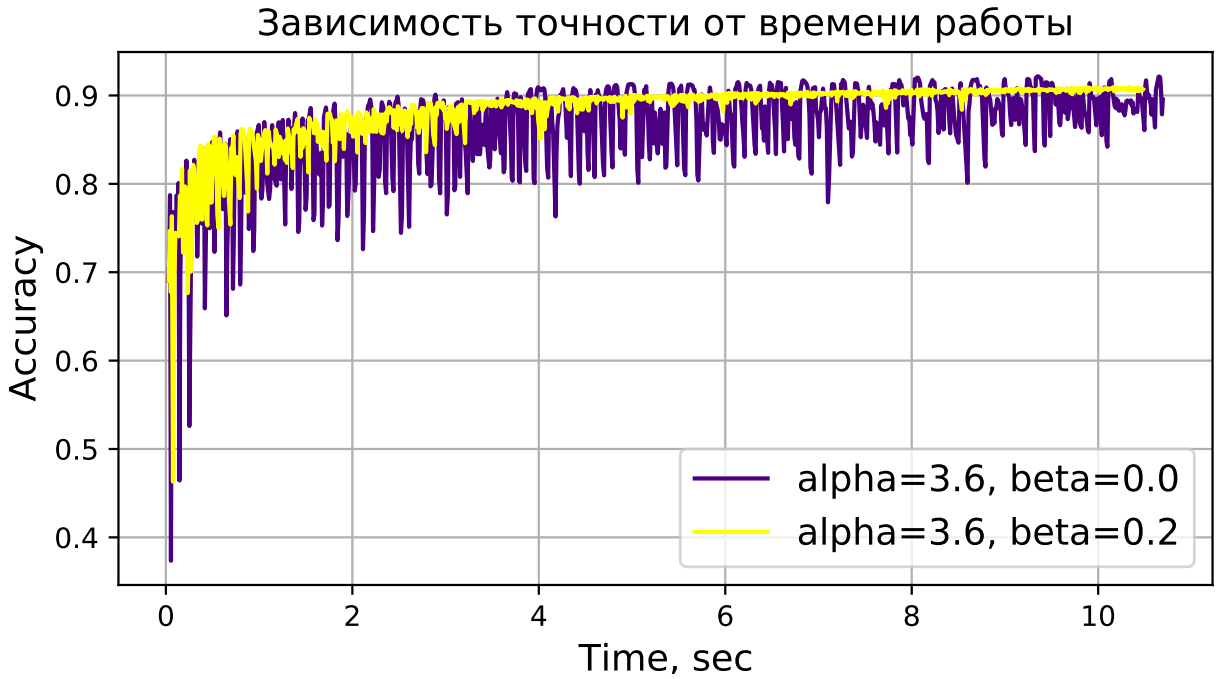


SGD, $w = 0, batch\_size = 5000$

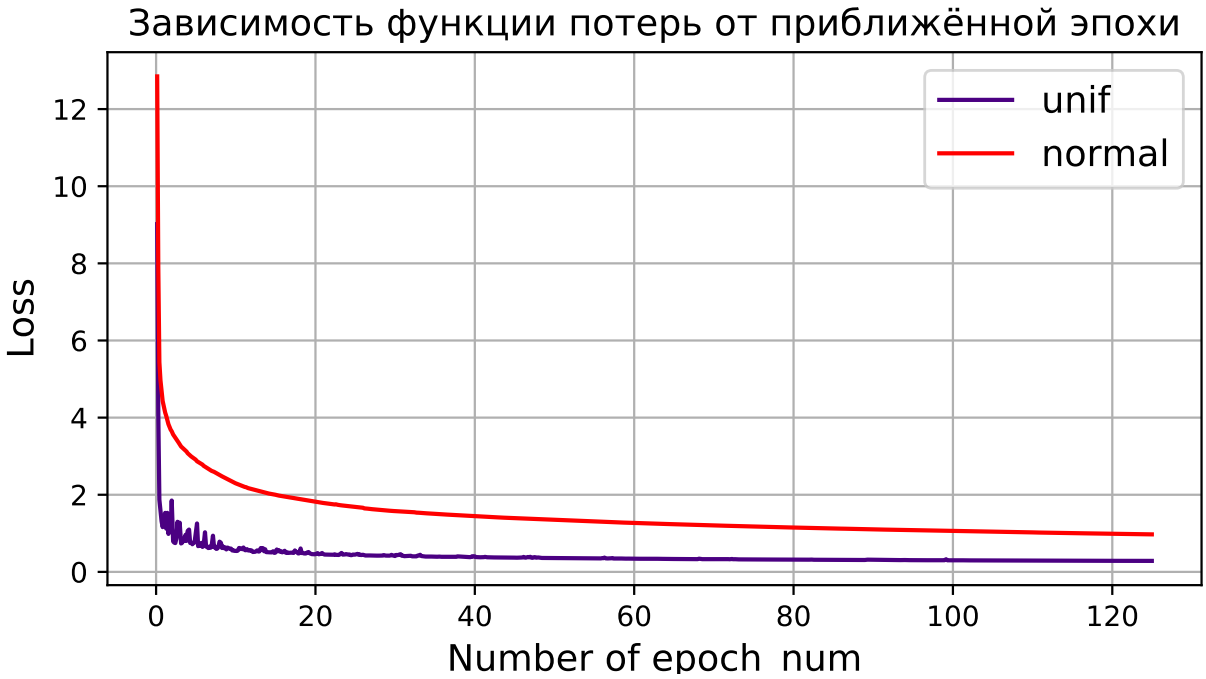
SGD, $w = 0, batch\_size = 5000$



SGD, $w = 0, batch\_size = 5000$

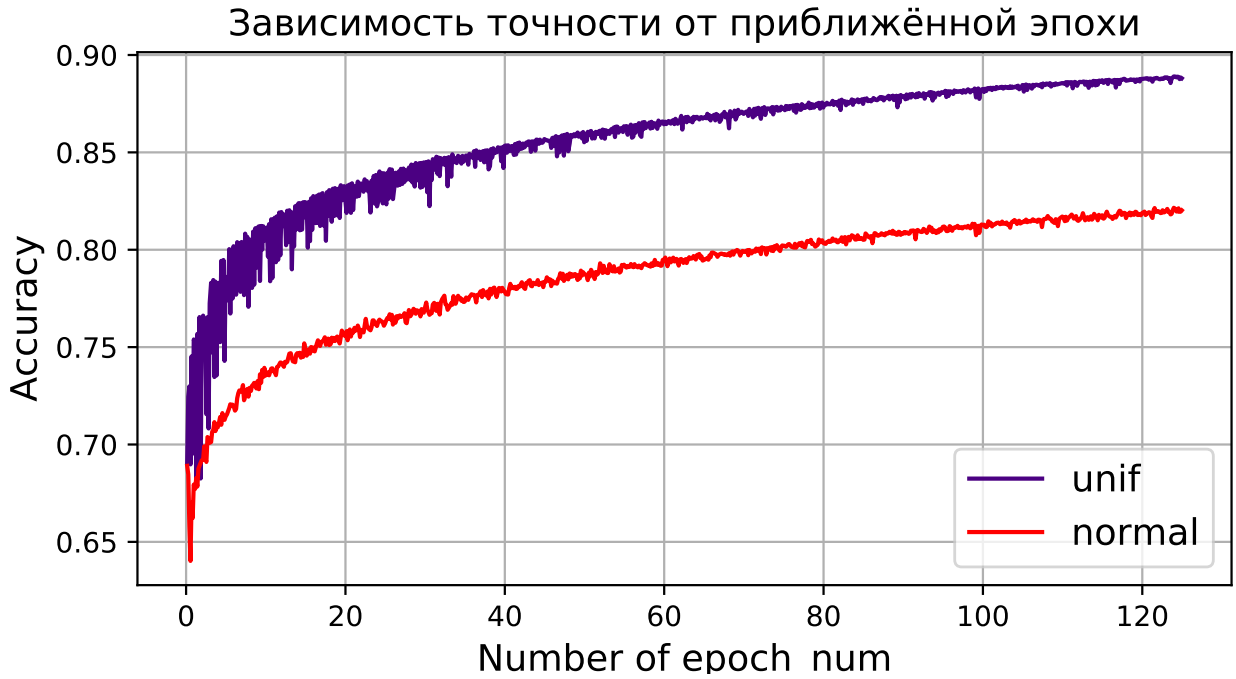


SGD, $w = 0, batch\_size = 5000$

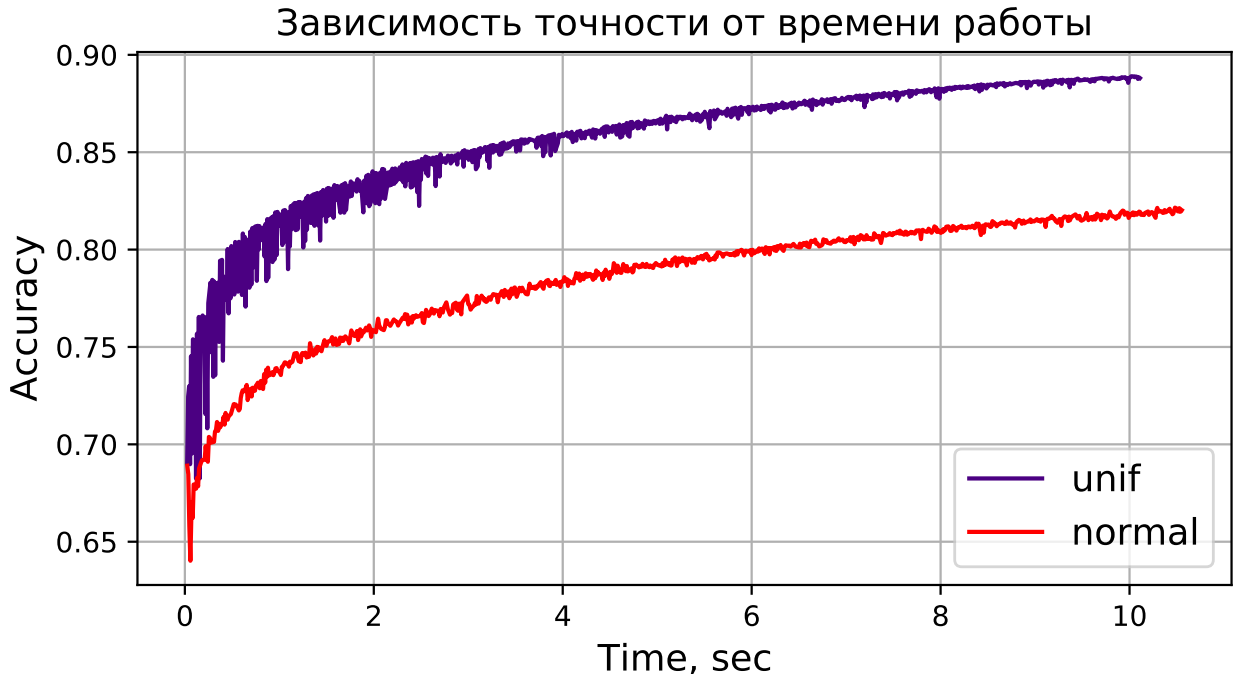


SGD, $\alpha = 3.6, \beta = 0.2, batch\_size = 5000$

SGD, $\alpha = 3.6, \beta = 0.2, batch\_size = 5000$



SGD, $\alpha = 3.6, \beta = 0.2, batch\_size = 5000$

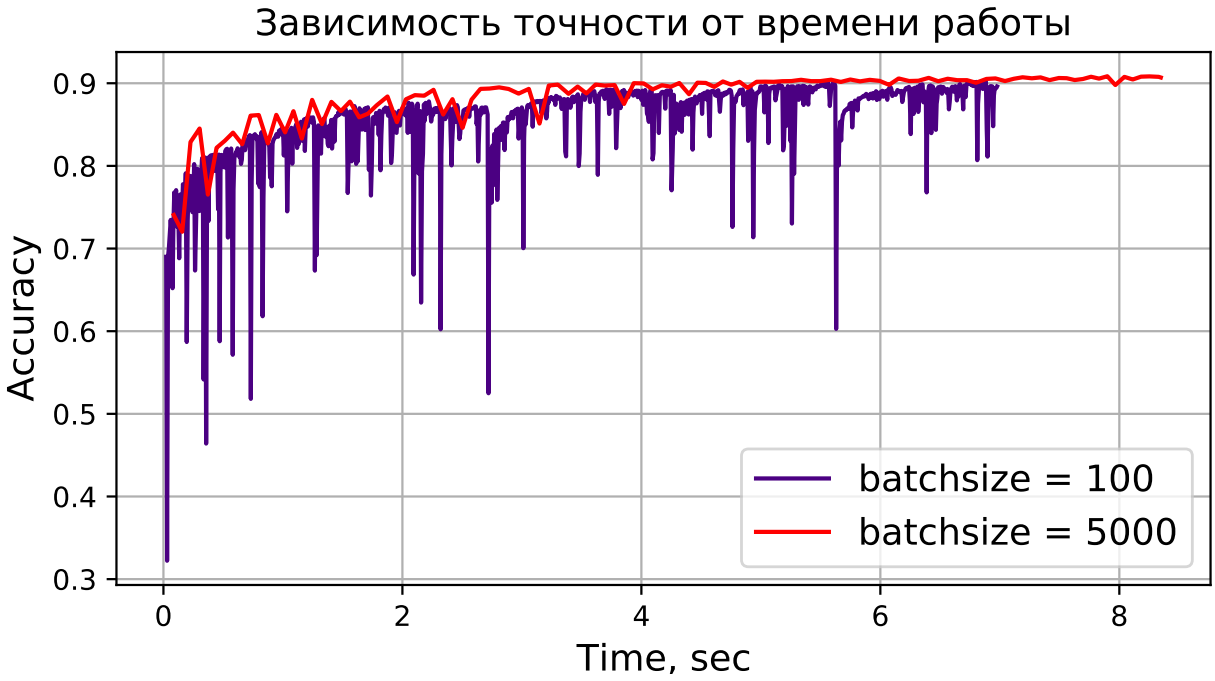


SGD, $\alpha = 3.6, \beta = 0.2, batch\_size = 5000$

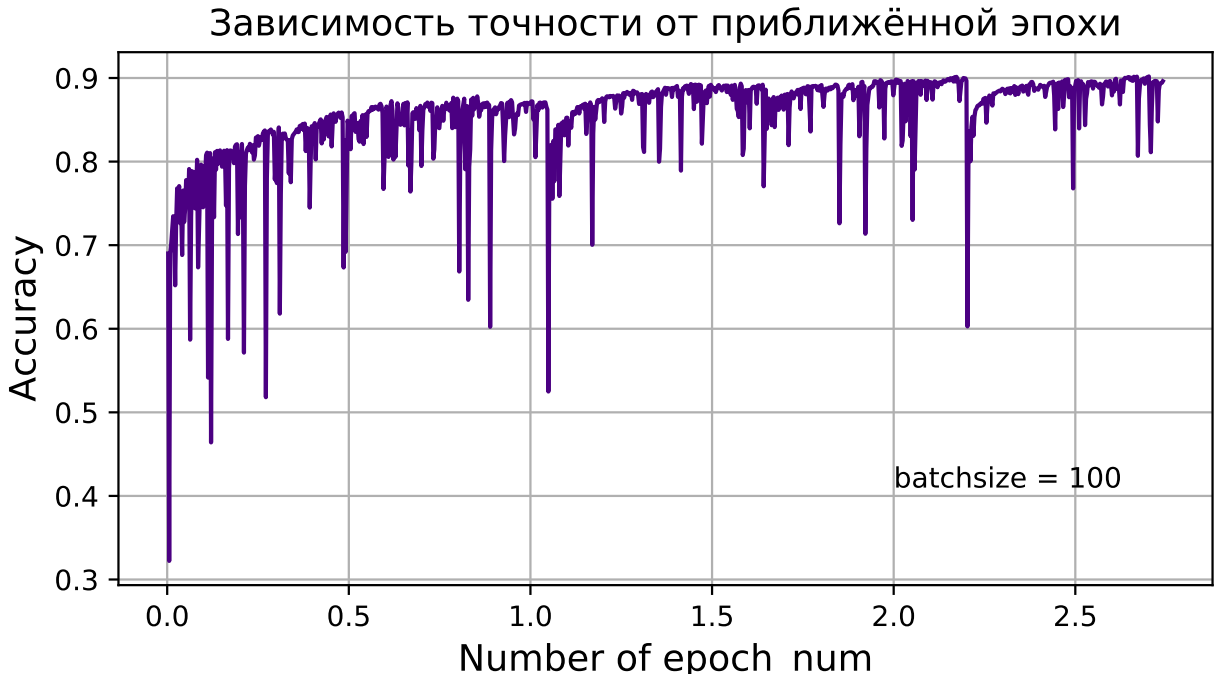
SGD, $\alpha = 3.6, \beta = 0.2, w_0 = 0$

SGD, $\alpha = 3.6, \beta = 0.2, w_0 = 0$

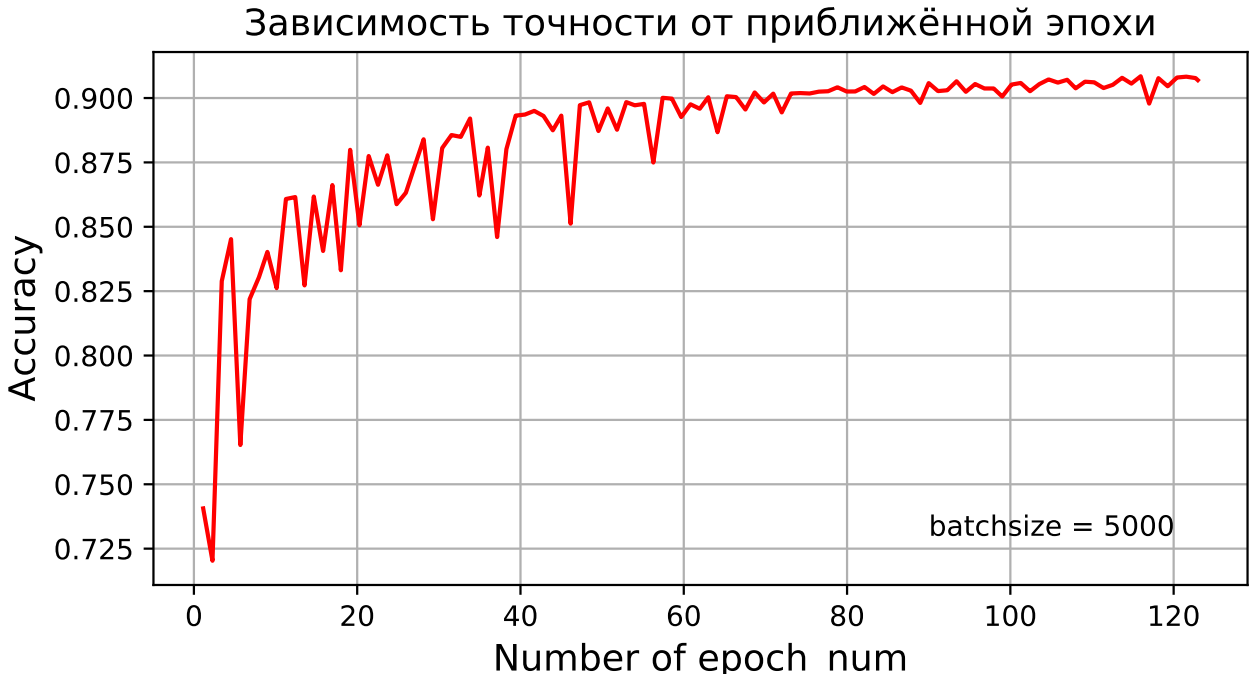
SGD, $\alpha = 3.6, \beta = 0.2, w_0 = 0$



SGD, $\alpha = 3.6, \beta = 0.2, w_0 = 0$



SGD, $\alpha = 3.6, \beta = 0.2, w_0 = 0$



SGD, $\alpha = 3.6, \beta = 0.2, w_0 = 0$