Отчет о практическом задании Градиентные методы обучения линейных моделей. Применение линейных моделей для определения токсичности комментария

Практикум 317 группы, ММП ВМК МГУ

Пшеничников Глеб Викторович

Москва
Ноябрь 2024

Содержание

1 Введение

Данное задание посвящено исследованию градиентных методов обучения линейных моделей и методов работы с текстами. Исследование будет проводиться на наборе комментариев из обсуждений английской Википедии *Toxic Comment Classification Challenge*[1]. Целью данной работы является сравнение градиентных методов обучения (градиентный спуск и стохастический градиентный спуск), сравнение методов обработки текстов, способов представления текстов в виде числовых признаков, подбор оптимальных параметров модели классификации для достижение лучшего качества предсказаний модели, сравнение способов вычисления градиентов (необходимых в данной задаче). Для анализа вышеперечисленных параметров, методов в работе строится модель классификации комментариев на токсичные и не токсичные.

2 Пояснение к задаче

Пусть имеется выборка объектов:

$$X = (x_i, y_i)_{i=1}^{l}, \text{ где } x_i \in \mathbb{R}^d, \ y_i \in \mathbb{Y} = \{1, -1\}$$

Требуется построить линейную модель классификации, которая будет делить комментарии на токсичные и нет. Будет строить модель следующего вида:

$$a(x) = sign(\langle w, x\rangle + b), \text{ где } w \in \mathbb{R}^d - \text{вектор весов}, b \in \mathbb{R} - \text{сдвиг}$$

То если, значение $a(x) \geq 0$, то предсказанное значение будет равно 1 (то есть комментарий токсичный), иначе метка 0 (комментарий не токсичный).

Дадим определение *отступу (margin)* объекта x_i относительно алгоритма $a(x)$:

$$M_i(w) = y_i(\langle w, x\rangle + b)$$

Эта величина показывает "уверенность"классификатора в предсказанном значении, она определяет расстояние от объекта до разделяющей гиперплоскости. Если $M_i(w) > 0$ - объект x_i правильно классифицирован , если $M_i(w) < 0$ - объект неправильно классифицирован. При этом, чем больше абсолютное значение отступа, тем увереннее классификатор в своем решении.

Пусть $L(M(w, b))$ - монотонная невозрастающая функция, такая что $[M(w, b) < 0] \leq L(M(w, b))$. Для решаемой задачи логистической регрессии (линейной классификации) возьмем следующую функцию потерь:

$$L(M) = \log(1 + \exp(-M))$$

При этом вероятность принадлежности объекта к каждому из классов (в решаемой задаче их два) будет иметь следующий вид:

$$p(y = 1 \mid x) = \frac{1}{1 + \exp(-\langle w, x\rangle + b)} = \sigma(\langle w, x\rangle + b))$$

Цель задачи - минимизировать функционал потерь:

$$Q(X, w, b) = \frac{1}{l}\sum_{i=1}^{l} L(M_i(w, b)) + \frac{\lambda}{2}\|w\|_2^2 \to \min_{w,b}$$

где при помощи коэфициента L_2 регуляризиции препятствуем переобучению модели.

В рамках данного исследования возьмем $b = 0$. Для того, чтобы минимизировать функционал $Q(X, w)$, необходимо реализовать алгоритм, изменяющий вектор весов (при которых функционал будет минимален). Для этой цели воспользуемся методом градиентного спуска:

$$w^{(k+1)} = w^{(k)} - \eta_k \frac{1}{l} \sum_{i=1}^{l} \nabla_w L_i$$

В рамках данной задачи $\eta_k = \frac{\alpha}{k^\beta}$. Параметры $\alpha, \beta, w^{(0)}$ - гиперпараметры (задаются вручную), подбор которых также будет осуществляться в данном исследование.

2.1 Теоретическая часть. Вычисление градиента функции потерь

Выведем формулы для вычисления градиента функции потерь для бинарной и многоклассовой логистической регрессии, а также укажем формулу для численного вычисления градиента.

2.1.1 Градиент для задачи бинарной логистической регрессии

Функция потерь в матричной форме:

$$L(w) = -\frac{1}{n}[y^T \log(\sigma(Xw)) + (1-y)^T \log(1-\sigma(Xw))]$$

$X \in \mathbb{R}^{n \times d}$ - матрица признаков (n объектов, d признаков), $w \in \mathbb{R}^{d \times 1}$ - вектор весов, $y \in \mathbb{R}^{n \times 1}$ - вектор меток классов, $Xw \in \mathbb{R}^{n \times 1}$ - результат умножения, $\sigma(z) = \frac{1}{1+e^{-z}}$ - сигмоидальная функция.

Пусть $h = Xw$ и $p = \sigma(h)$, тогда:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial p} \cdot \frac{\partial p}{\partial h} \cdot \frac{\partial h}{\partial w}$$

1) Найдем $\frac{\partial L}{\partial p}$:

$$\frac{\partial L}{\partial p} = -\frac{1}{n}\left[\frac{y}{p} - \frac{1-y}{1-p}\right] = -\frac{1}{n}\left[\frac{y-p}{p(1-p)}\right]$$

2) Найдем $\frac{\partial p}{\partial h}$:

$$\frac{\partial p}{\partial h} = \frac{\partial(\frac{1}{1+e^{-h}})}{\partial h} = \frac{e^{-h}}{(1+e^{-h})^2} = p \cdot \frac{1}{1+e^{-h}} = p \cdot (1 - \frac{1}{1+e^{-h}}) = p \cdot (1-p)$$

3) Найдем $\frac{\partial h}{\partial w}$:

$$\frac{\partial h}{\partial w} = \frac{\partial(Xw)}{\partial w} = X$$

Подставляя все части вместе:

$$\nabla_w L = \frac{1}{n} X^T (\sigma(Xw) - y)$$

2.1.2 Градиент для задачи многоклассовой логистической регрессии

Пусть теперь решается многоклассовая задача классификации, для которой было построено K линейных моделей.

Функция потерь для многоклассовой классификации:

$$L(w, X, y) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(\sigma_k(Xw_i)),$$

$X \in \mathbb{R}^{n\times d}$ - матрица признаков; $w \in \mathbb{R}^{d\times 1}$ - вектор весов; $P \in \mathbb{R}^{n\times K}$ - матрица вероятностей после применения softmax; $W \in \mathbb{R}^{d\times K}$ - матрица весов (K - количество классов); y_{ij} равно 1, если объект i принадлежит классу j, и 0 иначе;

$$\sigma_k(h) = \frac{e^{h_k}}{\sum_{i=1}^{K} e^{k_i}}, \quad k = 1, \ldots, K,$$

- функция softmax ($Xw = h$, а $h_k = Xw_k$.).

Пусть $Xw = h$, $\sigma(h) = p$, тогда:

$$\frac{\partial L}{\partial w_k} = \frac{\partial L}{\partial p}\cdot\frac{\partial p}{\partial h_k}\cdot\frac{\partial h_k}{\partial w_k}.$$

1) Найдём $\frac{\partial L}{\partial p}$:

$$\frac{\partial L}{\partial p_{ij}} = \frac{\partial\left(-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{K} y_{ij}\log(p_{ij})\right)}{\partial p_{ij}} = -\frac{1}{n}\cdot\frac{y_{ij}}{p_{ij}}$$

$$\frac{\partial L}{\partial p} = -\frac{1}{n}\cdot\frac{y}{p}$$

2) Найдем $\frac{\partial p}{\partial h}$. Для элемента p_{ij} (вероятность i-го объекта принадлежать к j-му классу) при дифференцировании по h_{ik} получаем два случая:

$$\frac{\partial p_{ij}}{\partial h_{ik}} = \frac{\partial \frac{e^{h_{ij}}}{\sum_{k=1}^{K} e^{h_{ik}}}}{\partial h_{ik}} = \begin{cases} p_{ij}(1-p_{ij}), & \text{если } j = k \\ -p_{ij}p_{ik}, & \text{если } j \neq k \end{cases}$$

$$\frac{\partial p}{\partial h_k} = \begin{pmatrix} \frac{\partial p_1}{\partial h_k} \\ \frac{\partial p_2}{\partial h_k} \\ \vdots \\ \frac{\partial p_K}{\partial h_k} \end{pmatrix} = \begin{pmatrix} -p_1p_k \\ -p_2p_k \\ \vdots \\ p_k(1-p_k) \\ \vdots \\ -p_Kp_k \end{pmatrix}$$

3) Частная производная $\frac{\partial h_k}{\partial w_k}$:

$$\frac{\partial h_k}{\partial w_k} = \frac{\partial Xw_k}{\partial w_k} = X$$

Тогда для класса k:

$$\frac{\partial L}{\partial w_k} = \frac{1}{n}\cdot X^T\cdot(p-y),$$

где p — это матрица вероятностей после softmax-функцией

$$\nabla_w L = \frac{1}{n}X^T(P-Y),$$

2.1.3 Сведение к бинарному случаю при K=2

При K=2 имеем матрицу весов $W = [w_1 w_2]$ и:

$$P = [p_1 p_2] = \left[\frac{e^{Xw_1}}{e^{Xw_1} + e^{Xw_2}} \quad \frac{e^{Xw_2}}{e^{Xw_1} + e^{Xw_2}} \right]$$

Функция потерь:

$$L(W) = -\frac{1}{n} \mathrm{tr}(Y^T \log(P))$$

Введем $w = w_1 - w_2$, тогда:

$$p_1 = \sigma(Xw)$$
$$p_2 = 1 - \sigma(Xw)$$

Подставляя в функцию потерь:

$$L(w) = -\frac{1}{n}[y^T \log(\sigma(Xw)) + (1-y)^T \log(1 - \sigma(Xw))]$$

что соответствует бинарной логистической регрессии

2.1.4 Формула для численного подсчета градиента

Градиент можно вычислить численно при помощи формулы:

$$[\nabla f(w)]_i \approx \frac{f(x + \varepsilon e_i) - f(x)}{\varepsilon}$$

$e_i = (0, 0, \ldots, 0, 1, 0, \ldots, 0)$ – базисный вектор, $\varepsilon > 0$ – небольшое положительное число.

2.2 Метод градиентного спуска

Рассмотрим два метода градиентного спуска: "обычный"и стохастический градиентные спуски (в рамках данного исследования будет проведено их сравнение).

2.2.1 Градиентный спуск

Алгоритм заключается в следующем:

- Вычислить значение градиента функции потерь
- Домножить градиент на соответствующий коэфициент
- Вычесть полученное значение из текущих весов
- Выполнить все заново, пока разность между значениями функции потерь на подряд идущих итерациях не станет меньше заданного значения или пока алгоритм не превысит максимально допустимое число итераций

2.2.2 Стохастический градиентный спуск

Алгоритм схож с градиентным спуском, за исключенем некоторых моментов. Изначально делим выборку на части (на батчи), предварительно перемешав тренировочную выборку. Затем вычисляем градиент на перовом батче (а не на всей выборке) и оптимизируем веса, затем на втором и так далее. Когда батчи закончатся - снова мешаем выборку и делим на батчи. Критерии выхода из алгоритма такие же, как в градиентом спуске.

2.3 Точность предсказаний

Качество модели будет проверяться при помощи метрики *accuracy*:

$$accuracy = \frac{\text{кол-во правильных предсказаний}}{\text{общее кол-во предсказаний}}$$

3 Эксперименты

В рамках данного исследования было проведено 9 экспериментов, каждое из которых сравнивает методы, метрики создаваемой модели бинарной классификации или улучшает их, также часть экспериментов посвящено работе с выборками текстов, на которых модель учится и тестируется.

3.1 Предварительная обработка данных

Данные, используемые в исследовании ([1]), представляют собой набор текстовых комментариев с меткой о токсичности/о не токсичности. Так как эти комментарии писали люди, имеет место наличие заглавных букв знаков препинания, повторяющихся пробелом. Распределение символов в тренировочной выборке представлено в таблице:

	Количество	**Часть от общего числа символов**
Всего символов	19449642	100%
Заглавные буквы	1211347	6.23%
Строчные буквы	13809506	71.00%
Цифры	164207	0.84%
Один пробел	3439551	17.68%
Более одного пробела (подряд)	51033	0.26%
Знаки пунктуации	758388	3.90%
Другие символы	66643	0.35%

Таблица 1: Распределение символов в тренировочной выборке

	Количество	**Часть от общего числа символов**
Всего символов	7296607	100%
Заглавные буквы	507982	6.96%
Строчные буквы	4973361	68.16%
Цифры	46969	0.64%
Один пробел	1347448	18.47%
Более одного пробела (подряд)	26621	0.36%
Знаки пунктуации	318136	4.36%
Другие символы	102711	1.41%

Таблица 2: Распределение символов в тестовой выборке

Как видно из таблиц, заглавные буквы, знаки пунктуации, прочие символы, подряд идущие пробелы (более одного) встречаются в тексте и составляют суммарно около 12% всех символов. Для модели бинарной классификации одно и то же слово, написанное с заглавной и со строчной буквы, или с "приклееным"знаком препинания (или пробелом, или другим символов) или без

них, будет представляться как два разных (хоть на самом деле это одно слово), что может значительно понизить точность классификации. Поэтому в тренировочной и тестовой приведем все символы к нижнему регистру, заменим все символы, не являющиеся буквой или цифрой, на пробелы, и затем заменим подряд идущие пробелы одним пробелом.

3.2 Преобразование выборки в разреженную матрицу

В создаваемая модели классификации в качестве данных для обучения и последующих предсказаний надо подавать на вход численные признаки. У нас есть текста (состоящие из слов), которые необходимо векторизовать (представить в виде числовых признаков), чтобы подавать на вход модели. Для этого преобразуем выборки в разряженную матрицу, где значение x в позиции *(i, j)* означает, что в документе i слово j встречалось x раз. Данных подход к векторизации называется ***Bag of Words***. То есть мы предположили, что порядок токенов (слов) в тексте не важен, важно лишь сколько раз оно входило в документ.

Для создания словаря (мешка слов) векторизатору подается на вход тренировочная выборка, затем для тренировочной и тестовой выборок создается матрица исходя из слов, которые есть в словаре. Также в данном алгоритме можно задать параметр, который будет отвечает за следующее: слово будет включено в словарь, если оно суммарно встречается во всех документах не менее заданного числа раз. Справедливо, что этот параметр будет влиять на размер признакового пространства (так как меньше или наоборот больше слов попадет в словарь). Зависимость размерности пространства от этого параметра (min_df) представлена в таблице:

min df	**Размерность признакового пространства**
1	89 657
2	37 793
3	26 739
4	21 390
5	18 252
6	16 050
7	14 400
8	13 121
9	12 101
10	11 237
11	10 563
12	9 995
13	9 525
14	9 068
15	9 068

Таблица 3: Зависимость размерности признакового пространства от параметра min_df

Как видно из таблицы при увеличении значения min_df размерность пространства уменьшается (следовательно потребуется меньше время и вычислительных мощностей для проведения экспериментов). В дальнейшем будем работать с выборками, преобразованным представленным алгоритмом с $min\_df = 500$ (признаковое пространство будет иметь размерность 586)

3.3 Вычисление градиента

В процессе исследования были реализованы методы градиентного и стохастического градиентного спуска, используя формулы из 2.1) (их анализ представлен в 3.4 и 3.5). Также реализован численный подсчет градиента функции потерь (по формуле из 2.1.4). Сравним аналитический и численный подсчет градиента.

Вычисление градиента проводилось на одних и тех же данных, для одного и того же вектора весов w (размерностью 586, заполненного случайными значениями). По аналитической формуле программа подсчитала градиент за: *0.3132*. Формула численного подсчета была запущена несколько раз с разными значениями ϵ (из формулы из 2.1.4) - считалось время работы и норма разности полученного значения и значения по аналитической формуле. Результаты представлены в таблице:

ϵ	**Время работы**	**Норма разности**
10^{0}	113.87	48.4165
10^{-1}	121.63	4.8417
10^{-2}	107.21	0.4842
10^{-3}	101.98	0.0484
10^{-4}	101.26	0.0048
10^{-5}	101.47	0.0005
10^{-6}	102.51	$4.74 \cdot 10^{-5}$
$\mathbf{10^{-7}}$	**103.13**	$\mathbf{9.18 \cdot 10^{-6}}$
10^{-8}	103.11	$1.27 \cdot 10^{-4}$
10^{-9}	102.46	$1.33 \cdot 10^{-3}$
10^{-10}	104.15	$1.30 \cdot 10^{-2}$

Таблица 4: Зависимость времени работы и нормы разности от степени

Из полученных результатов видно, что численное значение менее всего отличается от аналитического при значении $\epsilon = 10^{-7}$. При уменьшении или увеличении этого значения разница растет, это можно объяснить тем, что при большом приращении параметра или наоборот слишком маленьком

Из полученных результатов видно, что численное значение менее всего отличается от аналитического при значении $\epsilon = 10^{-7}$. При уменьшении или увеличении этого значения разница растет, это можно объяснить следующими рассуждениями:

- При большом ϵ линейное приближение производной становится неточным, так как формула конечных разностей основана на линейной аппроксимации функции в *окрестности точки*.
- При слишком маленьком ϵ начинают проявляться ошибки округления при вычислении разности близких чисел, что приводит к *увеличению погрешности*.

3.4 Градиентный спуск

В данном эксперименте будет проведен анализ градиентного спуска, а именно:

- Влияние параметра размера шага *step_ alpha* на точность предсказаний, функцию потерь, время работы метода

- Влияние параметра размера шага *step_ beta* на точность предсказаний, функцию потерь, время работы метода
- Влияние начального приближения $w^{(0)}$ на точность предсказаний, функцию потерь, время работы метода

То есть в данном эксперименте будут подобраны оптимальные гиперпараметры для градиентного спуска.

3.4.1 Размера шага *step_ alpha*

В данном эксперименте рассматривались значения *step_ alpha* от 0 до 1 с шагом 0.5. При этом остальные гипераметры были постоянны:

- $step\_beta = 0.1$
- $tolerance = 10^{-6}$ - разность между функциями потерь, вычисленных на подряд идущих итерациях, при достижении которой алгоритм заканчивается
- $max_iter = 500$ - максимально допустимое число итераций
- $\lambda = 0.1$- коэфициент L_2 регуляризации
- $w^{(0)} = [0, .., 0]$ - начальное приближение

Рассмотрим значения функции потерь, с которыми алгоритм закончился:

step_ alpha	**Функция потерь**
0.05	0.54807
0.10	0.54771
0.15	0.54769
0.20	0.54767
0.25	0.54767
0.30	0.54766
0.35	0.54766
0.40	0.54766
0.45	0.54765
0.50	0.54765
0.55	0.54765
0.60	0.54765
0.65	0.54765
0.70	**0.54764**
0.75	**0.54764**
0.80	0.54782
0.85	0.54776
0.90	0.55239
0.95	0.56042
1.00	0.56781

Таблица 5: Зависимость значения функции потерь от параметра *step_ alpha*

Можно сделать следующие выводы:

- Минимальное значение функции потерь достигается при $step\_alpha = 0.70$ и 0.75 и равно 0.54764
- При $step\_alpha < 0.70$ значения функции потерь наблюдается плавное уменьшение значения функции при увеличении tep_alpha
- При $step\_alpha > 0.70$ наблюдается более резкий рост значения функции потерь

Для каждого значения $step\_alpha$ рассмотрим зависимость значения функции потерь от номера итерации метода (полный набор графиков можно посмотреть в 32) и выделим три характерных поведения метода, для более детального анализа:

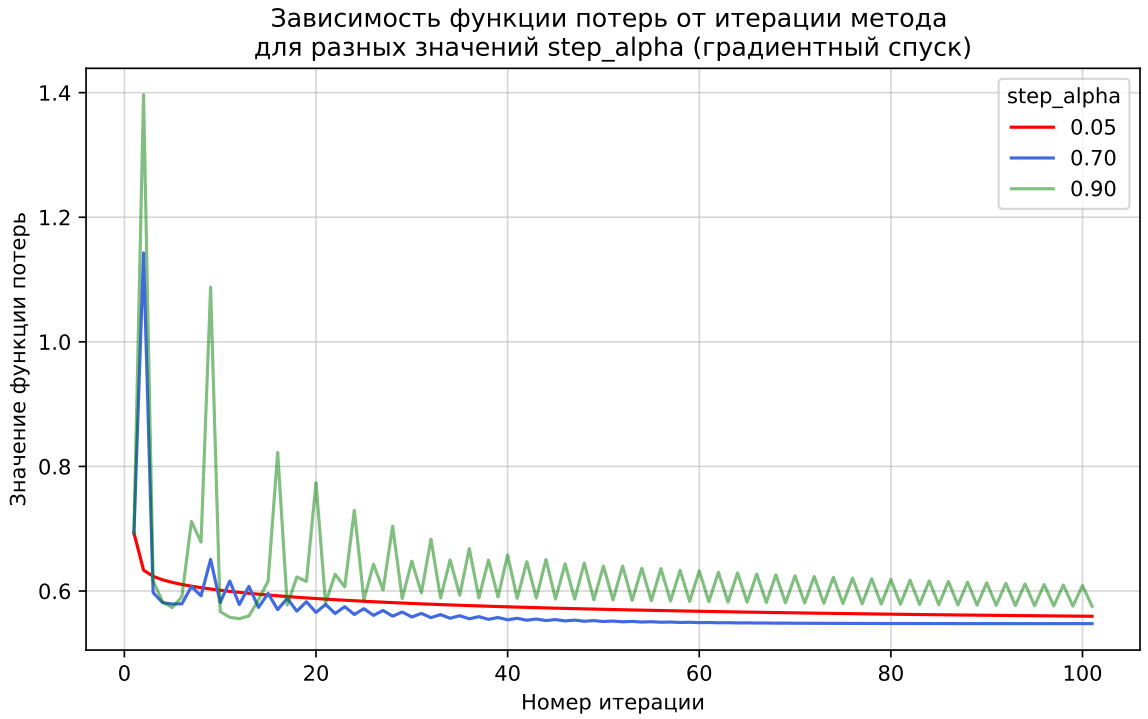


График 1: Зависимость функции потерь от итерации метода для разных значений $step\_alpha$

Из графика видно, что при малом значении $step\_alpha$ (от 0.05 до 0.35) функция потерь плавно уменьшается, это связно с тем, что в силу небольшого значения $step\_alpha$ веса меняются плавно (а значение функции потерь плавно приближается к минимальному значению). При средних значениях $step\_alpha$ (от 0.40 до 0.75)(при 0.7 было наименьшее значение функции потерь) функция потерь по-началу резко меняется но постепенно, начиная с 40-ой итерации значения плавно уменьшаются - такое поведение можно объяснить тем, что на первых итерациях метод резко меняет вектор весов (так как значения весов сильно отличатся от искомых и значение $step\_alpha$ не слишком маленькое), но постепенно, когда веса становятся близки к искомым (при которых функция потерь будет минимальны), то они меняются не сильно и плавно (так как $step\_alpha$ так же является не слишком большым значением). При больших значениях $step\_alpha$ (от 0.80 до 1) значения функции потерь резко меняются (на протяжении всех итераций) – это связано с тем, что так как значение этого гиперпараметра большое, то функция перепрыгивает точку минимума (не может достаточно близко к ней приблизиться). Так же подтверждением этого вывода является тот факт, что при больших $step\_alpha$ (и при

малых) происходит больше итераций, чем при средних - так как функция потерь перепрыгивает значение минимума (не успевает дойти)

Рассмотрим, как зависит точность от значения $step\_alpha$: результаты представленя в Таблице 6. Можно сделать следующие выводы:

- Максимальная точность достигается при $step\_alpha = 0.85$ и составляет 0.79416
- При малых значениях $step\_alpha$ ($step\_alpha < 0.35$) наблюдается постепенный рост точности с увеличением $step\_alpha$ (прирост точности замедляется после $step\_alph = 0.30$). Точность не максимальна, так как в силу маленьких изменений весов (и как следствие малых изменения функции потерь) метод завершится раньше, чем большая точность будет достигнута (так как разность соседних функций потерь уменьшается)
- В диапазоне $step\_alpha \in [0.35, 0.65]$ Точность стабилизируется на значении 0.79353
- В диапазоне $step\_alpha \in [0.70, 0.85]$ происходит небольшой скачок точности до 0.79401 при $step\_alpha = 0.70$
- При $step\_alpha > 0.85$ наблюдается резкое падение точности (так как метод становится нестабильным из-за большого коэфициента изменения весов)

step_ alpha	**accuracy**
0.05	0.79034
0.10	0.79242
0.15	0.79305
0.20	0.79324
0.25	0.79334
0.30	0.79343
0.35	0.79353
0.40	0.79353
0.45	0.79353
0.50	0.79353
0.55	0.79353
0.60	0.79353
0.65	0.79353
0.70	0.79401
0.75	0.79406
0.80	0.79406
0.85	**0.79416**
0.90	0.78705
0.95	0.77844
1.00	0.77118

Таблица 6: Зависимость точности классификации от параметра $step\_alpha$

Для каждого значения $step\_alpha$ рассмотрим зависимость точности от номера итерации метода (полный набор графиков можно посмотреть в 33) и выделим три характерных поведения метода, для более детального анализа - результаты в виде графика 2. Рассуждения такие

же, как и про функции потерь: при малых значениях *step_ alpha* (от 0 до 0.40) точность плавно повышается (без видимых скачков) - так как веса меняются постепенно и плавно (так как значение параметра небольшое); при средних значениях (от 0.45 до 0.85) точность по-началу резко меняется, а потом стабилизируется (так как веса резко, насколько позволяет коэфициент, приближаются к оптимальным значениям за несколько первых итераций и затем плавно приближают функцию потерь к минимуму - можно судить о наличии сходимости метода); при больших значениях *step_ alpha* (более 0.90) метод ведет себя нестабильно , резко меняя точность на каждой итерации (вплоть до последней), так как в силу большого значения весов, метод каждый раз "перепрыгивает"точку оптимума. Так же на графике видно, что при значении $step\_alpha = 0.55$ методу потребовалось совершить около 75 итераций для достижения заданной точности, а то время как, при значениях 0.05 и 0.85 требуется больше итераций для достижения заданной точности.

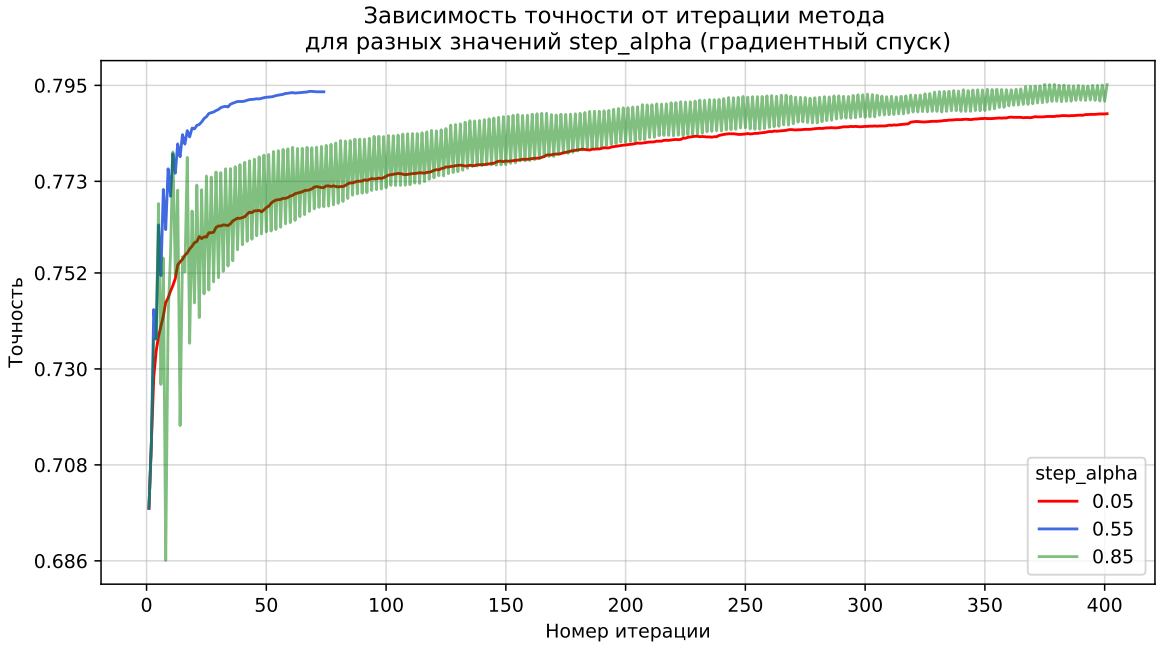


График 2: Зависимость точности от итерации метода для разных значений *step_ alpha*

Так же рассмотрим, как зависит значение функции потерь и точность от реального времени (как изменяются эти величины с течением времени). Рассмотрим графики для тех же значений *step_ alpha*, для которых строили графики выше (графики для всех значений *step_ alpha* зависимости функции потерь от времени можно посмотреть на 34, а зависимости точности от времени можно посмотреть на 35): график зависимости функции потерь от реального времени 3, график зависимости точности от реального времени 4. Видно, что графики похожи на соответствующие им графики зависимости функции потерь и точности от итерации. На этих графиках видно, что для небольших значений *step_ alpha* метод постепенно сходится (с течением времени уменьшаются колебания на обоих графиках для значений $step\_alpha < 0.90$), в то время как для большого значения 0.90 с течением времени колебания хоть и становятся меньше, но не исчезают, что говорит о том, что при данном значении параметра *step_ alpha* метод не сходится. Так же можно справедливо заметить, что чем больше итераций - тем больше времени надо алгоритму.

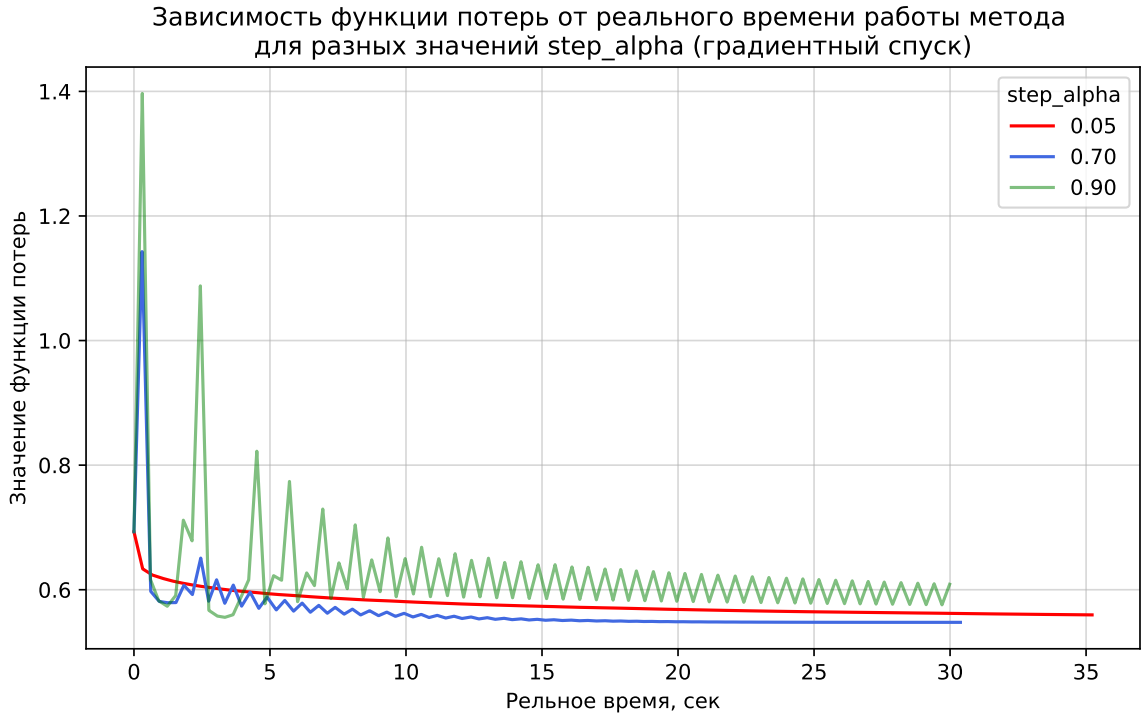


График 3: Зависимость функции потерь от реального времени работы метода для разных значений *step_ alpha*

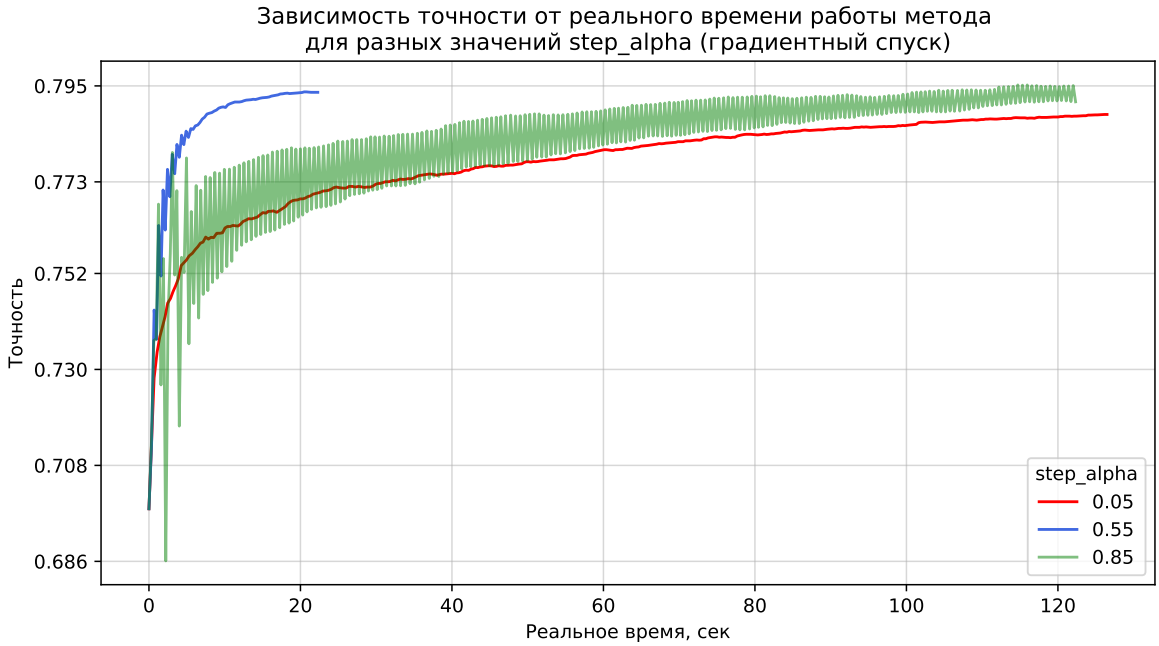


График 4: Зависимость точности от реального времени работы метода для разных значений *step_ alpha*

Рассмотрим зависимость времени работы метода от значения *step_ alpha* 5. Видно, что при малых значениях и при больших время работы больше. Это объясняется тем, что при малых значениях *step_ alpha*, скорость обучения маленькая и необходимо больше итераций алгоритма

для достижения заданной точности (следовательно нужно больше времени), а при больших значениях - алгоритм постоянно перепрыгивает точку минимума, и так же необходимо больше итераций (больше времени). Со средней скоростью обучения время, необходимое алгоритму, уменьшается.

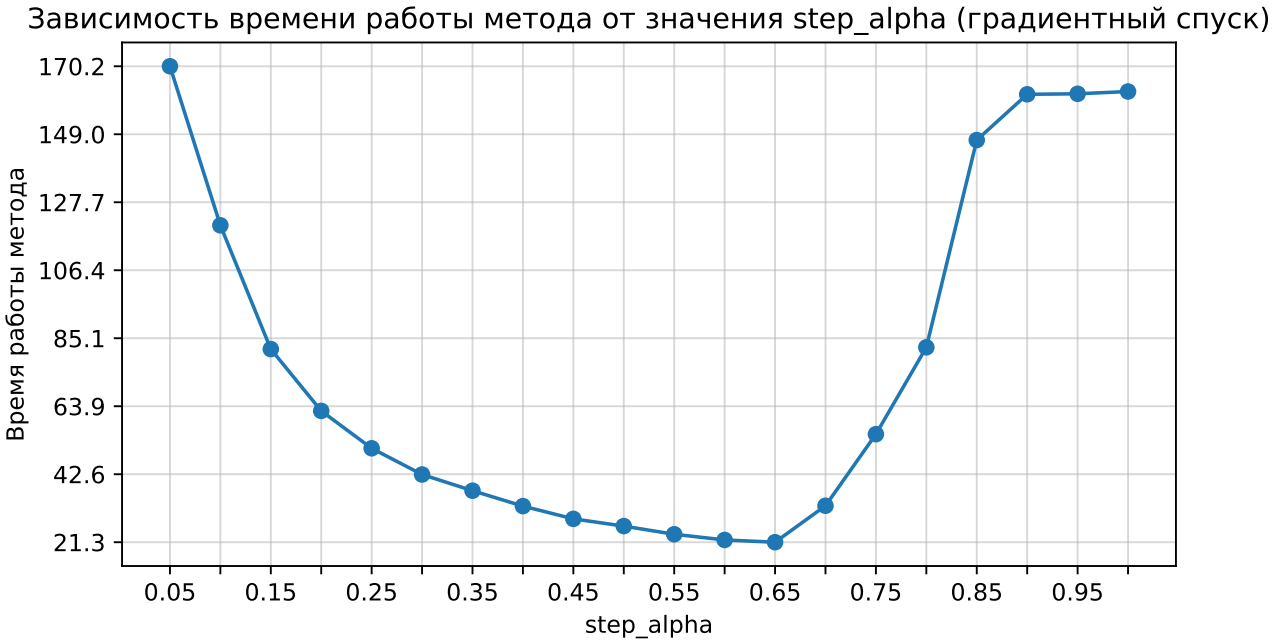


График 5: Зависимость времени работы метода от значения *step_ alpha*

Вывод: наилучшая точность достигается при *step_ alpha* = 0.85 (примерно в окрестности этого значения функция потерь минимальна), она равна 0.79416. Алгоритм наиболее быстро работает при значениях *step_ alpha* = 0.60, 0.65.

3.4.2 Размера шага *step_ beta*

Проанализируем теперь гиперпараметр *step_ beta*.В данном эксперименте рассматривались значения *step_ beta* от 0 до 1 с шагом 0.5. При этом остальные гипераметры были постоянны:

- $step\_alpha = 0.85$
- $tolerance = 10^{-6}$ - разность между функциями потерь, вычисленных на подряд идущих итерациях, при достижении которой алгоритм заканчивается
- $max_iter = 500$ - максимально допустимое число итераций
- $\lambda = 0.1$- коэфициент L_2 регуляризации
- $w^{(0)} = [0, .., 0]$ - начальное приближение

Рассмотрим значения функции потерь, с которыми алгоритм закончился 7. Можно сделать следующие выводы:

- Минимальное значение функции потерь достигается при *step_ beta* = 0.15 и равно 0.54764

- При малом значении $step\_beta = 0.05$ наблюдается значительно большее значение функции потерь (0.65462), что говорит о нестабильности алгоритма при слишком малых значениях параметра

step_ beta	**Функци потерь**
0.05	**0.65462**
0.10	0.54776
0.15	**0.54764**
0.20	0.54765
0.25	0.54765
0.30	0.54766
0.35	0.54766
0.40	0.54767
0.45	0.54768
0.50	0.54770
0.55	0.54773
0.60	0.54778
0.65	0.54785
0.70	0.54797
0.75	0.54822
0.80	0.54869
0.85	0.54935
0.90	0.55021
0.95	0.55127
1.00	0.55254

Таблица 7: Зависимость значения функции потерь от параметра $step\_beta$

- В диапазоне $step\_beta \in [0.15, 0.40]$ значения функции потерь практически не меняются - стабильную работу алгоритма
- При $step\_beta > 0.40$ наблюдается монотонный рост значения функции потерь. Так как с ростом $step\_beta$ скорость обучения все меньше и меньше, и алгоритм все больше и больше не доходит до точки минимума.

Обоснование такому поведение аналогично предыдущему пункту. При малых значений $step\_beta$ скорость обучения будет высокой (так как скорость обучения $\eta_k = \frac{\alpha}{k^\beta}$) и алгоритм будет перепрыгивать значение минимума. При больших же значениях скорость обучения будет маленькая, и алгоритму потребуется больше итераций, чтобы достичь необходимой точности (но на самом деле алгоритм при больших значениях выход из алгоритма будет по причине достижения максимального количества итераций).

Для каждого значения $step\_beta$ рассмотрим зависимость значения функции потерь от номера итерации метода (полный набор графиков можно посмотреть в 36) и выделим три характерных поведения метода, для более детального анализа. Полученный результат подтверждает мысль о том, что при малом значении $step\_beta$ алгоритм не сходится, при слишком больших сходится но медленно (требуется большее количество итераций).

График 6: Зависимость функции потерь от итерации метода для разных значений *step_ beta*

Рассмотрим, как зависит точность от значения *step_ beta*: результаты представленя в Таблице 8.

step_ beta	**accuracy**
0.05	0.77404
0.10	**0.79416**
0.15	0.79353
0.20	0.79343
0.25	0.79353
0.30	0.79358
0.35	0.79334
0.40	0.79329
0.45	0.79309
0.50	0.79275
0.55	0.79227
0.60	0.79208
0.65	0.79164
0.70	0.79116
0.75	0.79005
0.80	0.78835
0.85	0.78603
0.90	0.78400
0.95	0.78182
1.00	0.77902

Таблица 8: Зависимость точности классификации от параметра *step_ beta*

Можно сделать следующие выводы:

- Максимальная точность достигается при $step\_beta = 0.10$ и составляет 0.79416 (79.42%)
- При $step\_beta = 0.05$ наблюдается значительное падение точности до 0.77404 - нестабильное поведение из-за маленького значения параметра (и следовательного большого значения скорости обучения)
- В диапазоне $step\_beta \in [0.10, 0.40]$ точность остается стабильно высокой
- При $step\_beta > 0.40$ наблюдается монотонное снижение точности - так как скорость обучения уменьшается и алгоритм все больше не доходит до точки оптимума

Так же рассмотрим графики зависимости точности от итерации для трех характерных значений $step\_beta$ (для всех значений можно посмотреть тут 37):

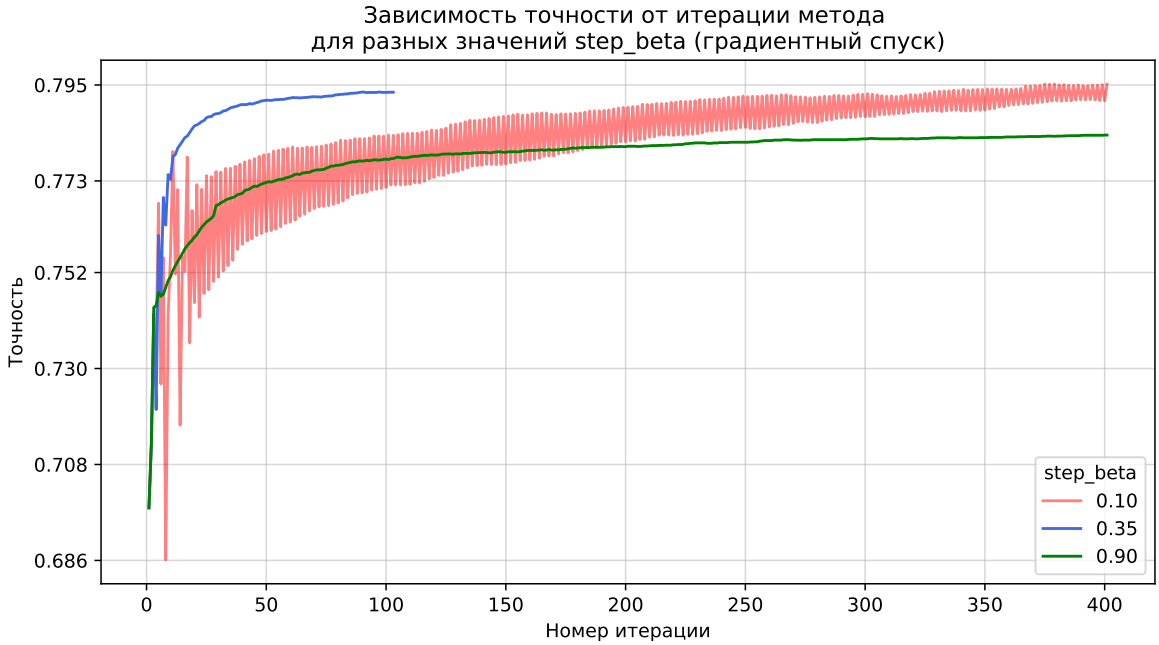


График 7: Зависимость точности от итерации метода для разных значений $step\_beta$

Полученные результаты еще раз подтверждают факт того, что малые значения $step\_beta$ делают скорость обучения большой (и точность на каждой итерации прыгает вверх/вниз), а большие значения делают скорость обучения маленькой и не позволяют алгоритму дойти до оптимума (при этом продвижение плавное, без скачков).

Рассмотрим, как меняеся функция потерь и точность со временем (графики для всех $step\_beta$ можно посмотерть 38 и 39). Для более детального рассмотрения возьмем те же значения $step\_beta$, что и в предыдущих графиках 8 и 9. Полученные зависимости похожи на зависимости функции потерь и точности от времени - наблюдаются такие же скачки/плавные участки. Так же можно увидеть, что при $step\_beta = 0.35$ алгоритм отработал за 30 секунд, в то время как для других значений необходимо больше времени (так как необходимо больше итераций).

График 8: Зависимость функции потерь от реального времени

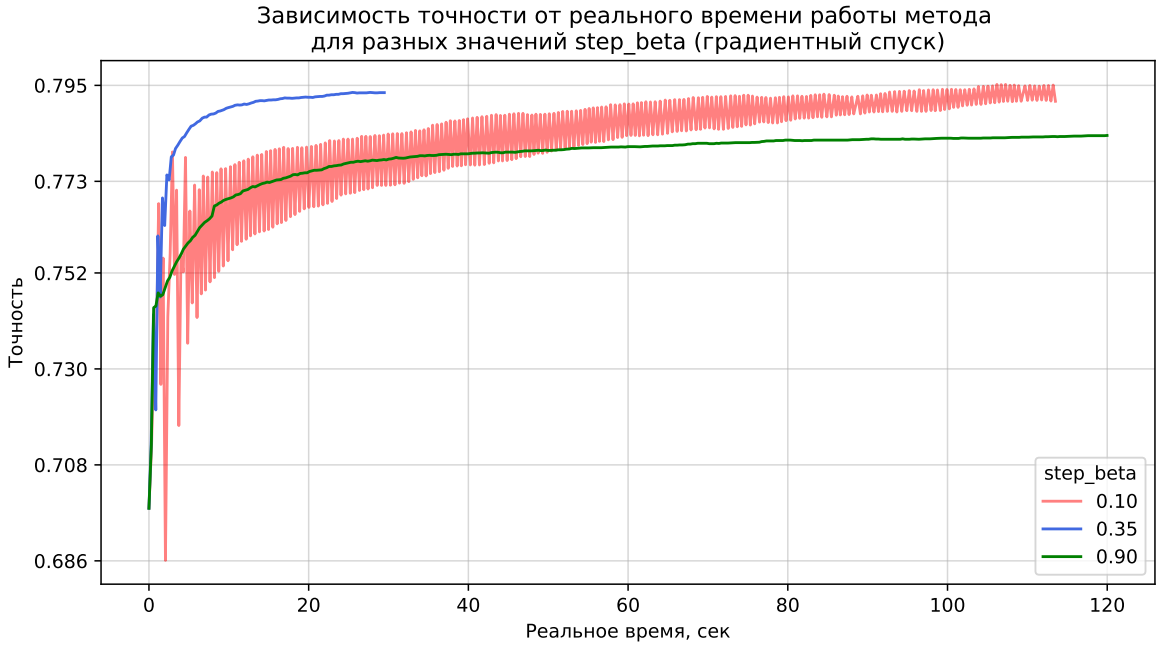


График 9: Зависимость точности от реального времени

Изобразим зависимость времени работы алгоритма от значения *step_ beta*:

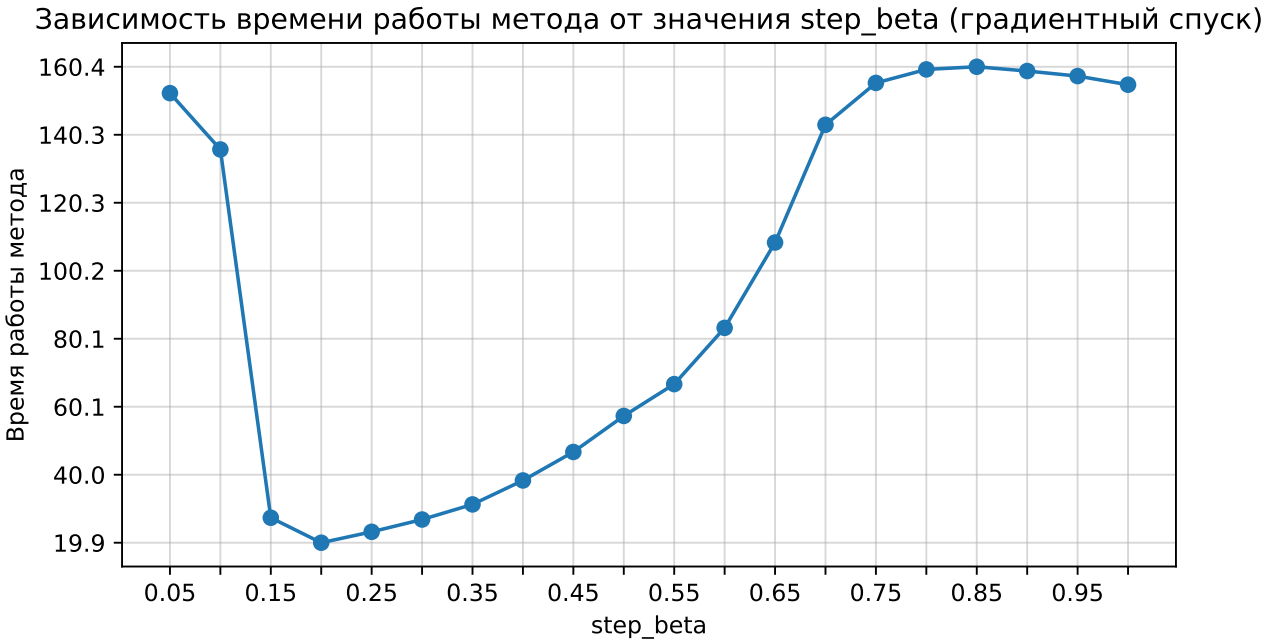


График 10: Зависимость времени работы алгоритма от значения *step_ beta*

Видно, что при малых значениях *step_ beta* алгоритм работает долго, при больших постепенно увеличивает время работы - происходит из большой и маленькой соответственно скоростей обучения.

Вывод: наилучшая точность достигнута при $step\_\ beta = 0.10$, она равна 0.79416. При отходе от этого значения точность падает, функция потерь растет, время работы алгоритма увеличивается.

3.4.3 Начальное приближение $w^{(0)}$

В данном эксперименте рассматривались несколько начальных приближений:

- '0.01s' - все значения в векторе весов равны 0.01
- '0.1s' - все значения в векторе весов равны 0.1
- '0.5s' - все значения в векторе весов равны 0.5
- '-0.5s' - все значения в векторе весов равны -0.5
- '1s' - все значения в векторе весов равны 1
- '-1s' - все значения в векторе весов равны -1
- '2s' - все значения в векторе весов равны 2
- '3s' - все значения в векторе весов равны 3
- '100s' - все значения в векторе весов равны 100
- '-100s' - все значения в векторе весов равны -100

- '0s' - все значения в векторе весов равны 0
- '1/max' - вектор заполняется средними по каждому признакам значением (среднему по всей выборке)
- 'max' - вектор заполняется максимальным по каждому признакам значением (максимальным по всей выборке)
- '-max' - вектор заполняется -максимальным по каждому признакам максимальным (среднему по всей выборке)

При этом остальные гипераметры были постоянны:

- $step\_alpha = 0.85$
- $step\_alpha = 0.1$
- $tolerance = 10^{-6}$ - разность между функциями потерь, вычисленных на подряд идущих итерациях, при достижении которой алгоритм заканчивается
- $max_iter = 500$ - максимально допустимое число итераций
- $\lambda = 0.1$- коэфициент L_2 регуляризации

Сравним значеня функции потерь:

Инициализация	**Значение функции потерь**
0.01s	0.54767
0.1s	0.54767
0.5s	**0.54760**
-0.5s	0.54767
1s	**0.54760**
-1s	**0.54760**
2s	0.**54760**
3s	0.54767
100s	0.54767
-100s	0.54767
0s	**0.54760**
1/max	0.54767
max	0.54767
-max	**0.54760**

Таблица 9: Зависимость функции потерь от способа инициализации весов

Можно сделать слудующие выводы:

- Наблюдаются только два различных значения функции потерь:
  - 0.54767
  - 0.54760
- Значение 0.54760 достигается при следующих инициализациях: $w_0 = 0.5, 1, -1, 2, 0, -\max$

- Все остальные инициализации дают значение 0.54767
- Выбор начального приближения практически не влияет на конечный результат. Алгоритм демонстрирует высокую устойчивость к выбору начальных весов
- Статистические методы инициализации не дают преимущества перед простыми константными значениями

Для каждого значения w_0 рассмотрим зависимость значения функции потерь от номера итерации метода (полный набор графиков можно посмотреть в 40) и выделим три характерных поведения метода, для более детального анализа:

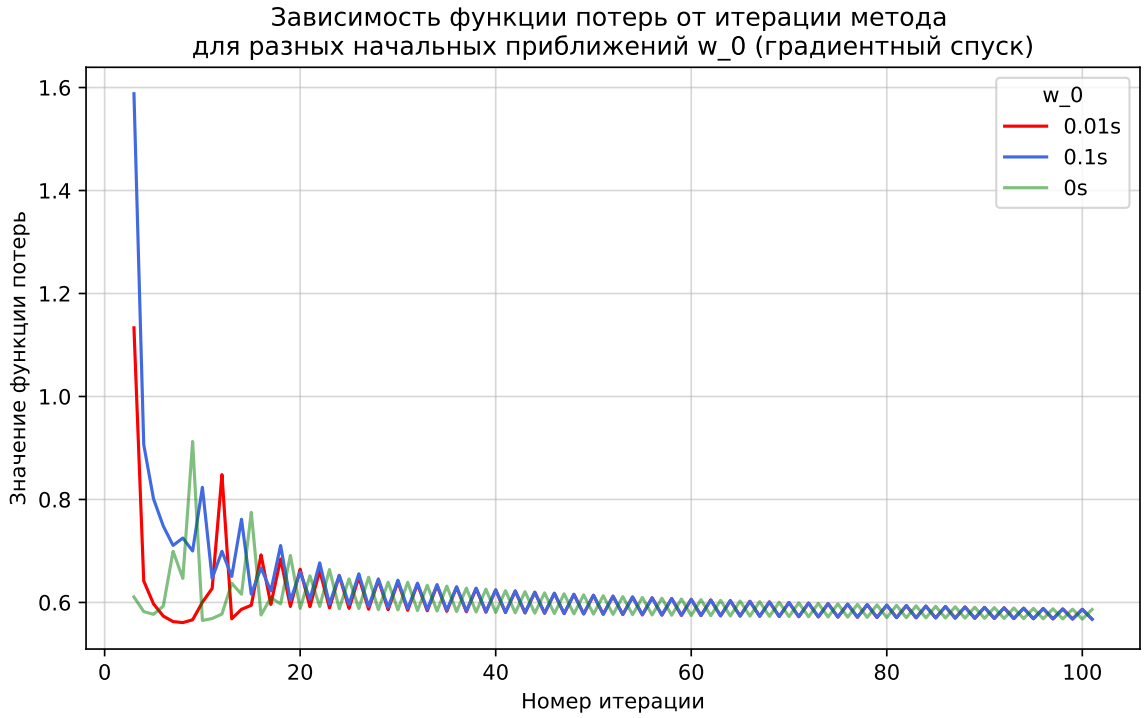


График 11: Зависимость функции потерь от итерации метода для разных значений w_0

По полученным результатам можно сделать вывод, что все три варианта демонстрируют колебательное поведение в первые 20 итераций, а после 40-й итерации колебания значительно уменьшаются, к 100-й итерации все три варианта сходятся примерно к одному значению (около 0.6). Наиболее стабильное поведение показывает вариант с $w_0 = 0s$ (так как два остальных имеют более сильные колебания в начале)

Рассмотрим зависимость точности от начального приближения 10. Можно сделать вывод, что все значения точности находятся в очень узком диапазоне: [0.794109, 0.794157], нет явной зависимости точности от абсолютного значения весов, знак начальных весов (положительный или отрицательный) не оказывает существенного влияния на конечную точность, экстремальные значения (100s, -100s, max, -max) не приводят к значительному ухудшению результатов. Алгоритм демонстрирует высокую устойчивость к выбору начальных весов. Но в то же время оптимальными можно считать значения (0.5s, 1s, -1s, 2s), так как они дают максимальную точность.

Построим график зависимости точности от итерации для трех характерных поведения (все графики для всех начальных приближений можно посмотреть 41):

Начальные веса	Точность
0.01s	0.794109
0.1s	0.794109
0.5s	0.794157
-0.5s	0.794109
1s	0.794157
-1s	0.794157
2s	0.794157
3s	0.794109
100s	0.794109
-100s	0.794109
0s	0.794157
1/max	0.794109
max	0.794109
-max	0.794157

Таблица 10: Сравнение точности для различных начальных весов

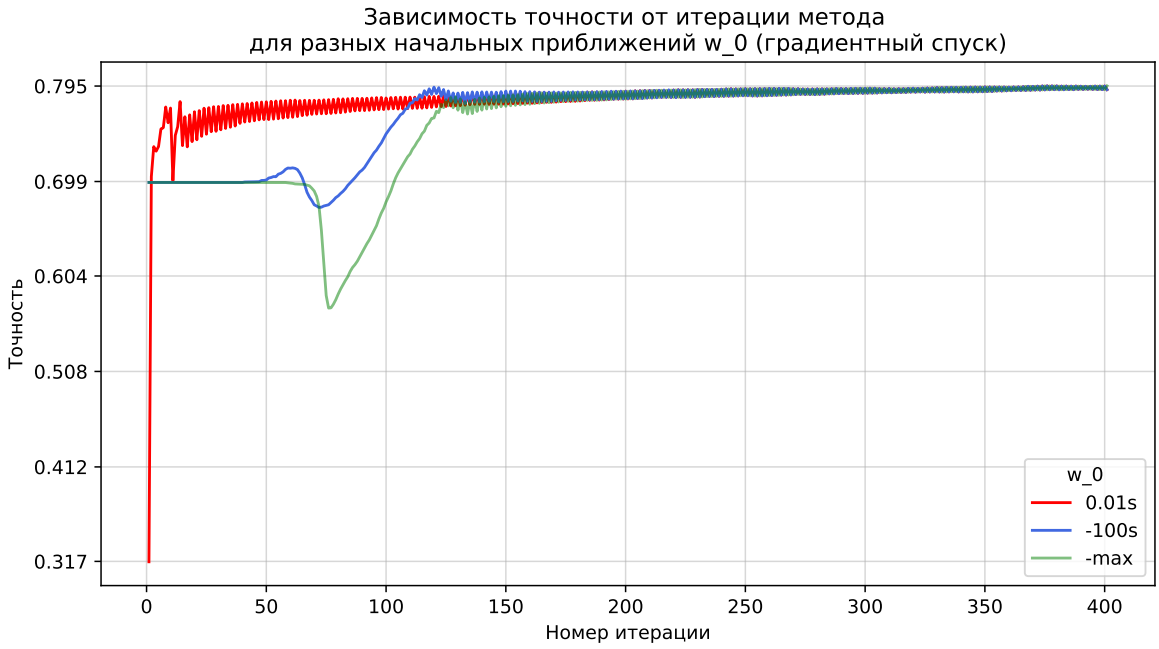


График 12: Зависимость точности от итерации метода для разных значений w_0

Видно как веса влияют на начальную точность (0.698 - точность, если все предсказания отрицательны, 0.3019 - если все положительны). То есть при малых весах метод считает все текста не токсичными, при больших - все токсичными. Однако спустя время все значения начинают сходится.

Так же можно рассмотреть зависимость точности и функции потерь от реального времени (смотри графики 38 и 39). Детально рассмотрим те же самые значения начальных приближений (13 и 14). Видно, что графики почти что идентичны с соответствующими им графиками зависимости от итерации (так как время одной итерации почти одинаковое).

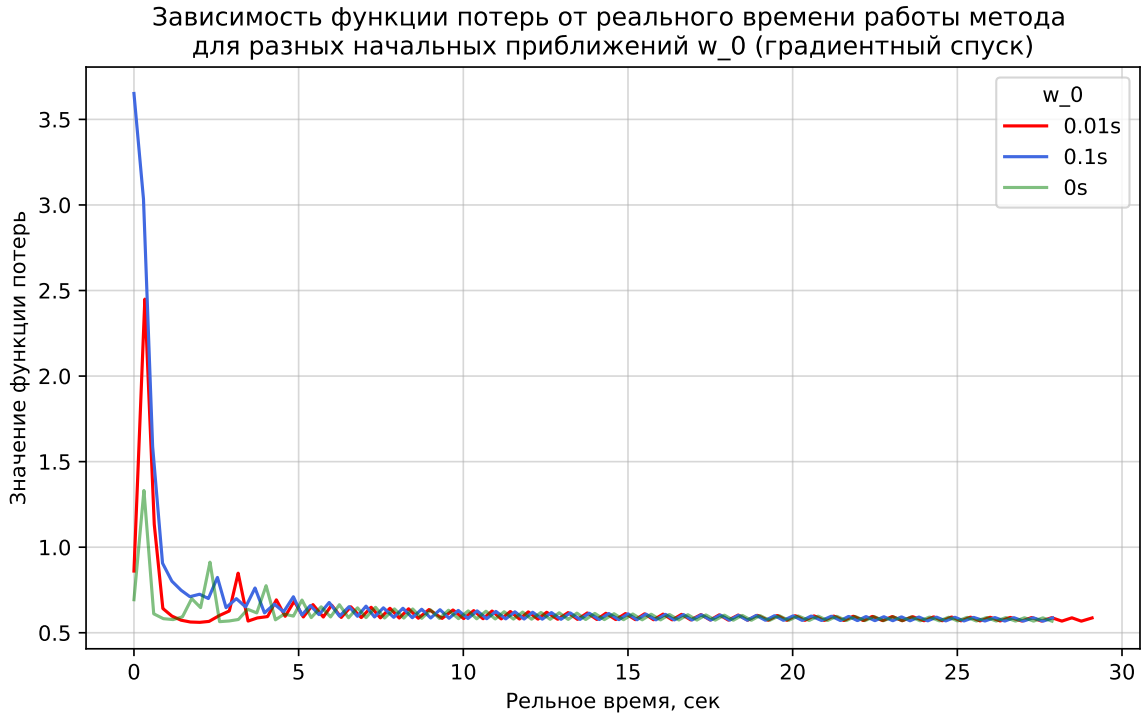


График 13: Зависимость точности от реального времени для разных значений w_0

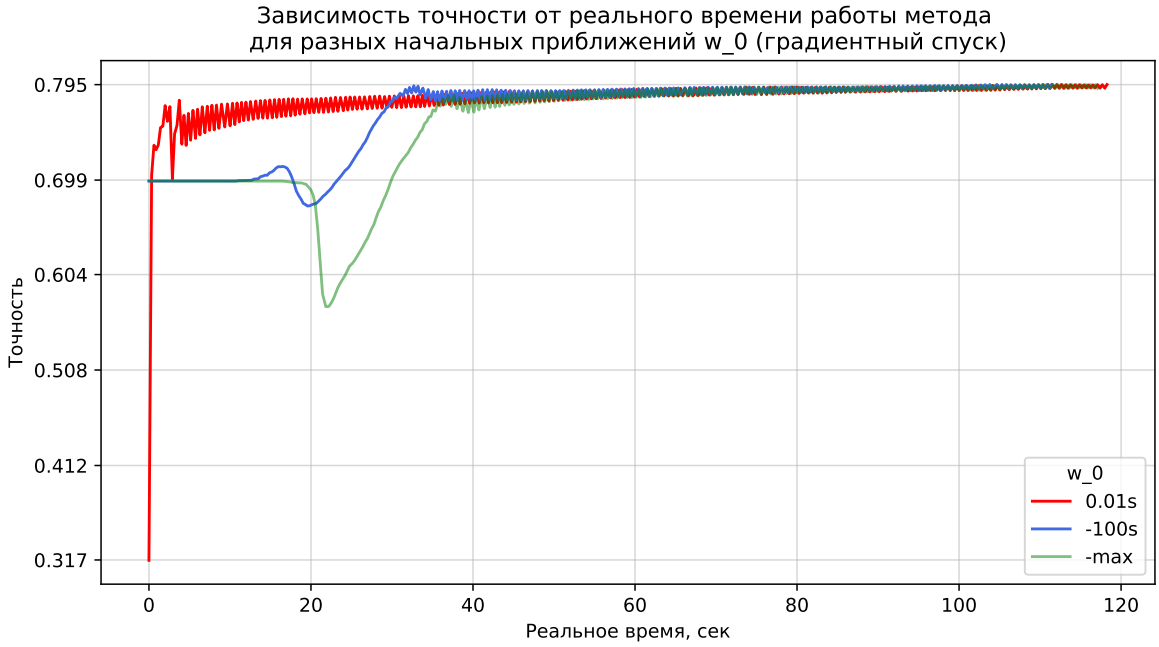


График 14: Зависимость точности от реального времени для разных значений w_0

Рассмотрим, сколько работает алгоритм для каждых начальных весов. Время почти одинаково.

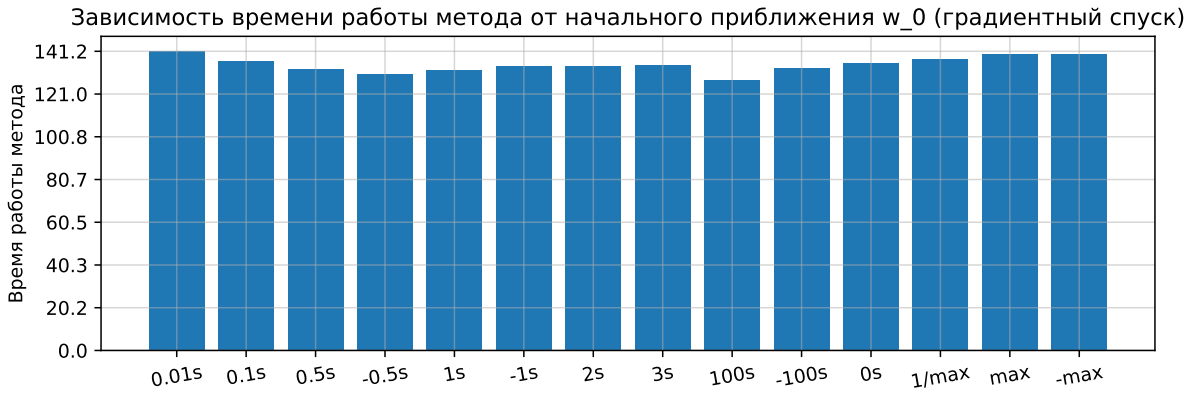


График 15: Время работы алгоритма для разных w_0

Вывод: выбор весов почти не влияет ни на точность, ни на функцию потерь, ни на время работы, поэтому можно выбирать любые начальные веса.

3.5 Стохастический градиентный спуск

В данном эксперименте будет проведен анализ градиентного спуска, а именно:

- Влияние параметра размера шага $step\_alpha$ на точность предсказаний, функцию потерь, время работы метода
- Влияние параметра размера шага $step\_beta$ на точность предсказаний, функцию потерь, время работы метода
- Влияние начального приближения $w^{(0)}$ на точность предсказаний, функцию потерь, время работы метода
- Влияние размера $batch$ на точность предсказаний, функцию потерь, время работы метода

То есть в данном эксперименте будут подобраны оптимальные гиперпараметры для стохастического градиентного спуска.

3.5.1 Размера шага *step_ alpha*

В данном эксперименте рассматривались значения $step\_alpha$ от 0 до 1 с шагом 0.05. При этом остальные гиперпараметры были постоянны:

- $step\_beta = 0.1$
- $tolerance = 10^{-6}$ - разность между функциями потерь, вычисленных на подряд идущих итерациях, при достижении которой алгоритм заканчивается
- $max\_iter = 500$ - максимально допустимое число итераций
- $\lambda = 0.1$ - коэффициент L_2 регуляризации
- $batch\_size = 10000$ - размер батча
- $w^{(0)} = [0, ..., 0]$ - начальное приближение

Рассмотрим значения функции потерь по исходу работы алгоритма:

step_ alpha	**Функция потерь**	***step_ alpha***	**Функция потерь**
0.05	0.54861	0.55	0.54789
0.10	0.55014	0.60	0.54786
0.15	0.54785	0.65	0.54777
0.20	0.54783	0.70	0.54807
0.25	0.54777	0.75	0.55777
0.30	0.54780	0.80	0.54846
0.35	0.54797	0.85	0.54884
0.40	**0.54775**	0.90	0.55145
0.45	0.54786	0.95	0.56529
0.50	0.54780	1.00	0.54962

Таблица 11: Зависимость значения функции потерь от параметра *step_ alpha*

Можно сделать вывод:

- Минимальное значение функции потерь достигается при $step\_alpha = 0.40$ и равно 0.54775
- При $step\_alpha < 0.40$ наблюдаются колебания значений функции потерь
- При $step\_alpha > 0.75$ наблюдается заметный рост значений функции потерь
- В диапазоне $step\_alpha \in [0.15, 0.70]$ значения функции потерь остаются стабильно низкими

Для каждого значения *step_ alpha* рассмотрим зависимость значения функции потерь от номера эпохи метода (полный набор графиков можно посмотреть в 44) и выделим три характерных поведения метода, для более детального анализа:

График 16: Зависимость функции потерь от итерации метода для разных значений *step_ alpha*

При $step\_alpha = 0.05$ наблюдается медленная, но стабильная сходимость, при $step\_alpha = 0.25$ достигается быстрая сходимость с небольшими колебаниями, при $step\_alpha = 0.90$ происходят значительные колебания на протяжении всего процесса обучения. Это происходит потому, что при малых значениях альфа скорость обучения маленькая (и медленное продвижение), при больших - происходит перескакивание через минимум.

Проанализируем зависимость точности от номера эпохи:

step_ alpha	**Точность**	***step_ alpha***	**Точность**
0.05	0.78840	0.55	0.79425
0.10	0.78463	0.60	0.79382
0.15	0.79213	0.65	0.79290
0.20	0.79145	0.70	0.79222
0.25	0.79319	0.75	0.79106
0.30	0.79425	0.80	0.79363
0.35	**0.79440**	0.85	0.79246
0.40	0.79387	0.90	0.78971
0.45	0.79135	0.95	0.78980
0.50	0.79425	1.00	0.78681

Таблица 12: Зависимость точности от параметра $step\_alpha$

Можно сделать вывод, что:

- Максимальная точность достигается при $step\_alpha = 0.35$ и составляет 0.79440
- При малых значениях $step\_alpha$ ($step\_alpha < 0.15$) точность ниже оптимальной
- В диапазоне $step\_alpha \in [0.15, 0.65]$ точность стабильно высокая (>0.792)
- При $step\_alpha > 0.85$ наблюдается заметное падение точности

Для каждого значения $step\_alpha$ рассмотрим зависимость значения точности от номера эпохи метода (полный набор графиков можно посмотреть в 45):

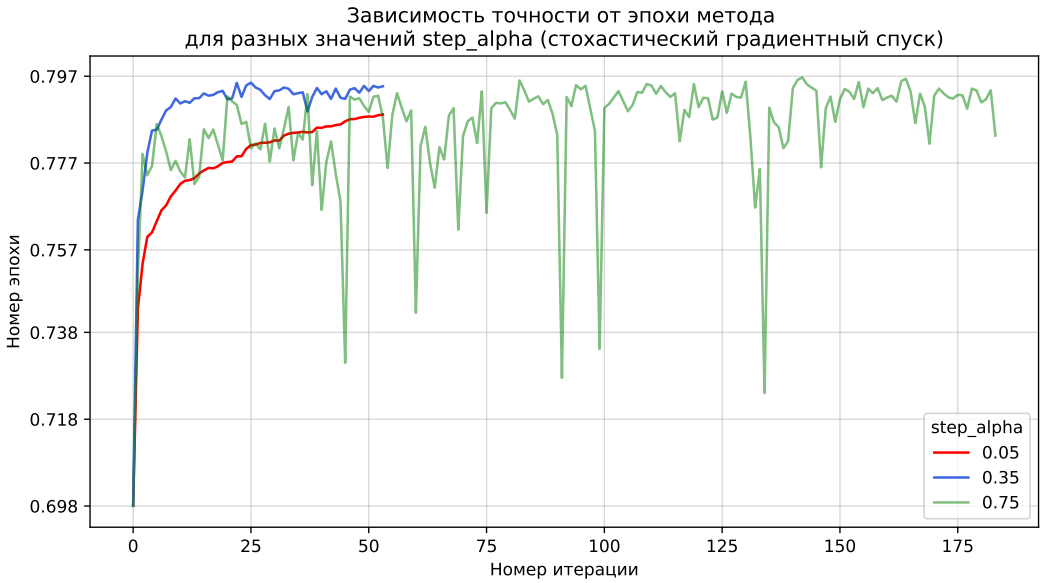


График 17: Зависимость точности от эпохи метода для разных значений $step\_alpha$

При малом значении *step_alpha* (0.05) наблюдается плавный рост точности, при среднем значении (0.35) достигается быстрый рост и стабилизация точности, при большом значении (0.75) наблюдаются заметные колебания точности (это происходит опять же из-за скорости обучения, которая определяется *step_ alpha*).

Рассмотрим, как зависит функция потерь и точность от реального времени (все графики 46 47):

График 18: Зависимость функции потерь от времени для разных значений *step_ alpha*

График 19: Зависимость точности от времени для разных значений *step_ alpha*

Графики почти что не отличаются (так как в среднем эпоха происходит за одинаковое время).

Общее время работы метода:

График 20: Зависимость времени работы от значений $step\_alpha$

Наименьшее время работы достигается в диапазоне $step\_alpha \in [0.30, 0.60]$, при крайних значениях $step\_alpha$ время работы существенно возрастает, так же наблюдается резкий рост времени работы при $step\_alpha > 0.85$

Вывод: лучшая точность 0.79440 при $step\_alpha = 0.35$

3.5.2 Размера шага *step_ beta*

В данном эксперименте рассматривались значения $step\_beta$ от 0 до 1 с шагом 0.05 при следующих фиксированных параметрах:

- $step\_alpha = 0.5$
- $tolerance = 10^{-6}$ - критерий остановки по разности функций потерь
- $max\_iter = 500$ - максимальное число итераций
- $\lambda = 0.1$ - коэффициент L_2 регуляризации
- $batch\_size = 10000$ - размер батча
- $w^{(0)} = [0, ..., 0]$ - начальное приближение

Принципы выведенные при анализе градиентного спуска, по которым при определенных значениях точность, значение функции потерь, время работы больше или меньше, остаются теми же (если скорость обучения маленькая, то выполняется долго, функция потерь и точность будут плавно меняться, но не достигнут лучших результатов), если скорость обучения большая то будут скачки около оптимального значения (на протяжении всей работы алгоритма).

step_beta	Функция потерь
0.05	0.54801
0.10	0.54780
0.15	0.54853
0.20	0.54797
0.25	**0.54769**
0.30	0.54774
0.35	0.54780
0.40	0.54777
0.45	0.54776
0.50	0.54801

График 21: Значения функции потерь

step_beta	Точность
0.05	**0.79450**
0.10	0.79425
0.15	0.79377
0.20	0.79324
0.25	0.79358
0.30	0.79319
0.35	0.79256
0.40	0.79382
0.45	0.79237
0.50	0.79106

График 22: Значения точности

Можно сделать выводы:

- Минимальное значение функции потерь (0.54769) достигается при $step\_beta = 0.25$
- Максимальная точность (0.79450) достигается при $step\_beta = 0.05$
- В диапазоне $step\_beta \in [0.05, 0.40]$ точность остается стабильно высокой (>0.792)
- При $step\_beta > 0.50$ наблюдается постепенное ухудшение обоих метрик

Ниже приведены графики (все графики можно посмотреть beta), по которым произведем анализ параметра бета:

1. **Влияние на функцию потерь:**
   - При малом значении $step\_beta = 0.05$ наблюдаются значительные колебания функции потерь как по эпохам, так и по времени
   - При среднем значении $step\_beta = 0.35$ достигается наиболее стабильное поведение с плавным убыванием
   - При большом значении $step\_beta = 0.75$ функция потерь убывает медленнее, но более монотонно
2. **Влияние на точность:**
   - Все значения демонстрируют быстрый начальный рост точности в первые 5 эпох
   - $step\_beta = 0.05$ показывает наибольшие колебания, но достигает лучшей точности
   - $step\_beta = 0.75$ приводит к более медленному, но стабильному росту точности
3. **Временные характеристики:**
   - При $step\_beta = 0.05$ время работы значительно больше (около 150 секунд)
   - Оптимальное время работы достигается при средних значениях $step\_beta$ (около 20-30 секунд)
   - График времени работы показывает резкое увеличение при малых значениях параметра

График 23: Зависимость функции потерь от эпохи

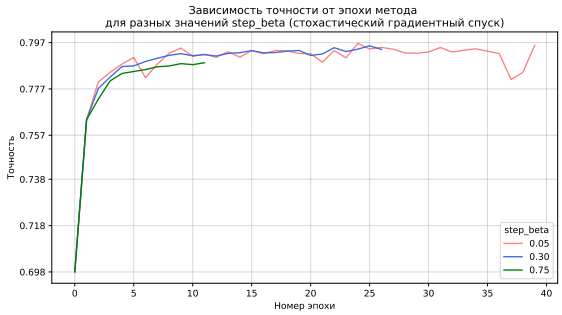


График 24: Зависимость точности от эпохи

График 25: Зависимость функции потерь от времени

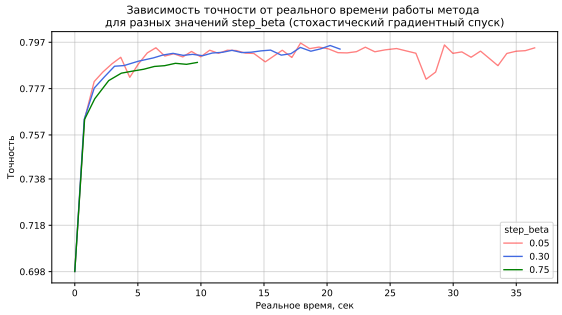


График 26: Зависимость точности от времени

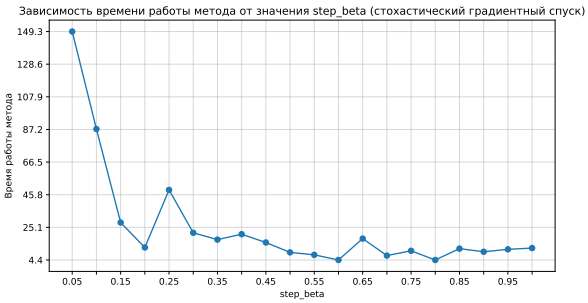


График 27: Зависимость времени работы от значения step_beta

Итого: Малые значения *step_beta* (0.05) дают лучшую точность, но требуют значительно большего времени работы. Средние значения (0.30-0.40) обеспечивают оптимальный баланс между точностью и временем работы. Большие значения (0.75) приводят к более стабильному, но медленному обучению

Вывод: лучшая точность 0.79450 при $step\_beta = 0.05$

3.5.3 Начальное приближение $w^{(0)}$

Параметры:

- $batch\_size = 10000$ - размер батча
- $step\_alpha = 0.5$ - параметр скорости обучения

- $step\_beta = 0.2$ - параметр скорости затухания
- $tolerance = 10^{-6}$ - критерий остановки по разности функций потерь
- $max\_iter = 500$ - максимальное число итераций
- $l2\_coef = 0.1$ - коэффициент L_2 регуляризации

Начальные веса	**Функция потерь**	**Точность**
0.01s	0.547819	0.793867
0.1s	0.547788	0.793384
0.5s	0.547821	0.793867
-0.5s	0.547820	0.793867
1s	0.547826	0.793916
-1s	0.547826	0.793867
2s	0.547848	**0.793964**
3s	0.547788	0.793384
100s	0.547744	0.793142
-100s	0.547744	0.793432
0s	**0.547969**	0.793239
1/max	0.547889	0.791207
max	0.547743	0.793384
-max	0.547805	0.793335

Таблица 13: Влияние начальных весов на функцию потерь и точность

Так же как и с градиентным спуском начальные веса сильно не влияют на точность и значение функции потерь (можно посмотреть графики F)

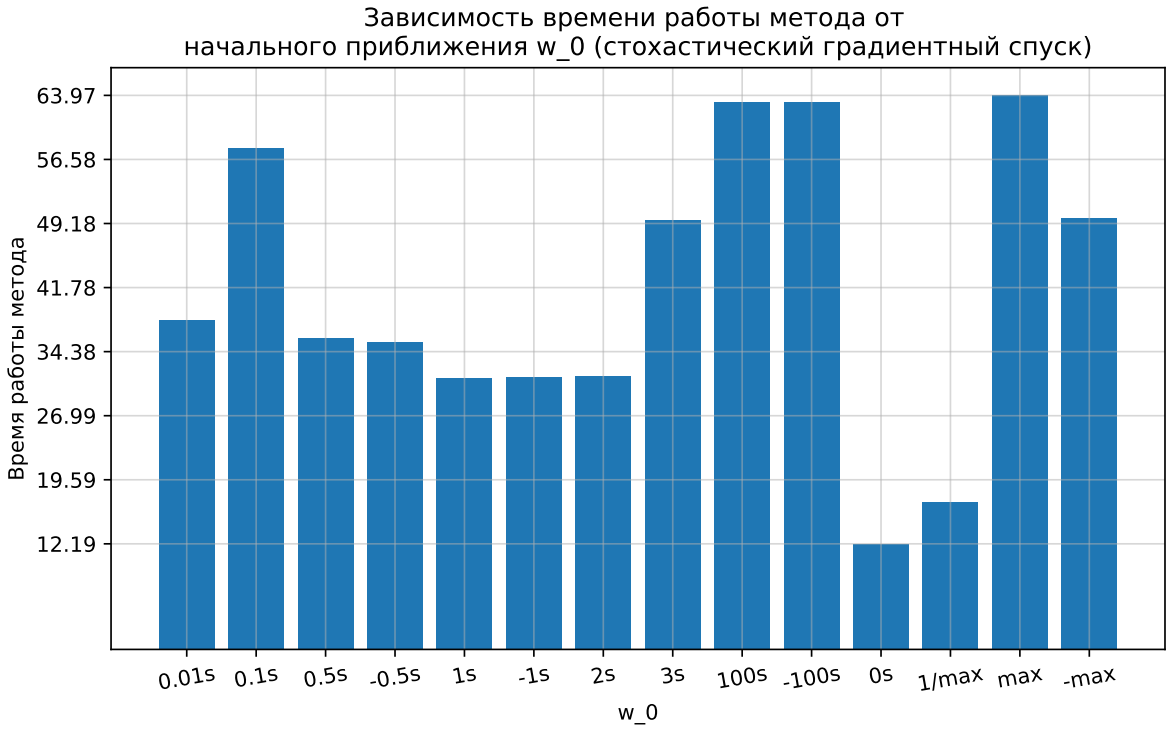


График 28: Зависимость времени работы метода от начального приближения w_0

По данной диаграмме моджно сказать, что наименьшее время работы (около 12 секунд) достигается при нулевой инициализации (0s), наибольшее время работы (около 64 секунды) наблюдается для экстремальных значений (100s, -100s, max). В силу того, что начальные веса на точность не влияют, но на 0s работает быстрее, будем работать с этим вектором весов.

3.5.4 Размер батча *batch_ size*

Параметры эксперимента:

- Функция потерь: бинарная логистическая регрессия
- $step\_alpha = 0.5$ - параметр скорости обучения
- $step\_beta = 0.5$ - параметр скорости затухания
- $tolerance = 10^{-6}$ - критерий остановки
- $max\_iter = 500$ - максимальное число итераций
- $l2\_coef = 0.1$ - коэффициент L_2 регуляризации
- $w_0 = [0, ..., 0]$ - начальное приближение

Размер батча	**Функция потерь**	**Точность**
1000	0.550065	0.789611
2000	0.548559	0.793722
3000	**0.547783**	**0.793867**
5000	0.547785	0.792465
7000	0.549099	0.786854
10000	0.548010	0.791062
20000	0.547895	0.791207
30000	0.547956	0.790917
40000	0.549470	0.786080
50000	0.548525	0.789660

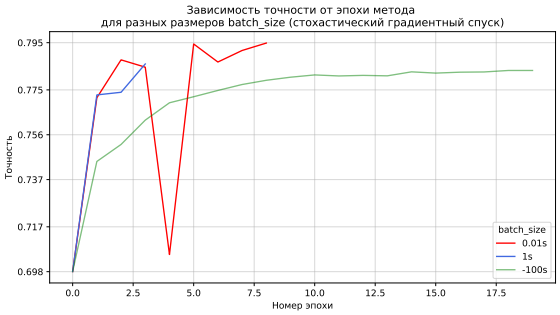
Таблица 14: Влияние размера батча на функцию потерь и точность

Было протестировано десять различных размеров батча: от 1000 до 50000 элементов. Рассмотрим влияние размера батча на значение функции потерь. Минимальное значение (0.547783) достигается при размере батча 3000. При этом наблюдается тенденция к увеличению значения функции потерь как при очень малых, так и при очень больших размерах батча. Так как диапазон изменения функции потерь относительно небольшой от 0.547783 до 0.550065, то метод стабильный.

Максимальная точность (0.793867) также достигается при размере батча 3000. Наименьшая точность (0.786080) наблюдается при размере батча 40000. Так же при размере от 2000 до 5000 точность стабильно высокая. При увеличении размера батча более 7000 наблюдается устойчивая тенденция к снижению точности. Можно сделать слудеющие выводы:

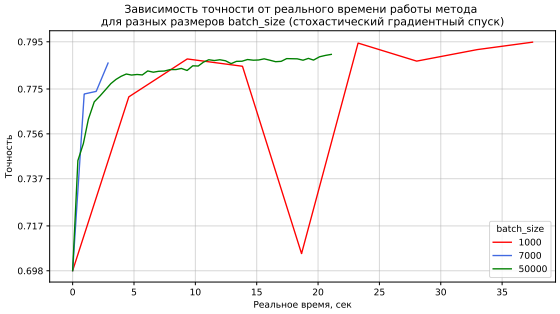
- оптимальный диапазон размера батча (2000-5000), в котором достигаются наилучшие результаты как по функции потерь, так и по точности

- маленький и большие размер батча приводит к увеличению функции потерь и снижению точности
- оптимальным выбором является размер батча 3000



(a) Зависимость точности от эпохи

(b) Зависимость функции потерь от эпохи



(c) Зависимость точности от времени

(d) Зависимость функции потерь от времени

(e) Зависимость времени работы от размера батча

График 29: Анализ влияния размера батча на характеристики обучения

Быстрый начальный рост точности характерен для всех размеров батча. При малом размере батча имеются большие колебания по точности и по значению функции потерь, в том время как средние и большие батчи стабильно растут (месатми не очень плавно, но явных скачков нет). По времени малые батчи выполняются дольше всего (так как им необходимо совершить большее число итераций в эпохе. При этом при сильном увеличении размера батча время тоже увеличивается (так как реже проверяется на достижение необходимой точности).

Вывод: оптимальный размер батча 3000

3.6 Сравнительный анализ градиентного и стохастического градиентного спуска

Будем сравнивать эти два метода по результатам, полученных из предыдущих экспериментов (по гиперараметрам, по времени выполнения).

1. Влияние параметра step_alpha:
Градиентный спуск:

- Оптимальное значение step_alpha = 0.85 с точностью 0.79416
- В диапазоне [0.35, 0.65] точность стабилизируется на 0.79353
- При step_alpha > 0.85 резкое падение точности до 0.77118 (при step_alpha = 1.0)
- Минимальное значение функции потерь 0.54764 при step_alpha = 0.70, 0.75

Стохастический градиентный спуск:

- Оптимальное значение step_alpha = 0.35 с точностью 0.79440
- В диапазоне [0.15, 0.65] точность стабильно высокая (>0.792)
- При step_alpha > 0.85 падение точности менее резкое - до 0.78681 (при step_alpha = 1.0)
- Минимальное значение функции потерь 0.54775 при step_alpha = 0.40

2. Влияние параметра step_beta:
Градиентный спуск:

- Оптимальное значение step_beta = 0.15 с минимальной функцией потерь 0.54764
- При значении 0.05 значительно хуже результат (функция потерь 0.65462)
- В диапазоне [0.15, 0.40] значения функции потерь стабильны
- При step_beta > 0.40 монотонный рост значения функции потерь

Стохастический градиентный спуск:

- Лучшая точность 0.79450 при step_beta = 0.05
- Функция потерь минимальна (0.54769) при step_beta = 0.25
- В диапазоне [0.05, 0.40] точность стабильно высокая (>0.792)
- При step_beta > 0.50 постепенное ухудшение обеих метрик

3. Влияние начальных весов w_0:
Градиентный спуск:

- Диапазон точности: [0.794109, 0.794157]
- Оптимальные значения: 0.5s, 1s, -1s, 2s (точность **0.794157**)
- Время работы почти одинаковое для всех инициализаций

- Нулевая инициализация дает точность 0.794157

Стохастический градиентный спуск:

- Диапазон точности: [0.791207, 0.793964]
- Лучший результат при w_0 = 2s (точность 0.793964)
- Наименьшее время работы при нулевой инициализации (около 12 секунд)
- Экстремальные значения (±100s, max) требуют около 64 секунд работы

4. Временные характеристики:
Градиентный спуск:

- Оптимальное время при step_alpha = 0.60, 0.65
- U-образная зависимость времени от step_alpha
- При малых и больших значениях step_alpha время существенно возрастает
- Время работы слабо зависит от начальной инициализации

Стохастический градиентный спуск:

- Сильная зависимость от размера батча
- Оптимальное время при batch_size = 7000 (около 4 секунд)
- Малые батчи (1000-3000) требуют 40-50 секунд
- При batch_size $>$ 30000 время снова возрастает

5. Особенности сходимости:
Градиентный спуск:

- Плавное изменение функции потерь при малых step_alpha
- Более резкие изменения в начале при средних значениях
- Сильные колебания при больших step_alpha
- Стабильная сходимость при оптимальных параметрах

Стохастический градиентный спуск:

- Более выраженные колебания на всех этапах
- Быстрый начальный рост точности
- Зависимость стабильности от размера батча
- При малых батчах высокая волатильность метрик

Общие выводы:

1. **Точность:** SGD показывает незначительно лучшие результаты (0.79450 против 0.79416), но требует более тщательной настройки параметров.

2. **Стабильность:** GD демонстрирует более стабильное поведение и меньшую чувствительность к гиперпараметрам.

3. **Время работы:** SGD при оптимальных параметрах работает быстрее, но имеет больший разброс времени работы в зависимости от параметров.

4. **Практическое применение:** Выбор метода зависит от приоритетов:

- Для максимальной точности: SGD с тщательной настройкой параметров
- Для стабильности: GD с параметрами из середины допустимого диапазона
- Для быстрой работы: SGD с оптимальным размером батча

3.7 Лемматизация, удаление стоп-слов

Исходный размер признакового пространства (при использовании min_df = 500) был 586. После лемматизации - 577. После удаления стоп-слов - 474. После лемматизации и удаления стоп слов - 470. То есть эти методы сокращают признаковое пространство.

Метод	$\alpha + \beta$	Accuracy	Time (s)
GD обычный	0.1 + 0.05	0.79280	89.326
	0.1 + 0.2	0.79159	149.650
	0.35 + 0.1	0.79353	34.112
	0.4 + 0.2	0.79329	42.914
	0.5 + 0.05	0.79358	21.556
	0.85 + 0.1	0.79416	140.569
SGD обычный	0.1 + 0.05	0.79358	47.675
	0.1 + 0.2	0.79082	21.577
	0.35 + 0.1	0.79537	221.896
	0.4 + 0.2	0.79343	127.129
	0.5 + 0.05	0.77994	80.248
	0.85 + 0.1	0.78831	481.012

Таблица 15: Результаты экспериментов с различными методами и параметрами оптимизации

Метод	$\alpha + \beta$	Accuracy	Time (s)
GD lem	0.1 + 0.05	0.79716	101.963
	0.1 + 0.2	0.79619	157.130
	0.35 + 0.1	0.79764	34.868
	0.4 + 0.2	0.79769	43.219
	0.5 + 0.05	0.79725	154.747
	0.85 + 0.1	0.76141	153.476
SGD lem	0.1 + 0.05	0.79662	17.495
	0.1 + 0.2	0.79421	21.445
	0.35 + 0.1	0.79691	64.319
	0.4 + 0.2	0.79672	103.321
	0.5 + 0.05	0.79798	393.592
	0.85 + 0.1	0.77999	1288.685
GD removed	0.1 + 0.05	0.81225	80.035
	0.1 + 0.2	0.81109	135.677
	0.35 + 0.1	0.81249	29.836
	0.4 + 0.2	0.81249	34.177
	0.5 + 0.05	0.81244	16.601
	0.85 + 0.1	0.81244	11.570
SGD removed	0.1 + 0.05	0.81128	15.615
	0.1 + 0.2	0.81026	15.853
	0.35 + 0.1	0.81171	7.431
	0.4 + 0.2	0.81350	15.120
	0.5 + 0.05	0.81312	10.007
	0.85 + 0.1	0.81350	935.191
GD lem+removed	0.1 + 0.05	0.81491	75.533
	0.1 + 0.2	0.81399	121.548
	0.35 + 0.1	0.81486	26.834
	0.4 + 0.2	0.81481	33.761
	0.5 + 0.05	0.81505	16.696
	0.85 + 0.1	0.81520	11.907
SGD lem+removed	0.1 + 0.05	0.81375	21.063
	0.1 + 0.2	0.81326	21.251
	0.35 + 0.1	0.81529	9.846
	0.4 + 0.2	0.81534	10.112
	0.5 + 0.05	0.81534	34.160
	0.85 + 0.1	0.81607	107.977

Таблица 16: Результаты экспериментов с различными методами и параметрами оптимизации

Время работы:
GD обычный:

- Минимальное время - 21.556 секунд при alpha=0.5, beta=0.05
- Максимальное время - 149.650 секунд при alpha=0.1, beta=0.2

SGD обычный:

- Минимальное время - 21.577 секунд при alpha=0.1, beta=0.2
- Максимальное время - 481.012 секунд при alpha=0.85, beta=0.1

GD lem - лемматизация:

- Минимальное время - 34.868 секунд при alpha=0.35, beta=0.1
- Максимальное время - 157.130 секунд при alpha=0.1, beta=0.2

SGD lem - лемматизация:

- Минимальное время - 17.495 секунд при alpha=0.1, beta=0.05
- Максимальное время - 1288.685 секунд при alpha=0.85, beta=0.1

GD removed - удаление стоа слов:

- Минимальное время - 11.570 секунд при alpha=0.85, beta=0.1
- Максимальное время - 135.677 секунд при alpha=0.1, beta=0.2

SGD removed - удаление стоа слов:

- Минимальное время - 7.431 секунд при alpha=0.35, beta=0.1
- Максимальное время - 935.191 секунд при alpha=0.85, beta=0.1

GD lem+removed - удаление стоа слов и лемматизация:

- Минимальное время - 11.907 секунд при alpha=0.85, beta=0.1
- Максимальное время - 121.548 секунд при alpha=0.1, beta=0.2

SGD lem+removed - удаление стоа слов и лемматизация:

- Минимальное время - 9.846 секунд при alpha=0.35, beta=0.1
- Максимальное время - 107.977 секунд при alpha=0.85, beta=0.1

Из этого анализа можно заключить, что использование сглаживания и удаления данных позволяет существенно сократить время работы алгоритмов, особенно для версий SGD

Точность:
GD обычный: 0.79416 при alpha=0.85, beta=0.1
SGD обычный: 0.79537 при alpha=0.35, beta=0.1
GD lem: 0.79769 при alpha=0.4, beta=0.2
SGD lem: 0.79798 при alpha=0.5, beta=0.05
GD removed: 0.81249 при alpha=0.35, 0.4, beta=0.1, 0.2
SGD removed: 0.81350 при alpha=0.4, 0.85, beta=0.2
GD lem+emoved: 0.81520 при alpha=0.85, beta=0.1
SGD lem+removed: 0.81607 при alpha=0.85, beta=0.1

Как и в случае времени работы, наибольшая точность достигается в модифицированных версиях алгоритмов с использованием сглаживания и удаления данных.

Можно сделать вывод, что выбор оптимальных параметров оптимизации,таких как сглаживание и удаление данных, позволяют существенно улучшить как время работы, так и точность алгоритмов GD и SGD.

3.8 Сравнение способов обработки датасета

min_df	Shape	Accuracy	Time (s)
100	(52061, 1905)	0.83474	64.928
300	(52061, 764)	0.82390	14.386
500	(52061, 470)	0.81607	111.986
700	(52061, 309)	0.80199	17.096
900	(52061, 240)	0.78942	5.608
1000	(52061, 211)	0.77945	6.730
2000	(52061, 82)	0.75237	2.698
5000	(52061, 17)	0.70280	0.547

Таблица 17: BagOfWords для MIN_DF

max_df	Shape	Accuracy	Time (s)
100	(52061, 13)	0.69806	0.092
300	(52061, 1144)	0.76204	78.558
500	(52061, 1435)	0.78415	95.989
700	(52061, 1596)	0.80518	56.819
900	(52061, 1665)	0.82274	28.201
1000	(52061, 1696)	0.82840	89.367
3000	(52061, 1865)	0.83212	251.266
5000	(52061, 1888)	0.84470	25.667

Таблица 18: BagOfWords для MAX_DF

min_df	Shape	Accuracy	Time (s)
100	(52061, 1905)	0.81138	8.584
300	(52061, 764)	0.80170	3.859
500	(52061, 470)	0.79571	2.412
700	(52061, 309)	0.78381	1.549
900	(52061, 240)	0.77181	1.506
1000	(52061, 211)	0.76146	1.119
2000	(52061, 82)	0.74512	0.693
5000	(52061, 17)	0.70023	0.365

Таблица 19: Tfidf для MIN_DF

min_df	Shape	Accuracy	Time (s)
100	(52061, 1905)	0.81138	8.584
300	(52061, 764)	0.80170	3.859
500	(52061, 470)	0.79571	2.412
700	(52061, 309)	0.78381	1.549
900	(52061, 240)	0.77181	1.506
1000	(52061, 211)	0.76146	1.119
2000	(52061, 82)	0.74512	0.693
5000	(52061, 17)	0.70023	0.365

Таблица 20: Tfidf для MAX_DF

Для BagOfWords и Tfidf точность уменьшается с увеличением min_df. Наилучшая точность достигается при min_df=100 (0.83474 для BagOfWords и 0.81138 для Tfidf). Для BagOfWords точность увеличивается с ростом max_df, достигая максимума 0.84470 при max_df=5000. Для Tfidf также наблюдается рост точности с увеличением max_df, но максимум 0.83130 достигается уже при max_df=1000.В целом BagOfWords показывает более высокую точность, чем Tfidf, при одинаковых значениях min_df и max_df. Размерность признакового пространства уменьшается с увеличением min_df для обоих представлений, а увеличение max_df повышает точность и размерность, но также увеличивает время обучения.

Время обучения в целом уменьшается с увеличением min_df для обоих представлений. Для BagOfWords время снижается с 64.928 с при min_df=100 до 0.547 с при min_df=5000. Для Tfidf время уменьшается с 8.584 с до 0.365 с при тех же min_df

3.9 Лучшая модель

Лучшие параметры для модели (исходя из предыдущего эксперимента):SGD, альфа = 0.05, бетта = 0.1, размер батча = 2000, BagOfWords с min_df = 100 и max_df = 5000.
Модель выдала точность 0.84470 (время работы 26.264 секунд)
Так же проведем анализ ошибок Всего ошибок: 3211
Ложноположительных : 776
Ложноотрицательных : 2435

Ложноположительные ошибки
Исходный: dear god this site is horrible
Лемм+удал: dear god site horrible
Исходный: i will burn you to hell if you revoke my talk page access
Лемм+удал: burn hell revoke talk page access

Ложноотрицательные ошибок:
Исходный: arabs are committing genocide in iraq but no protests in europe may europe also burn in hell
Лемм+удал: arab commit genocide iraq protest europe may europe also burn hell
Исходный: how dare you vandalize that page about the hms beagle don t vandalize again demon
Лемм+удал: dare vandalize page hm beagle vandalize demon

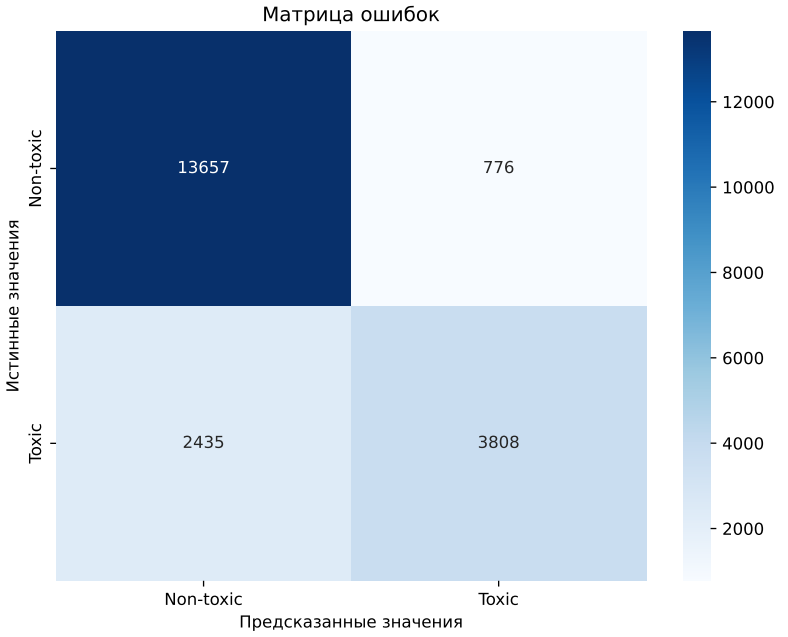


График 30: Матрица ошибок

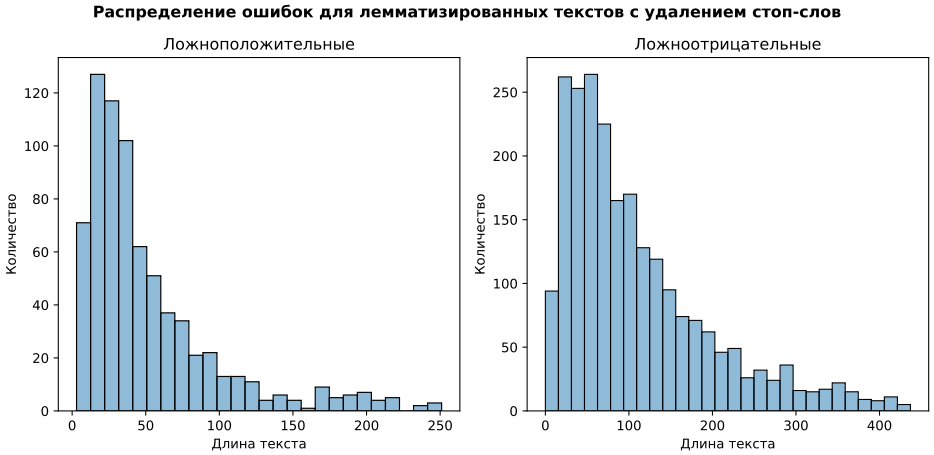


График 31: Зависимость числа ошибок от длины текста

Можно сделать вывод, что модель чаще ошибается на нетоксичных комментариях. Так же на графике видно, что с увеличением длины текста ошибок становится меньше. Два выведенных примера имеют короткую длину.

4 Заключение

В рамках данного исследования была построена модель классификации. Наибольшая точность, которая была достигнута **0.84470**. В данном исследовании были преобретены знания построения модели линейной регрессии, навыки обработки текстов, преобразование текстов в числовые признаки.

A Графики для анализа градиентного спуска (для *step_ alpha*)

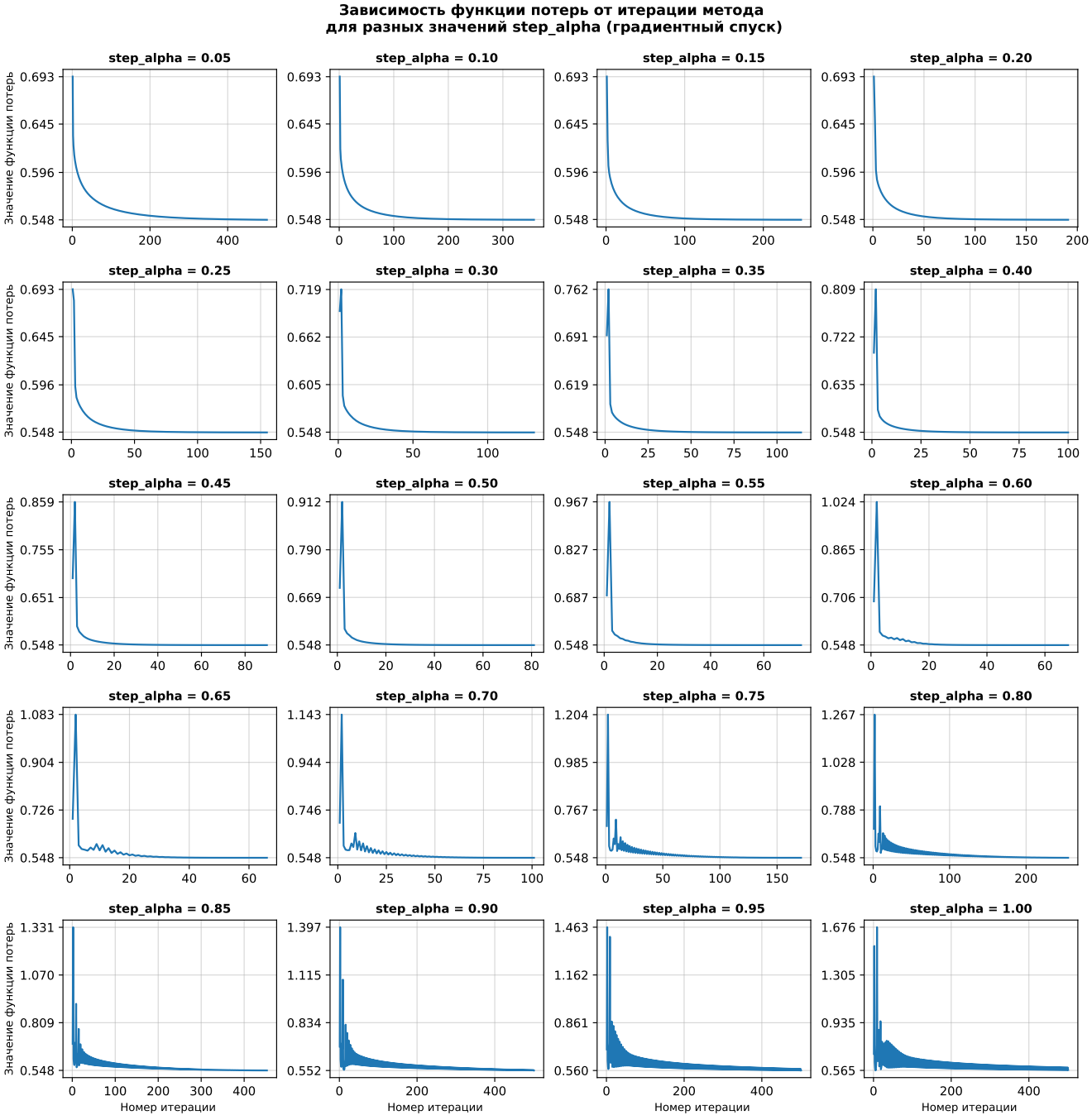


График 32: Зависимость функции потерь от итерации метода для разных значений *step_ alpha* (градиентный спуск)

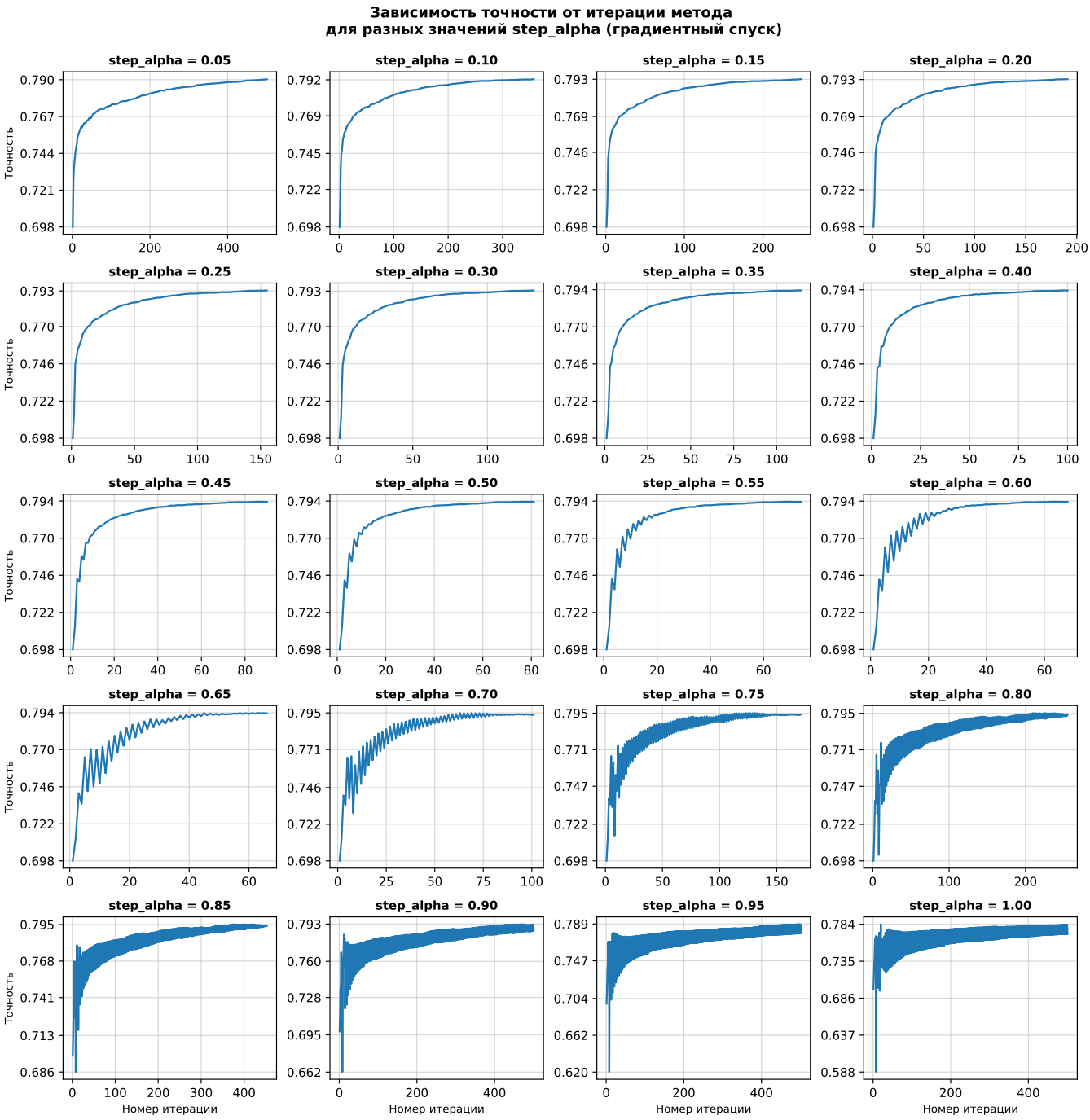


График 33: Зависимость точности от итерации метода для разных значений *step_ alpha* (градиентный спуск)

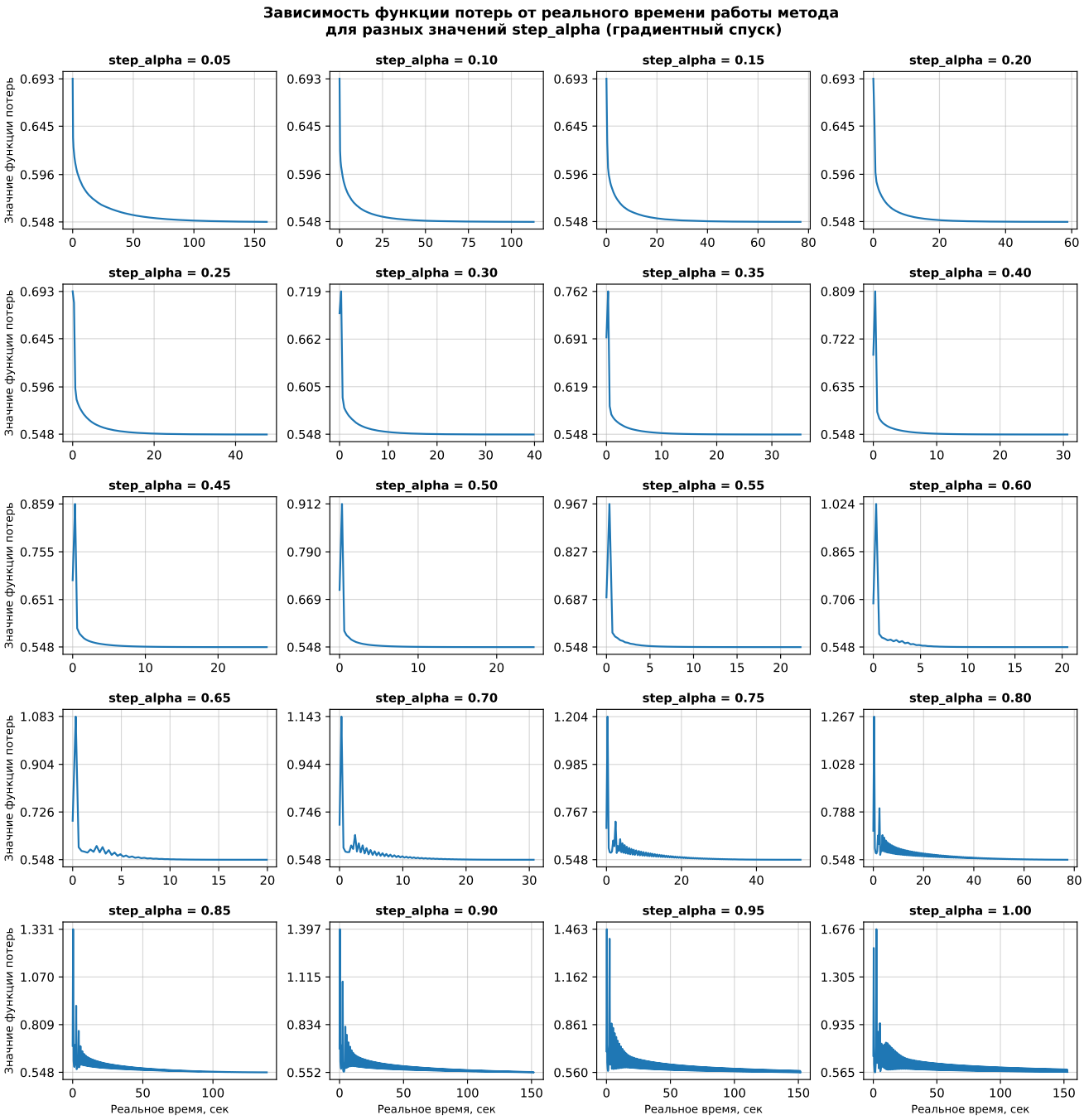


График 34: Зависимость функции потерь от реального времени работы метода для разных значений *step_ alpha* (градиентный спуск)

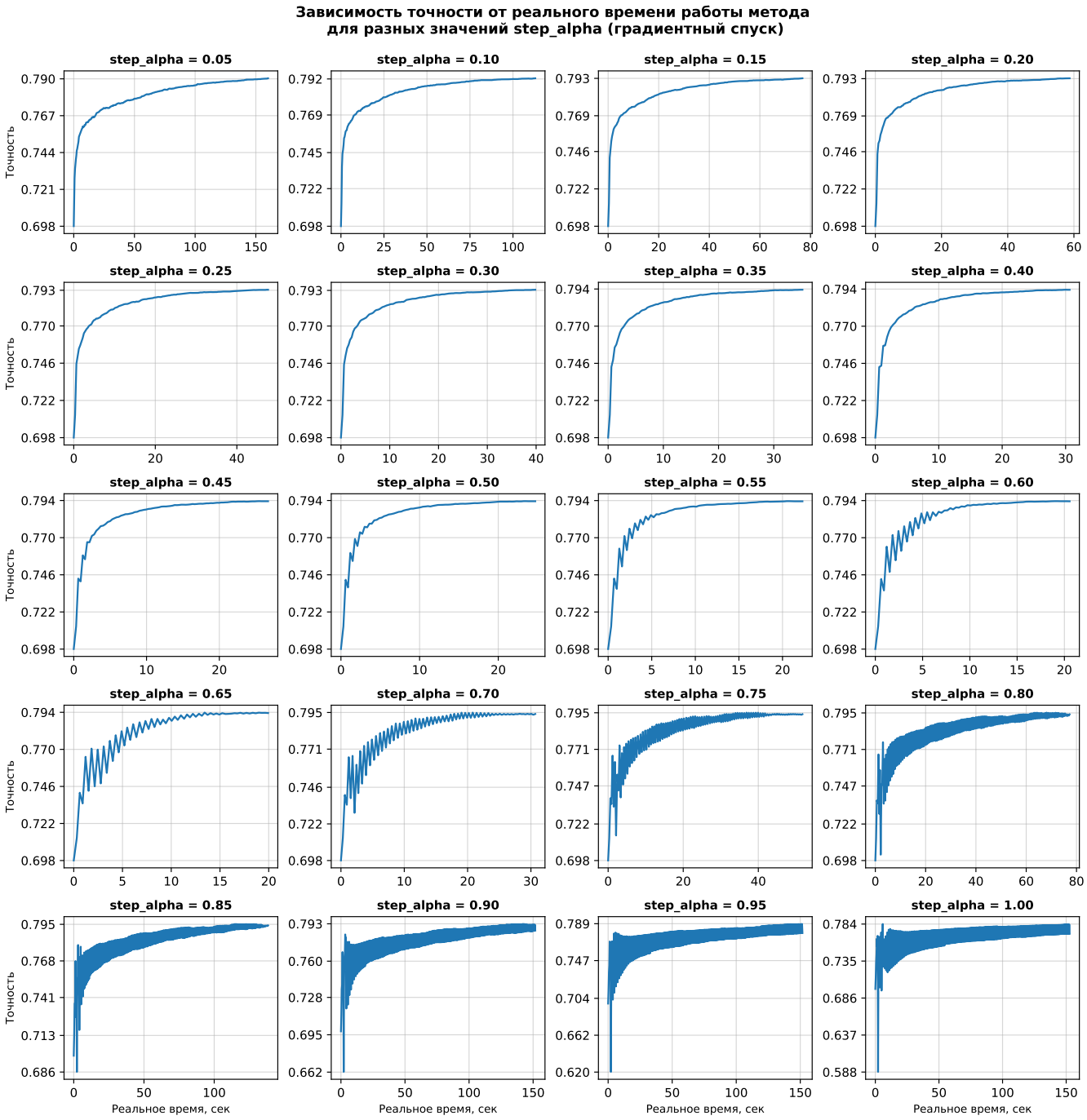


График 35: Зависимость точности от реального времени работы метода для разных значений *step_alpha* (градиентный спуск)

B Графики для анализа градиентного спуска (для *step_ beta*)

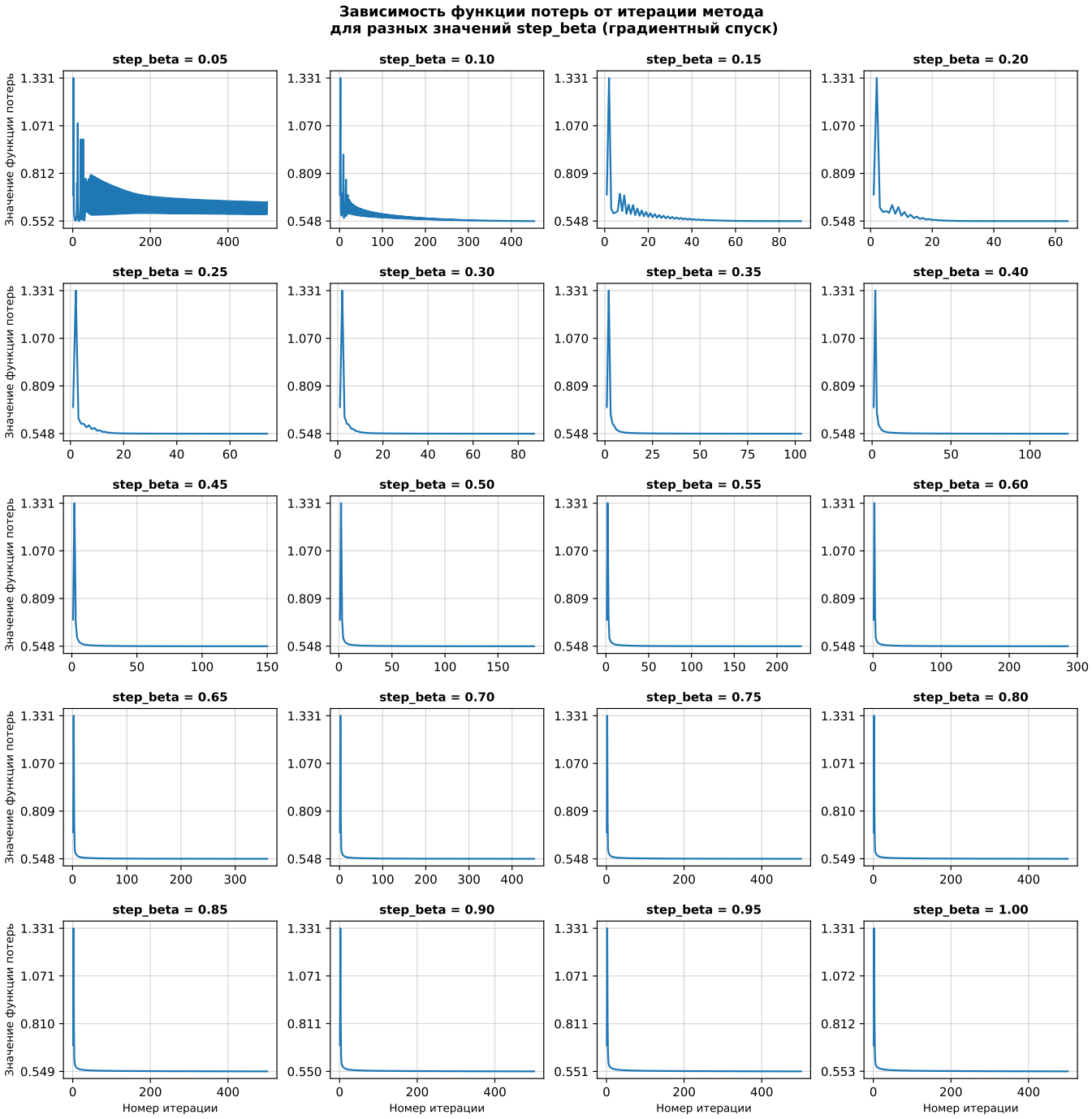


График 36: Зависимость функции потерь от итерации метода для разных значений *step_ beta* (градиентный спуск)

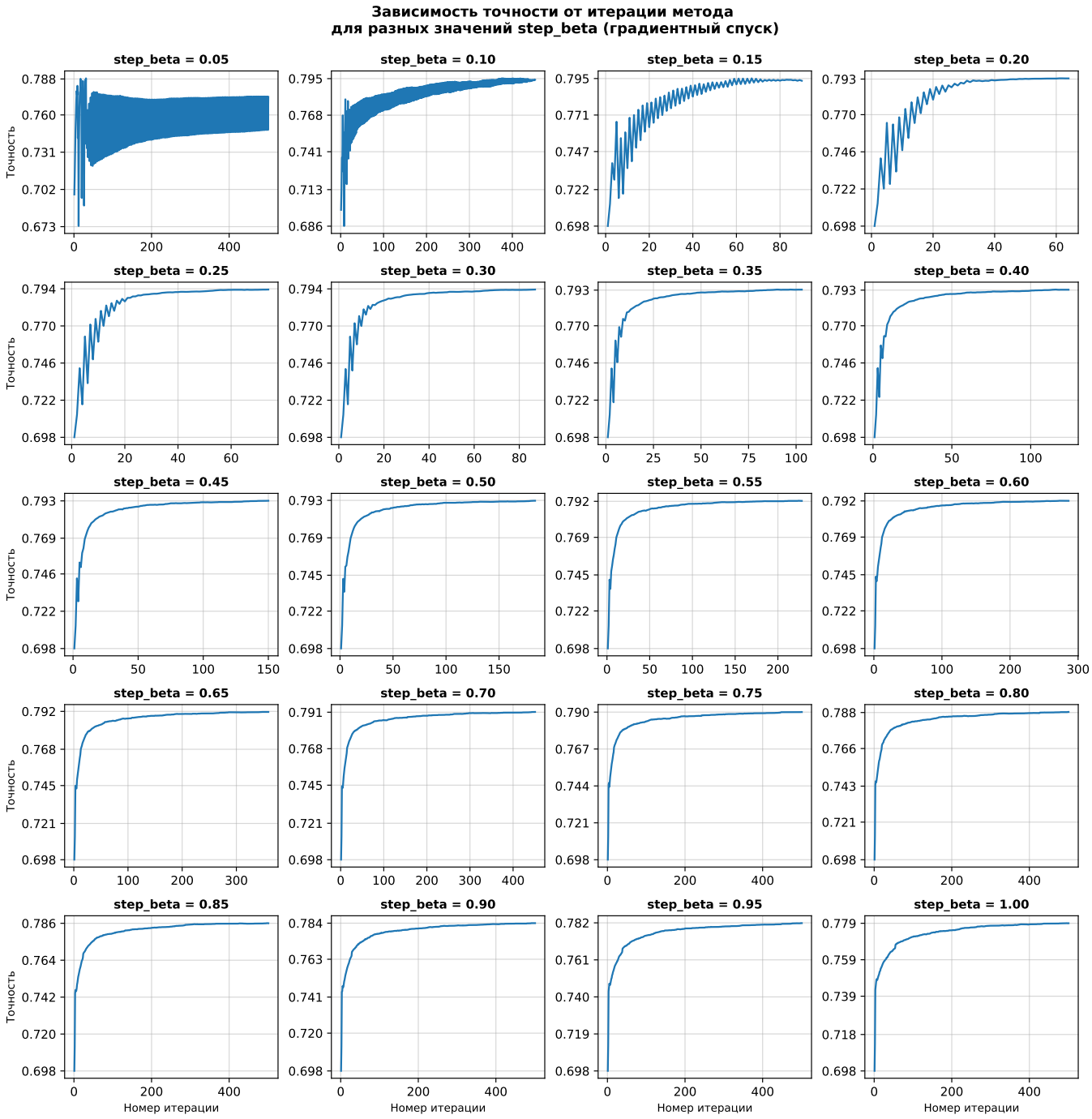


График 37: Зависимость точности от итерации метода для разных значений *step_ beta* (градиентный спуск)

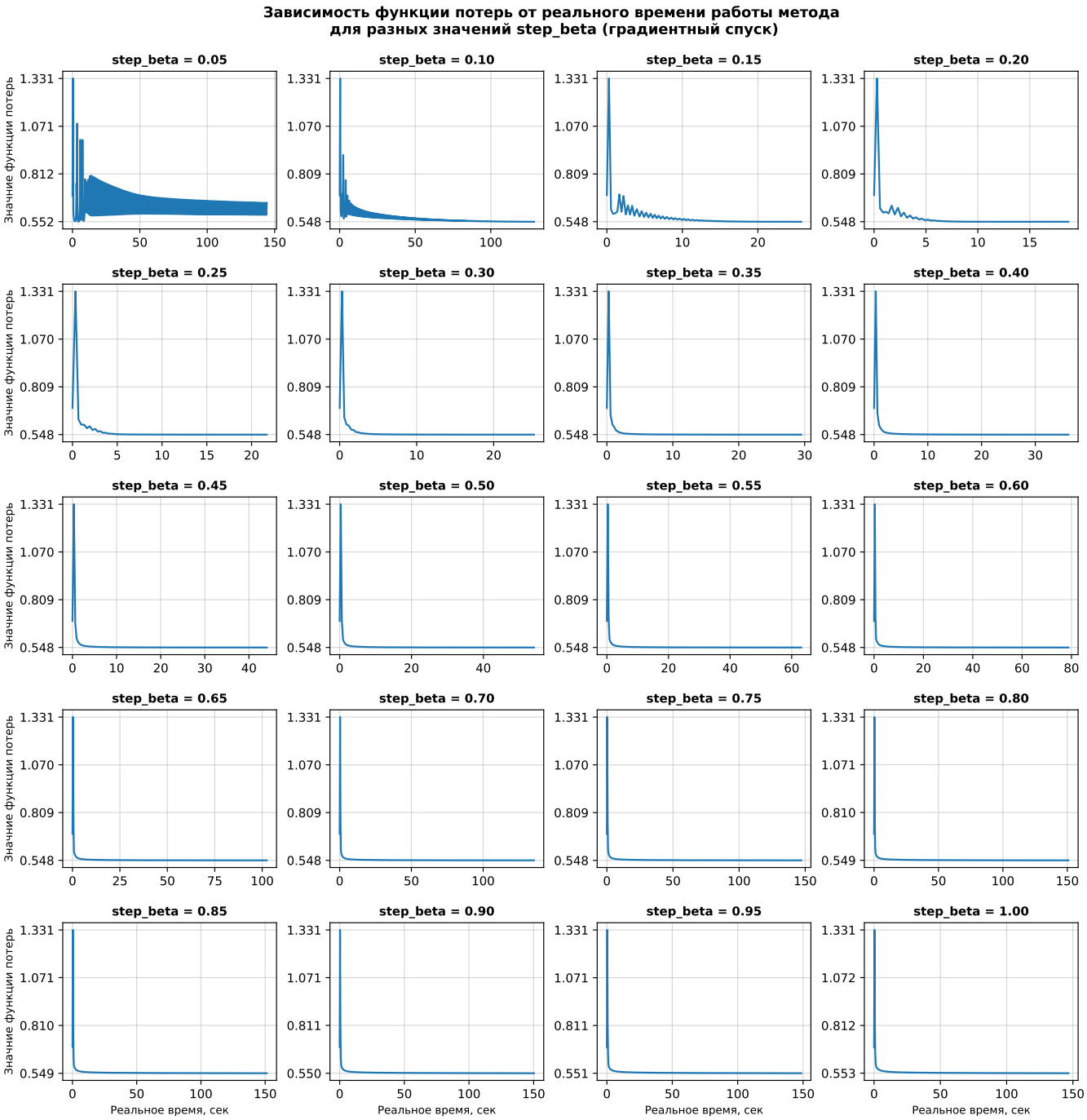


График 38: Зависимость функции потерь от реального времени работы метода для разных значений *step_ beta* (градиентный спуск)

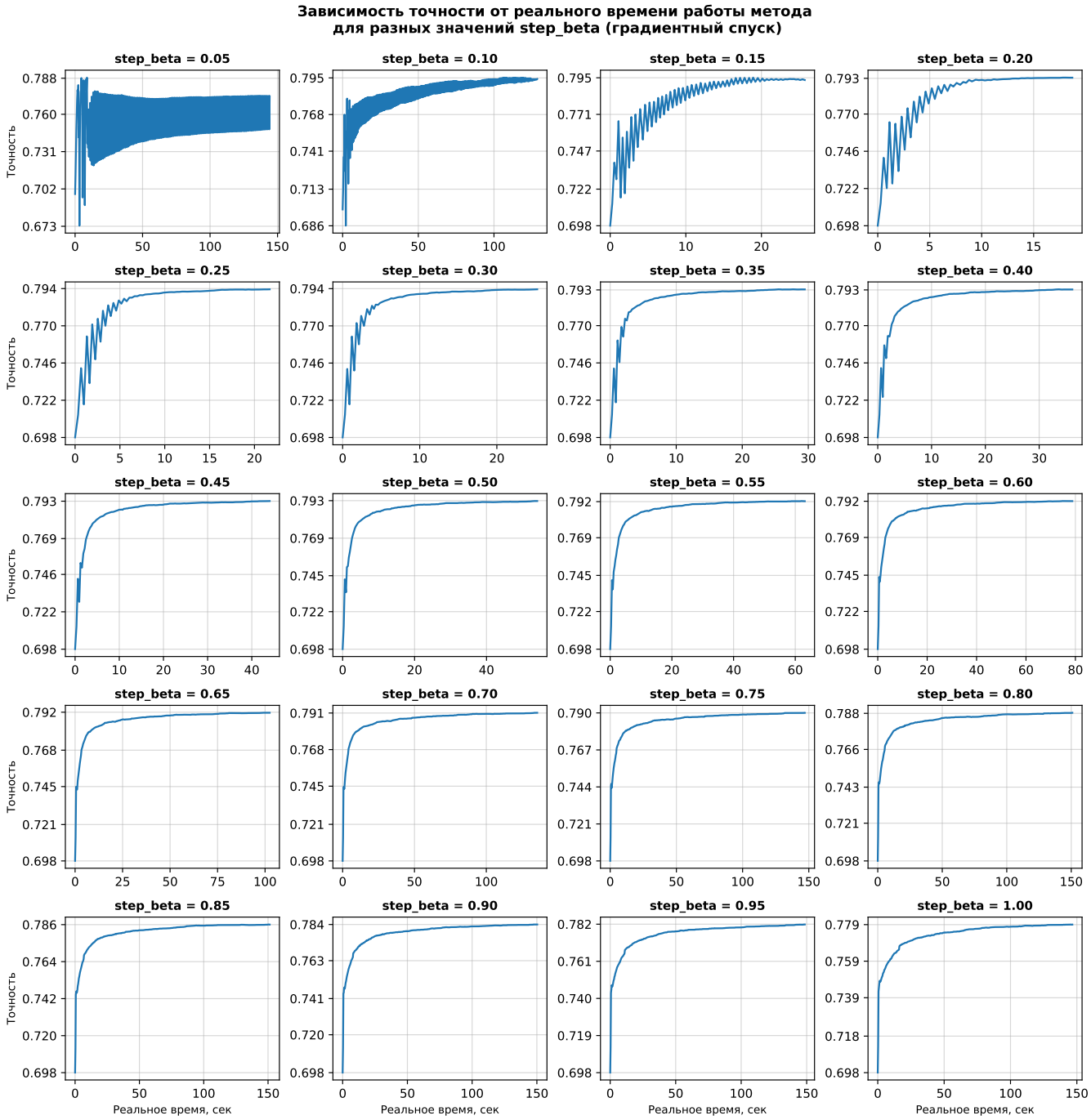


График 39: Зависимость точности от реального времени работы метода для разных значений *step_ beta* (градиентный спуск)

C Графики для анализа градиентного спуска (для начального приближения)

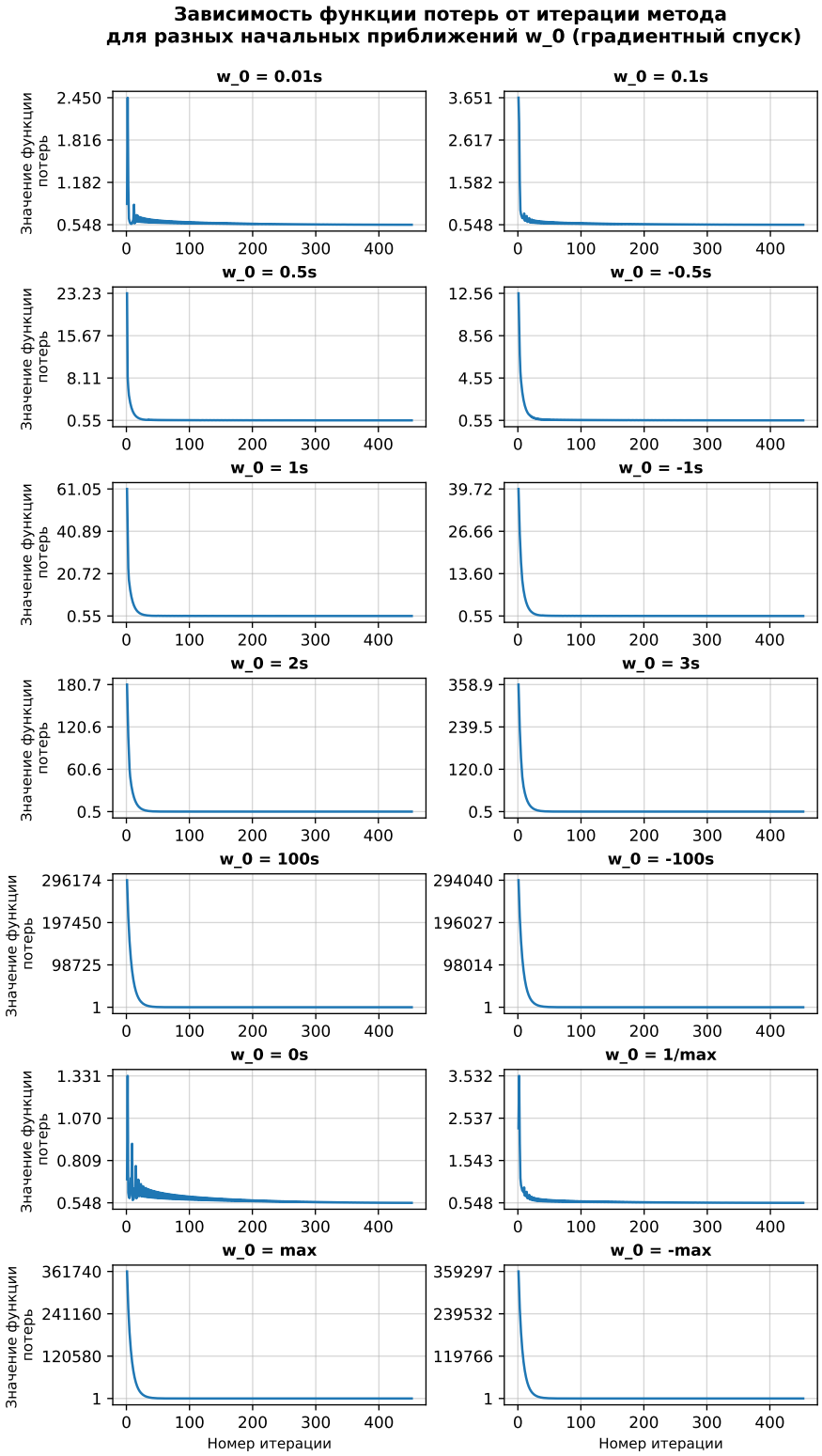


График 40: Зависимость функции потерь от итерации метода для разных w_0 (градиентный спуск)

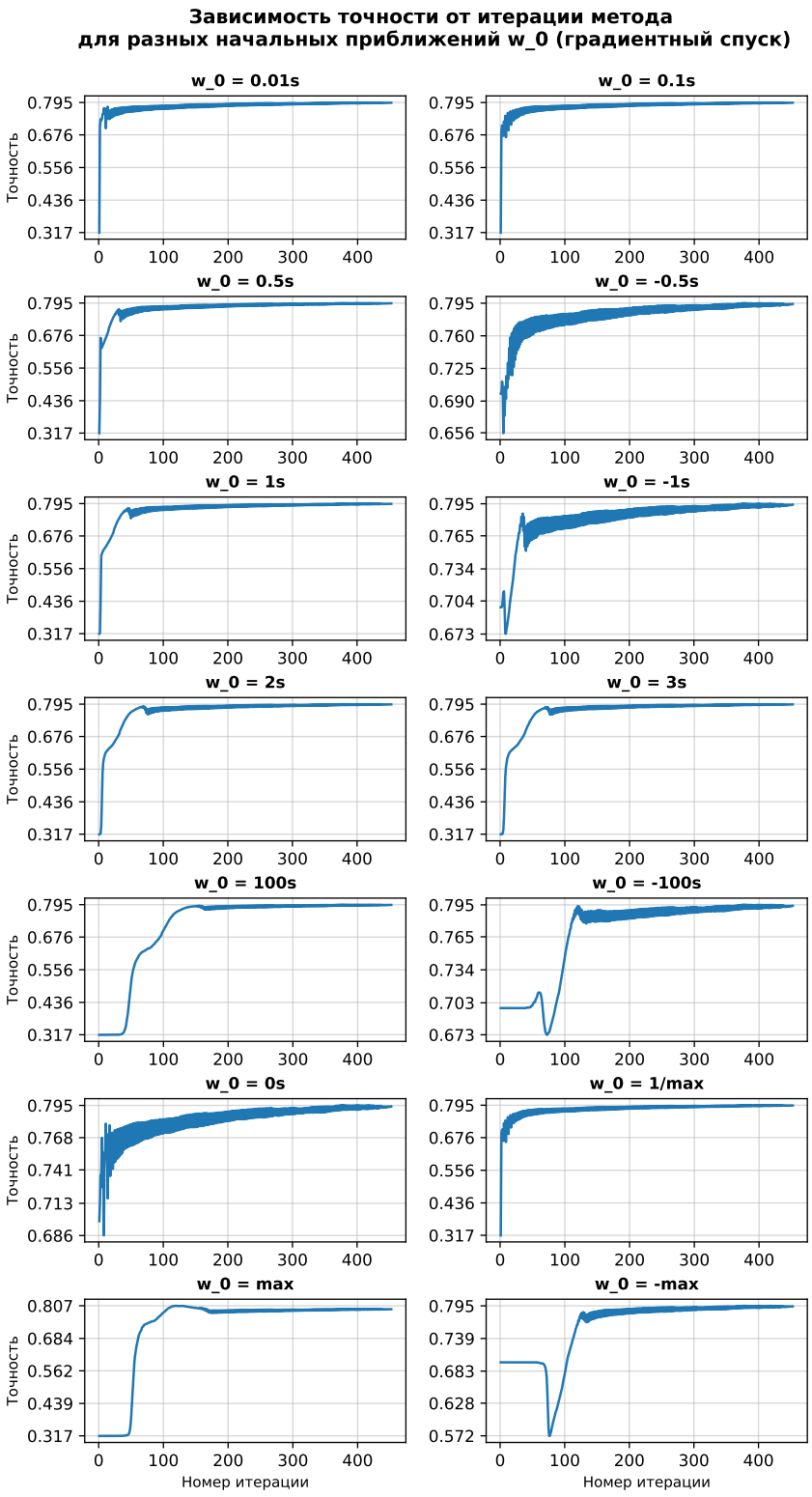


График 41: Зависимость точности от итерации метода для разных $w_0()$

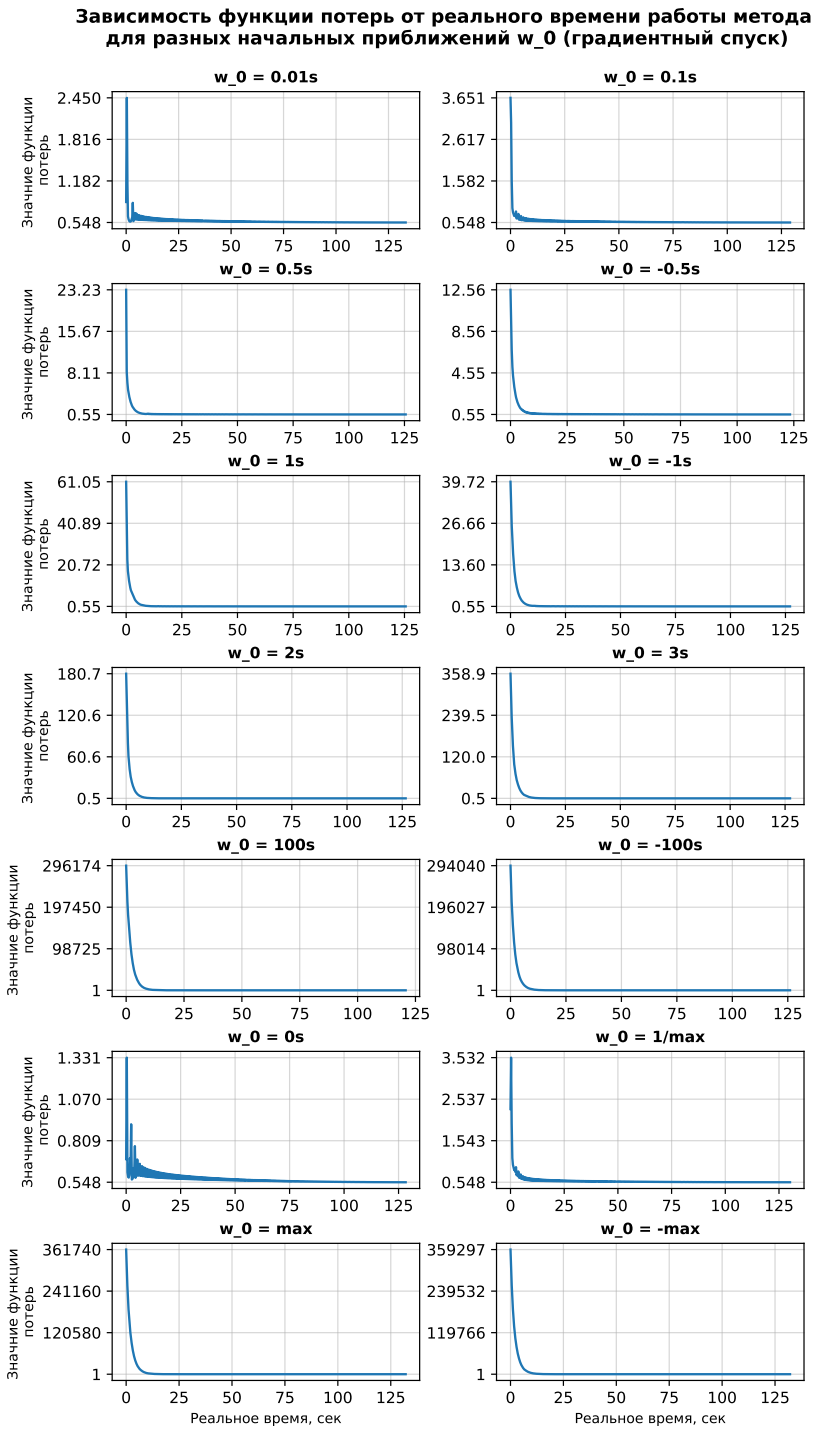


График 42: Зависимость функции потерь от реального времени работы метода для разных $w_0()$

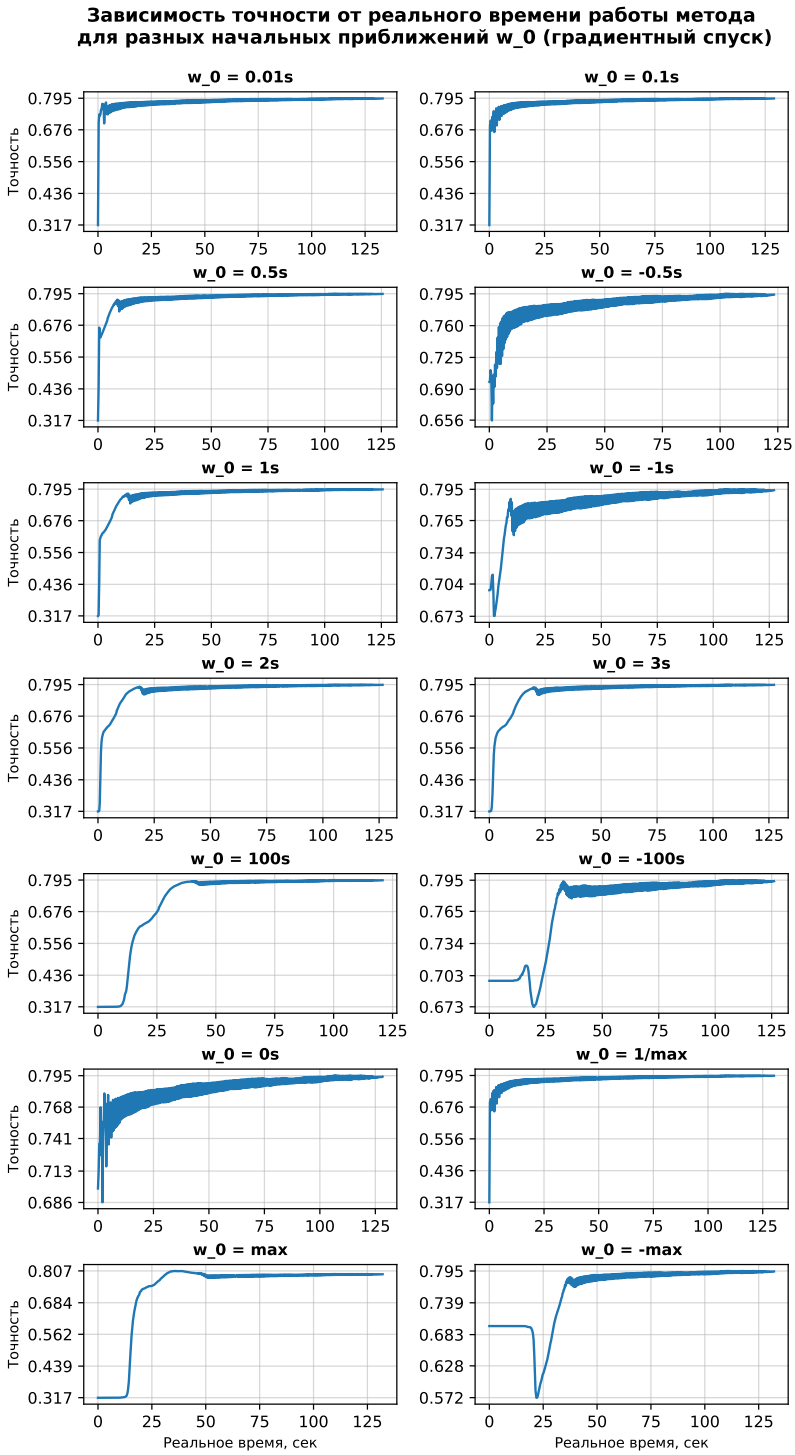


График 43: Зависимость точности от реального времени работы метода для разных w_0 (градиентный спуск)

D Графики для анализа стохастического градиентного спуска (для *step_ alpha*)

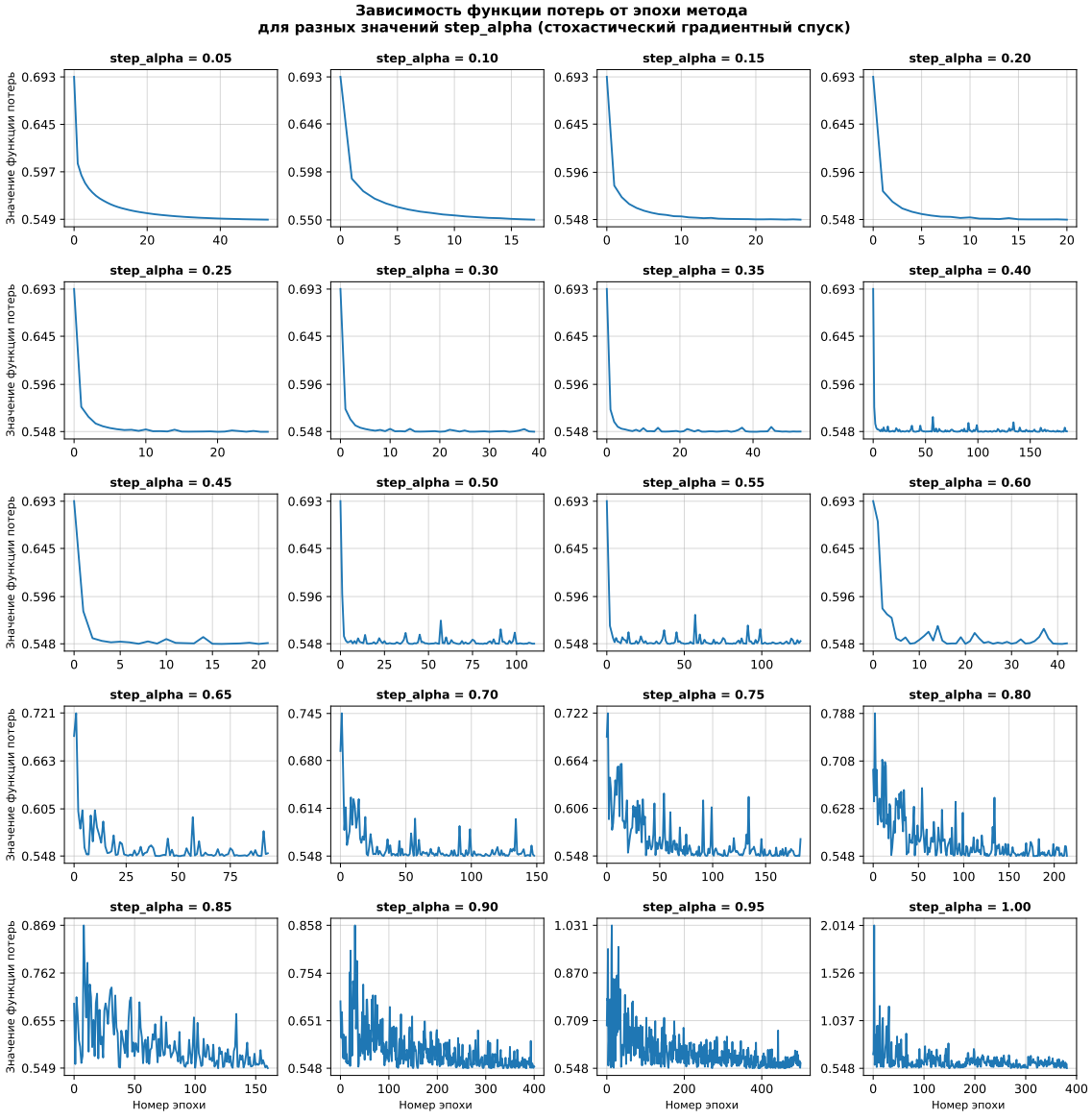


График 44: Зависимость функции потерь от итерации метода для разных значений *step_ alpha* (стохастический градиентный спуск)

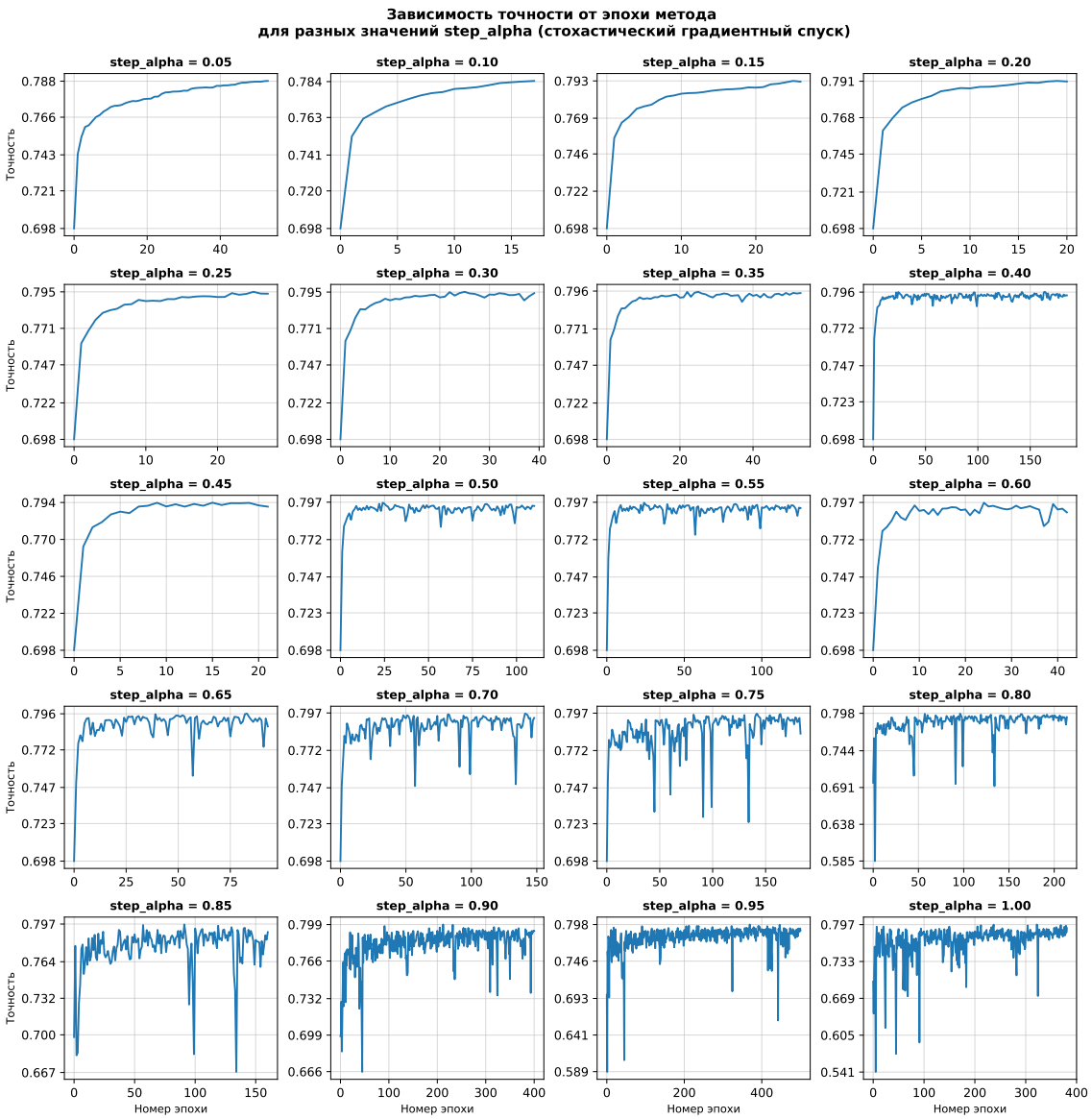


График 45: Зависимость точности от итерации метода для разных значений *step_ alpha* (стохастический градиентный спуск)

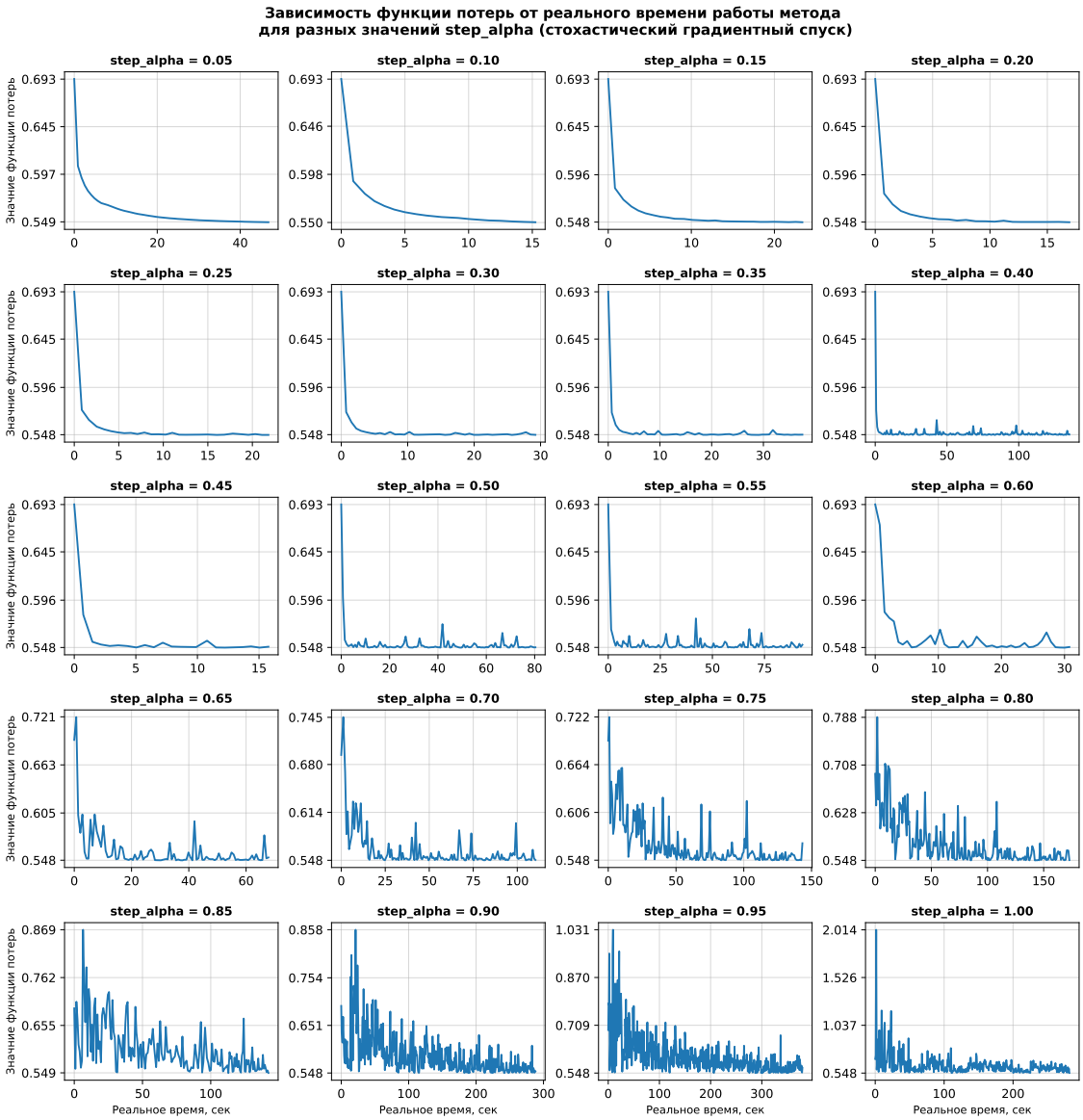


График 46: Зависимость функции потерь от реального времени работы метода для разных значений *step_ alpha* (стохастический градиентный спуск)

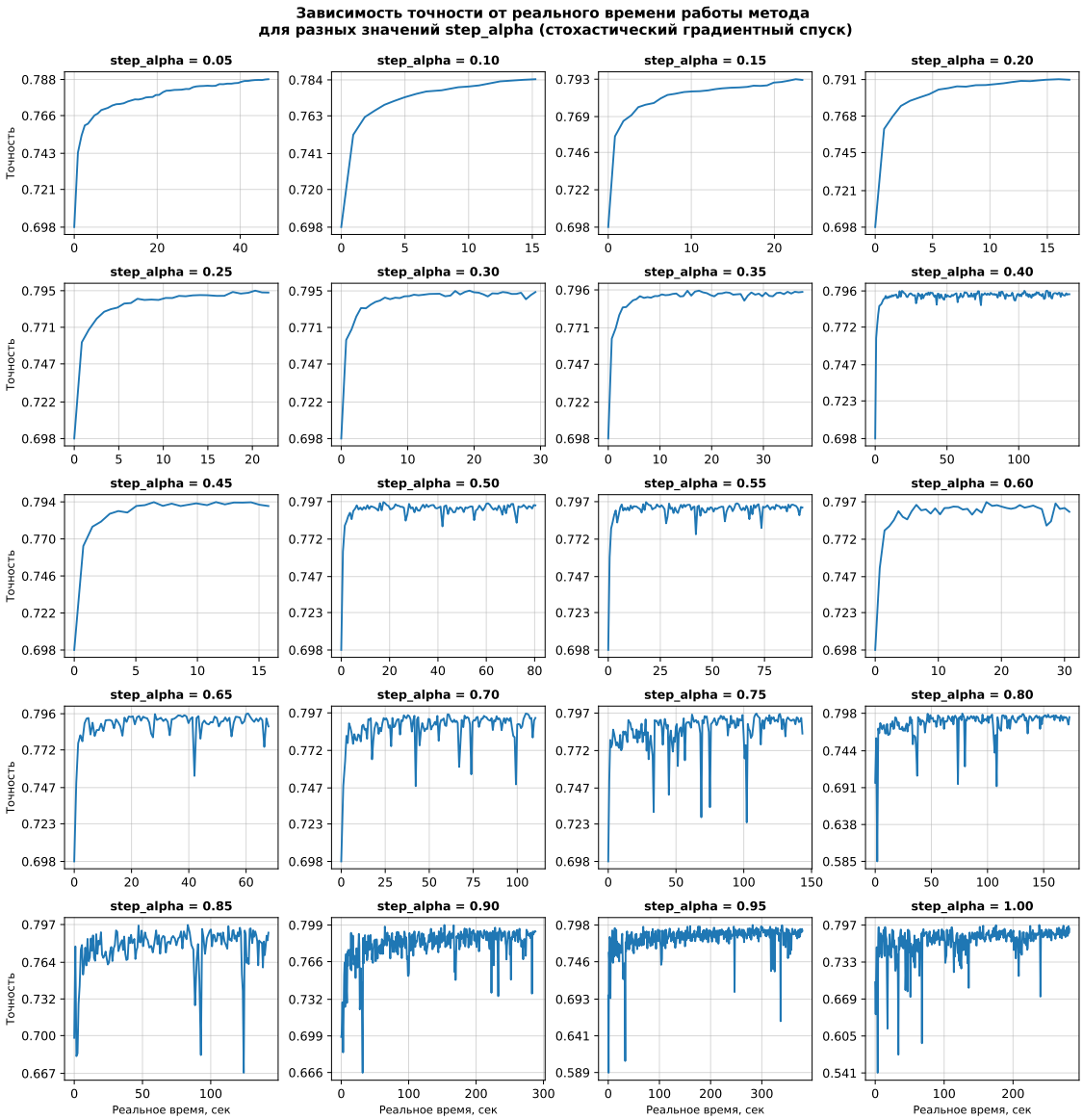


График 47: Зависимость точности от реального времени работы метода для разных значений *step_alpha* (стохастический градиентный спуск)

E Графики для анализа стохастического градиентного спуска (для *step_ beta*)

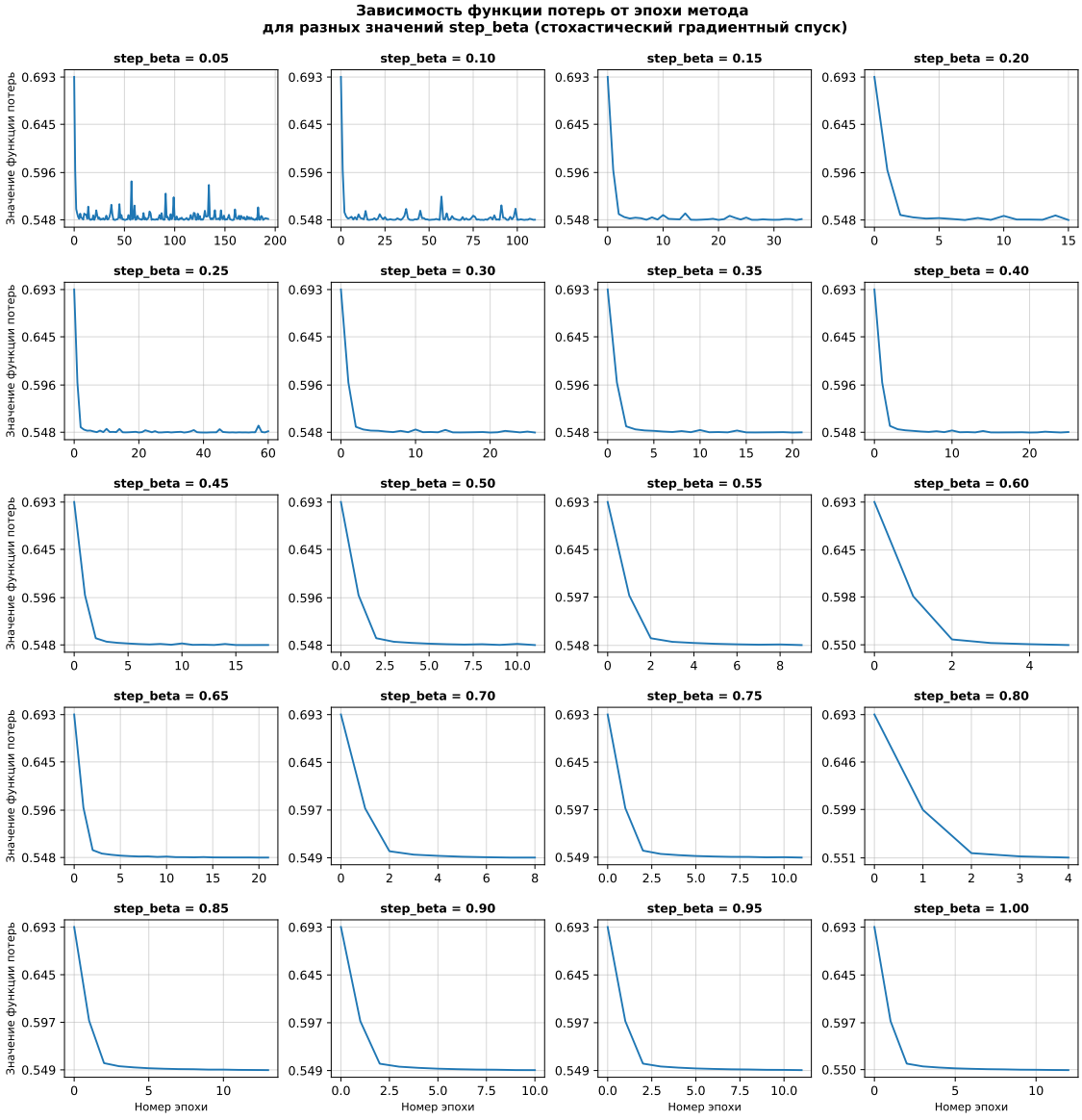


График 48: Зависимость функции потерь от итерации метода для разных значений *step_ beta* (стохастический градиентный спуск)

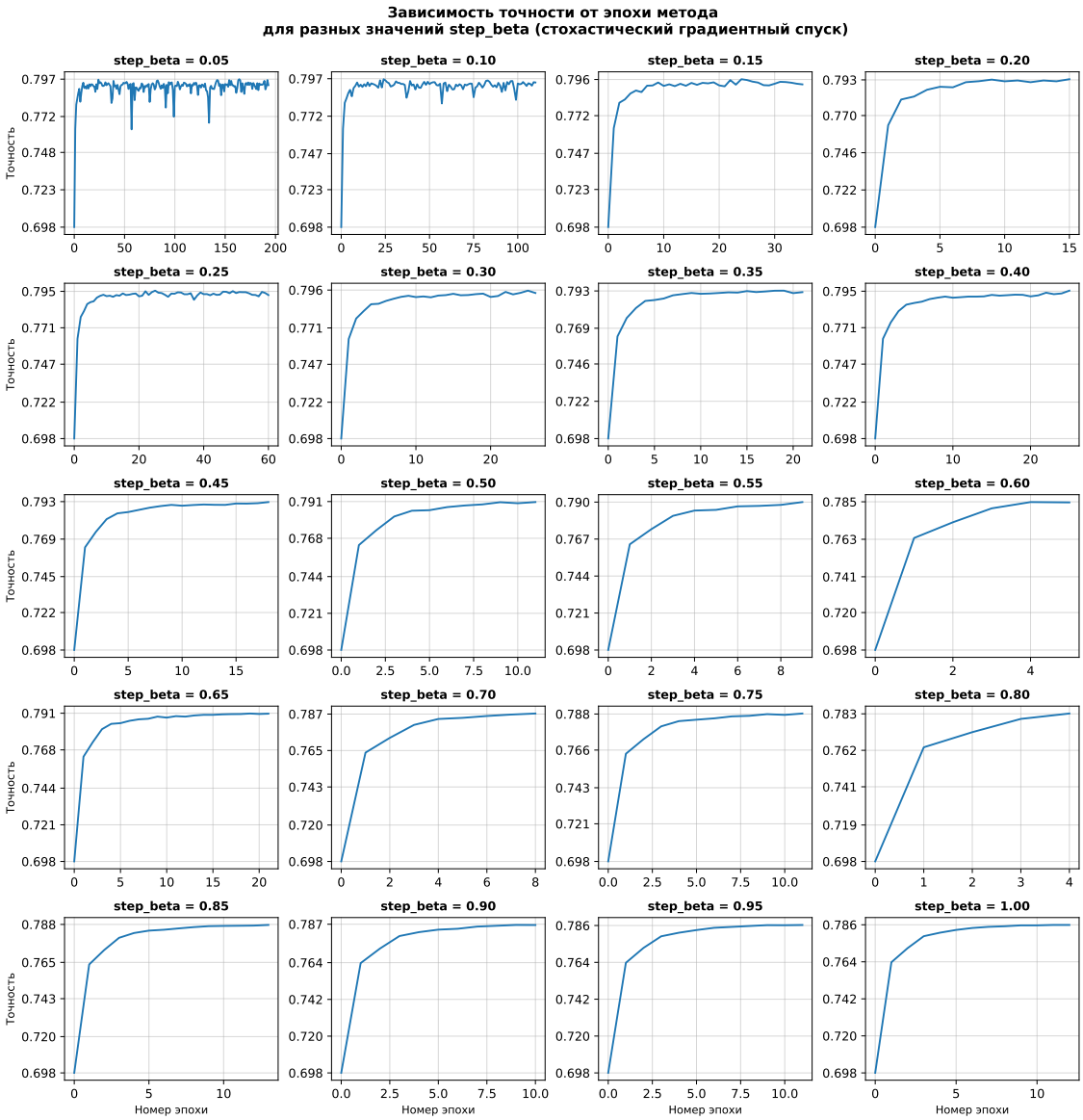


График 49: Зависимость точности от итерации метода для разных значений *step_ beta* (стохастический градиентный спуск)

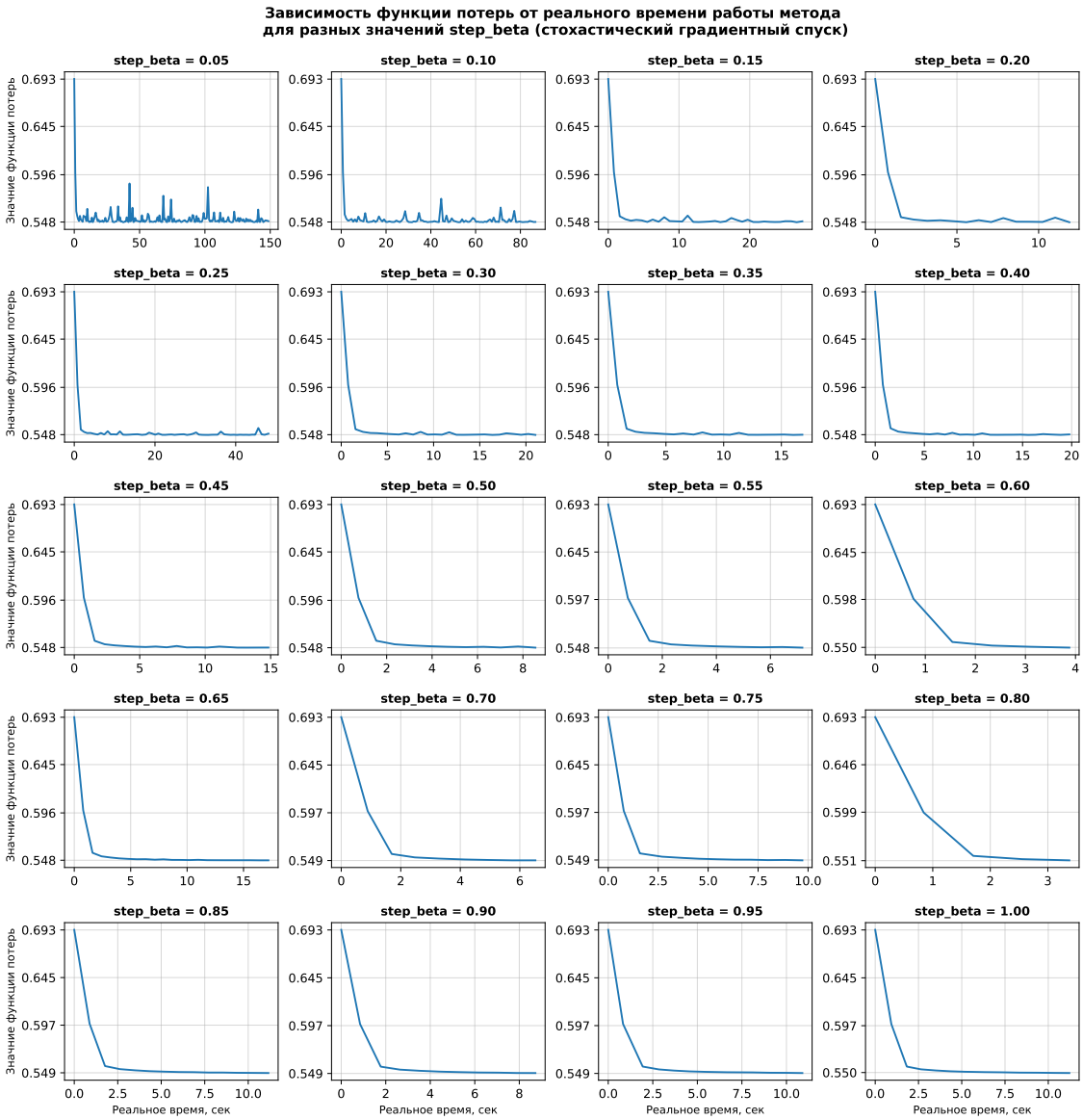


График 50: Зависимость функции потерь от реального времени работы метода для разных значений *step_beta* (стохастический градиентный спуск)

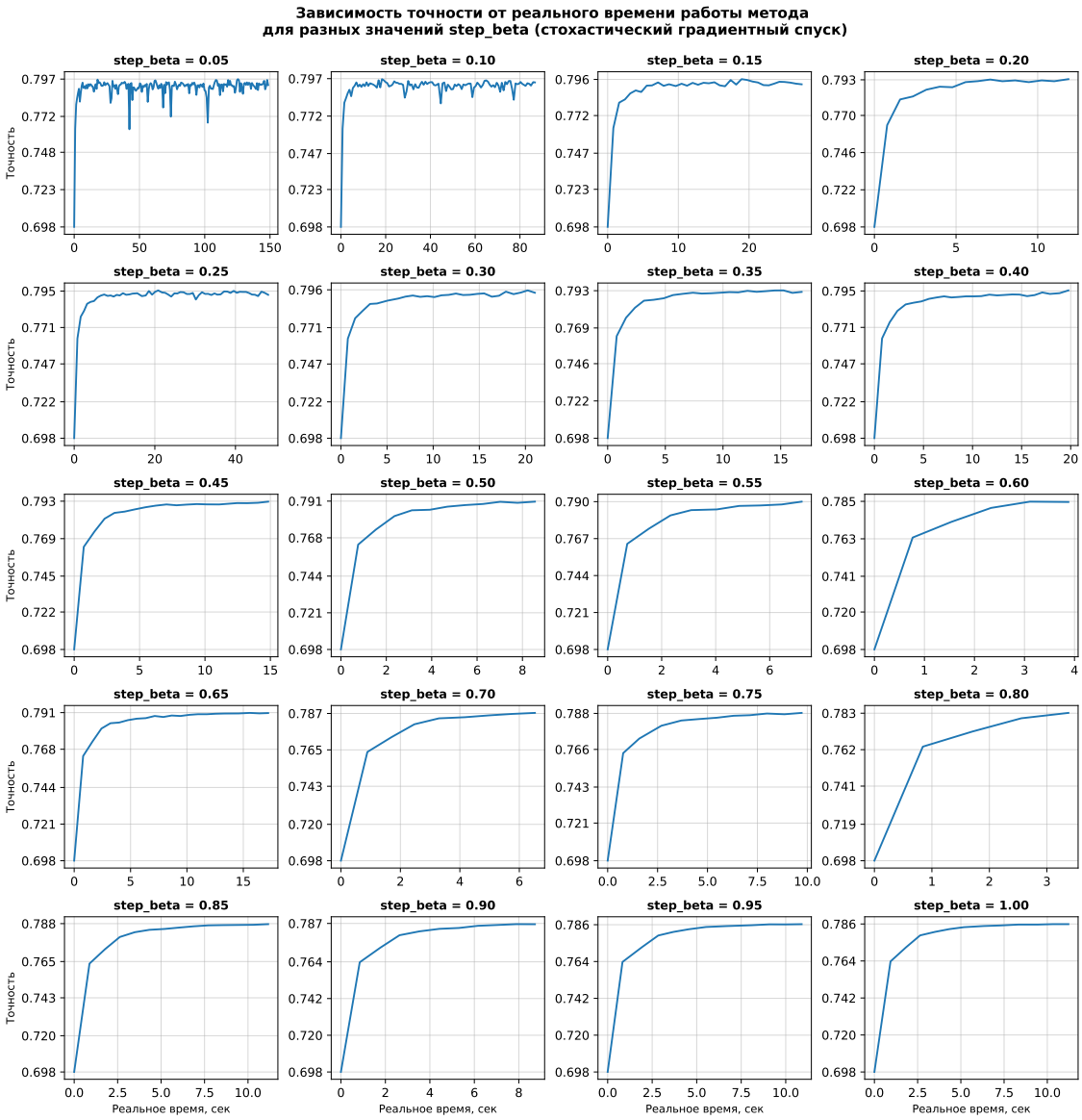


График 51: Зависимость точности от реального времени работы метода для разных значений *step_beta* (стохастический градиентный спуск)

F Графики для анализа стохастического градиентного спуска (для начального приближения)

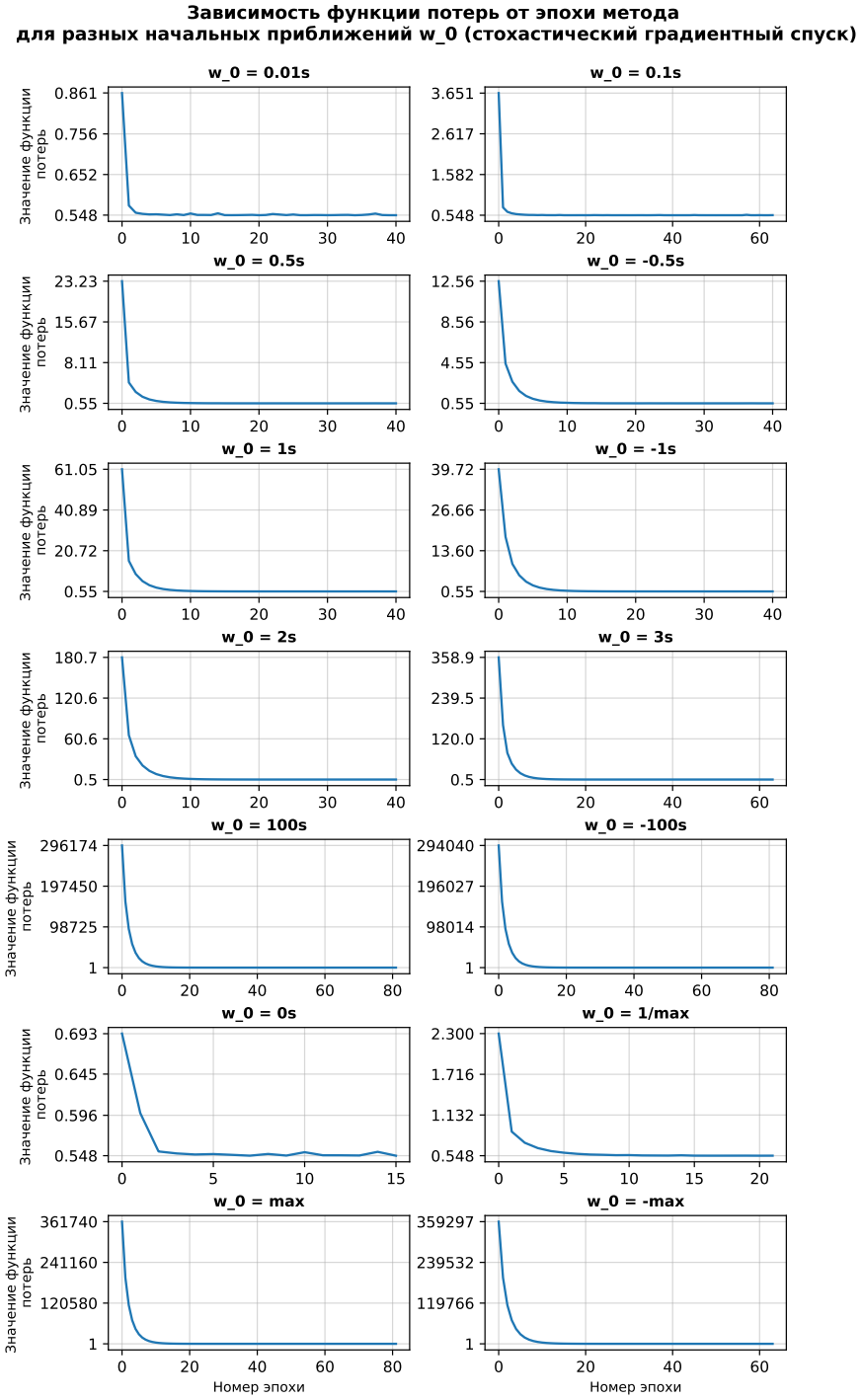


График 52: Зависимость функции потерь от итерации метода для разных w_0 (стохастический градиентный спуск)

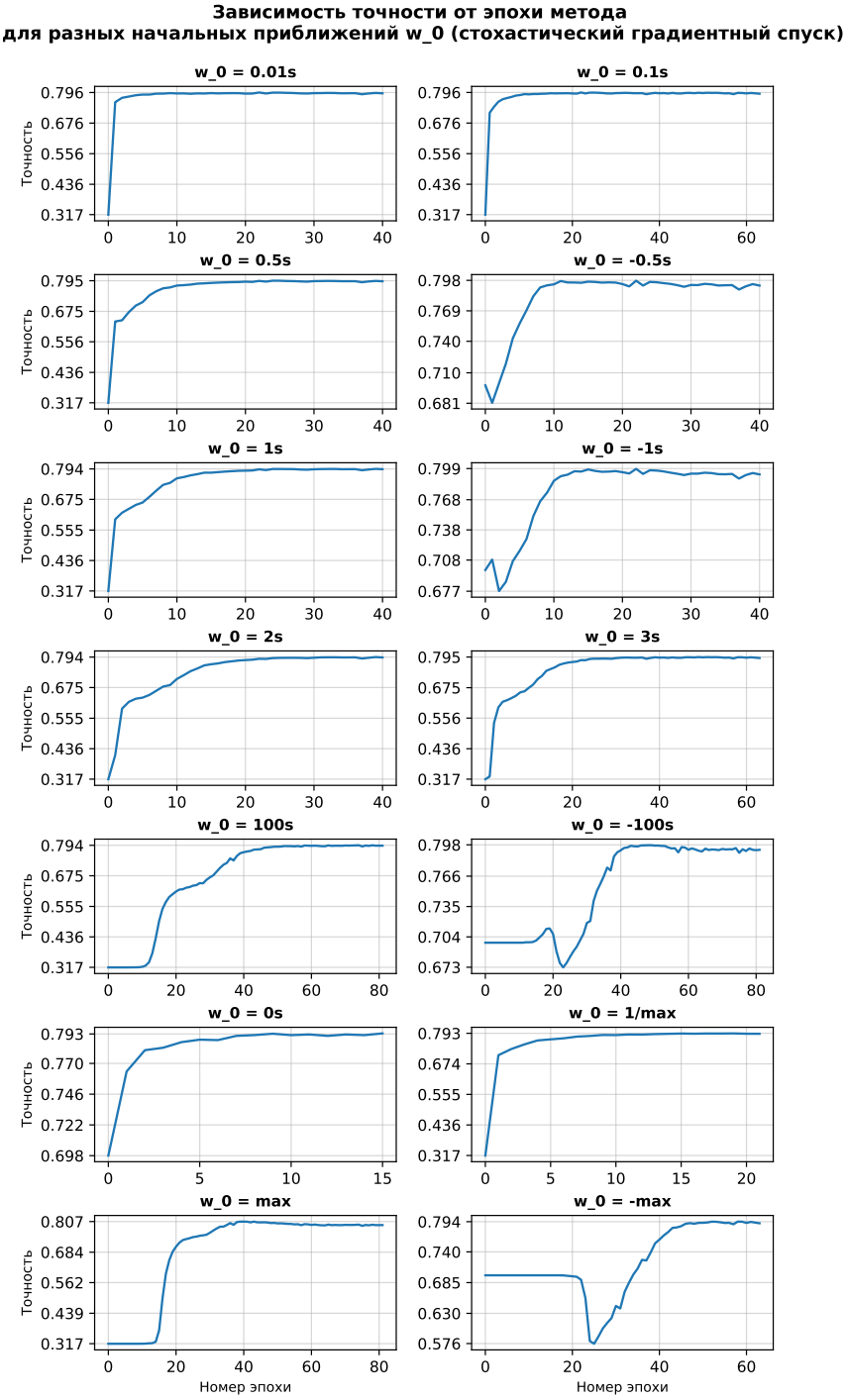


График 53: Зависимость точности от итерации метода для разных w_0 (стохастический градиентный спуск)

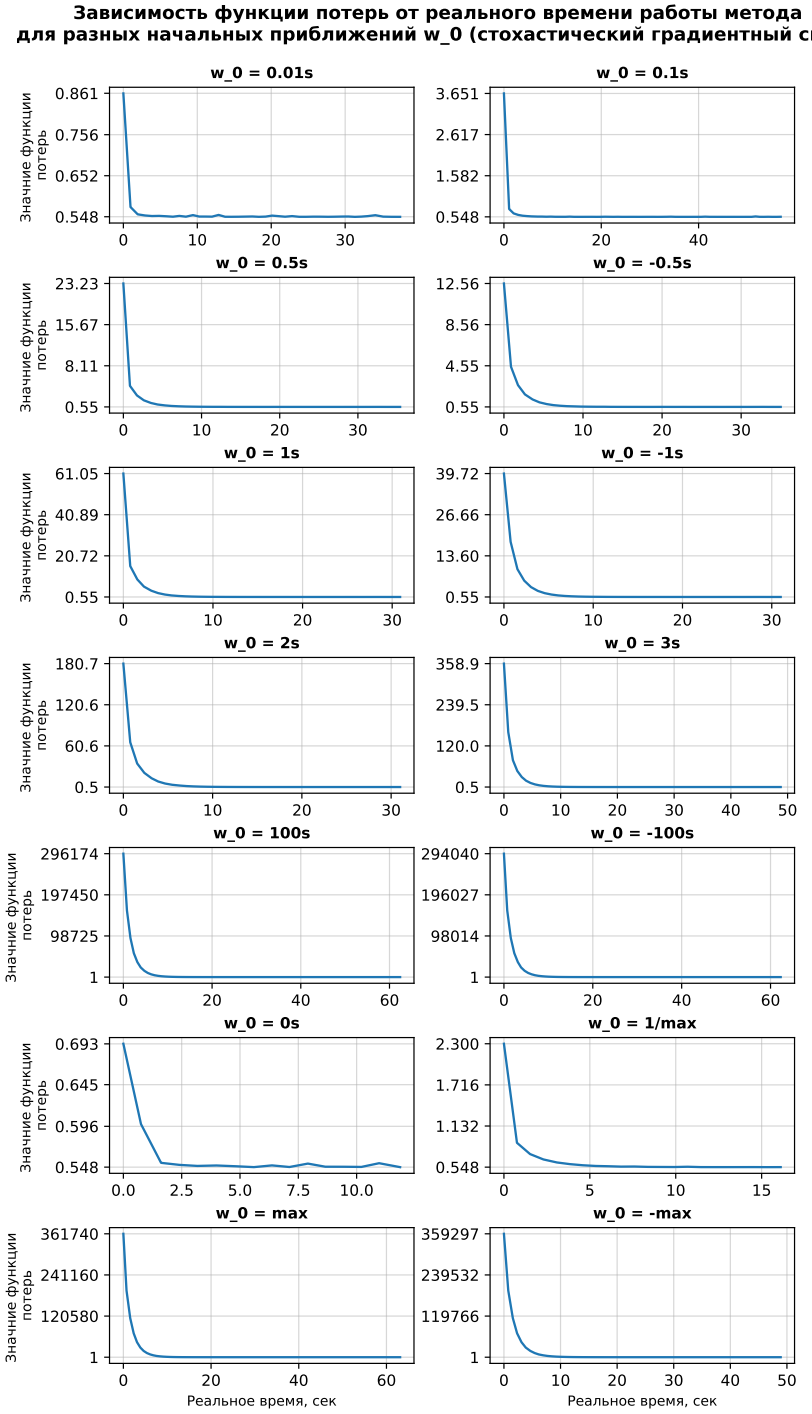


График 54: Зависимость функции потерь от реального времени работы метода для разных w_0 (стохастический градиентный спуск)

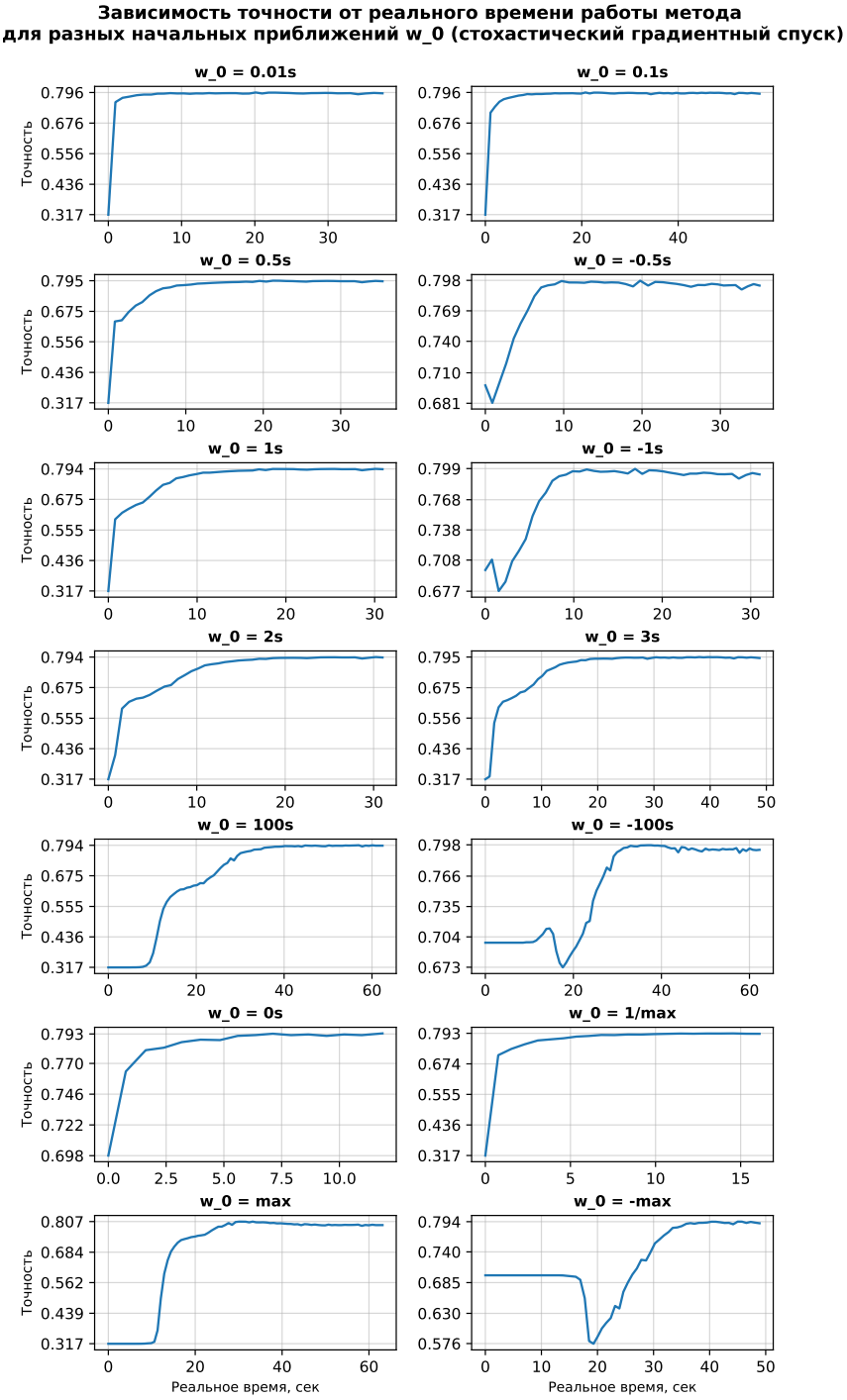


График 55: Зависимость точности от реального времени работы метода для разных w_0 (стохастический градиентный спуск)

Список литературы

[1] Toxic Comment Classification Challenge:
https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge