МГУ им.Ломоносова
Факультет ВМК кафедра ММП

Применение линейных моделей для определения токсичности комментария

Батшева Анастасия 317 группа

Москва
2020

Содержание

1 Формулировка задания

1. Написать на языке Python собственную реализацию линейного классификатора с произвольной функцией потерь.
2. Вывести все необходимые формулы и привести выкладки
3. Провести описанные ниже эксперименты с модельными данными и приложенным датасетом

2 Теоретическая часть

2.1 Вывод формулы градиента функции потерь для задачи бинарной логистической регрессии

Логистическая регрессия по выборке с l2-регуляризацией (Ridge) определяется функцией потерь вида:

$$L(w) = \frac{1}{l}\sum_{i=1}^{l}\log(1 + e^{-y_i\langle x_i, w\rangle}) + \frac{\lambda}{2}\|w\|_2^2$$

Найдем дифференциал с помощью матричного дифференциирования:

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

и, соответственно, градиент:

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

2.2 Вывод формулы градиента функции потерь для задачи многоклассовой логистической регрессии

Для мультиномиальной регрессии с k классами строится k линейных моделей $\alpha_1(x), ..., \alpha_k(x)$, где каждая модель $\alpha_i(x)$ задает вероятность, объект x с которой принадлежит к классу i:

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

Значит, логистическая регрессия с l2-регуляризацией задается следующей функцией ошибки:

$$Q(w) = -\frac{1}{l}\sum_{i=1}^{l}\log(\mathbb{P}(y_i|x_i)) + \frac{\lambda}{2}\sum_{j=1}^{k}\|w_j\|_2^2 =$$

Найдем дифференциал по заданному вектору w_n:

$$d\log(\mathbb{P}(y_i|x_i)) = \frac{d(\frac{e^{\langle x,w_i\rangle}}{\sum_{j=1}^k e^{\langle x,w_j\rangle}})}{\frac{e^{\langle x,w_i\rangle}}{\sum_{j=1}^k e^{\langle x,w_j\rangle}}} = \frac{\frac{x_i e^{\langle x,w_i\rangle}\sum_{j=1}^k e^{\langle x,w_i\rangle} - e^{\langle x,w_i\rangle}e^{\langle x,w_n\rangle}x_i}{(\sum_{j=1}^k e^{\langle x,w_i\rangle})^2}}{\frac{e^{\langle x,w_i\rangle}}{\sum_{j=1}^k e^{\langle x,w_i\rangle}}} =$$

$$= \frac{x_i\sum_{j=1}^k e^{\langle x_i,w_i\rangle} - e^{\langle x_i,w_n\rangle}x_i}{\sum_{j=1}^k e^{\langle x_i,w_i\rangle}} = x_i([y_i = n] - \frac{e^{\langle x_i,w_n\rangle}}{\sum_{j=1}^k e^{\langle x_i,w_j\rangle}})$$

$$\Rightarrow dQ(w) = \langle\frac{1}{l}\sum_{i=1}^l(\frac{e^{\langle x,w_n\rangle}}{\sum_{i=1}^k e^{\langle x,w_n\rangle}} - [y_i = n])x_i + \lambda w_n, dw_n\rangle$$

Значит, градиент по нему w_n равен:

$$\bigtriangledown Q(w) = \frac{1}{l}\sum_{i=1}^l(\frac{e^{\langle x,w_n\rangle}}{\sum_{i=1}^k e^{\langle x,w_n\rangle}} - [y_i = n])x_i + \lambda w_n$$

2.3 Сведение многоклассовой логистической регрессии с k = 2 к задаче бинарной логистической регрессии

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

$\mathbb{P}(y = 1|x) + \mathbb{P}(y = 2|x) = 1 \Rightarrow \mathbb{P}(y = 2|x) = 1 - \mathbb{P}(y = 1|x)$
Значит, задача сведена к бинарной классификации.

2.4 Градиентный спуск

Для численного решения задачи оптимизации (минимизация функции ошибки) будем использовать итерационные методы, а именно градиентный спуск: Если F - функция ошибки, то

$$F(w_{i+1}) = F(w_i) - \frac{\alpha}{i^\beta}\bigtriangledown F(w_i)$$

α и β в экспериментах называются `step_alpha` и `step_beta`.

3 Экспериментальная часть

Датасет для данного задания представляет собой статистику комментариев из раздела обсуждений английской Википедии, который был преобразован для решения задач бинарной классификации: является ли данный комментарий токсичным или нет.

Список эесперименов:

1. Произвести предварительную обработку датасета: привести все тексты к нижнему регистру, избавиться от символов, не являющихся ни цифрами, ни баквами (заменить их пробелами)

2. Преобразовать датасет в разреженную матрицу

3. Исследовать поведение градиентного спуска для задачи логистической регрессии в зависимости от параметров: `step_alpha, step_beta, w_0` - начальное приближение

4. Исследовать поведение стохастического градиентного спуска для задачи логистической регрессии в зависимости от параметров: `step_alpha, step_beta, bat` - размер подвыборки `w_0` - начальное приближение

5. Сравнить поведение двух методов между собой

6. Применить алгоритм лемматизации к коллекции, удалить из текста стопслова. Исследовать, как предобработка корпуса повлияла на точность классификации, время работы и размерность признакового пространства

7. Исследовать качество, время работы алгоритма и размерность признакового пространства в зависимости от следующих факторов:

   - Использовать представление BagOfWords или Tfidf
   - Параметров `min_df, max_df` конструкторов

8. Выбрать лучший алгоритм для тестовой выборки и применить его. Проанализировать ошибки алгоритма. Проанализировать общие черты объектов, на которых были допущены ошибки.

4 Результаты экспериментов

4.1 Предобработка датасета

До и после первых преобразований датасета:

0	Explanation\nWhy the edits made under my usern...	0	explanation why the edits made under my userna...
1	D'aww! He matches this background colour I'm s...	1	d aww he matches this background colour i m se...
2	Hey man, I'm really not trying to edit war. It...	2	hey man i m really not trying to edit war it s...
3	"\nMore\nI can't make any real suggestions on ...	3	more i can t make any real suggestions on impr...
4	You, sir, are my hero. Any chance you remember...	4	you sir are my hero any chance you remember wh...
	...		...
159566	":::::And for the second time of asking, when ...	159566	and for the second time of asking when your vi...
159567	You should be ashamed of yourself \n\nThat is ...	159567	you should be ashamed of yourself that is a ho...
159568	Spitzer \n\nUmm, theres no actual article for ...	159568	spitzer umm theres no actual article for prost...
159569	And it looks like it was actually you who put ...	159569	and it looks like it was actually you who put ...
159570	"\nAnd ... I really don't think you understand...	159570	and i really don t think you understand i came...

Анализ датасета показывает, что классы (токсичный / нетоксичный) несбалансированны:

```
Test:Train = 49.0% : 51.0%
Train size: 159571 toxic: 9.6%
Test size: 153164 toxic: 4.0%
```

После этого датасет преобзразуется в разреженную матрицу со словами (выделенными из комментариев) с помощью `sklearn.feature_extraction.text.CountVe` с параметром `min_df = 0.0001`. При таком параметре слова, у которых частота появления меньше, чем в 0.01% документов, будут проигнорированы. Такой выбор позволяет ускорить проведение экспериментов, существенно снизив размерность данных.

```
min_df = 1      <159571x182232 sparse matrix ...
min_df = 0.0001 <159571x17033 sparse matrix ...
```

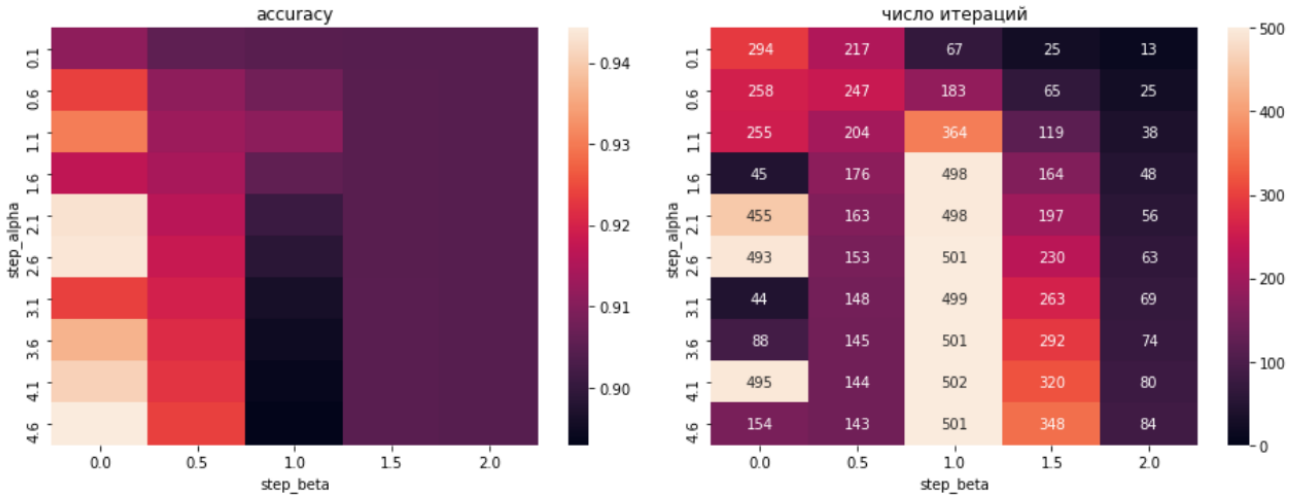
Так вместо 180 тысяч остается всего 17. Приблизительно 10%.

4.2 Исследование градиентного спуска

4.2.1 Зависимость от параметров шага

Начальное приближение w_0 нулевое

Создадим датафрейм `df_history` с экспериментами: `step_alpha` перебирается от 0.1 до 5 с шагом 0.5, а `step_beta` от 0 до 2 с шагом в 0.5. Помимо перебираемых параметров в датафрейме присутствуют время выполнения, итоговая точность (accuracy), итоговое значение функции ошибки, а так же history для каждого спуска. Подобный перебор занимает около 32 минут. Зависимость можно визуализировать с помощью тепловой карты:



Видно, что качество ухудшается при увеличении `step_beta`. Это логично, ведь при росте `step_beta` i^β растет, а значит, $\frac{\alpha}{i^\beta}$ стремится к нулю. То есть функции ошибки на соседних шагах оказываются достаточно близки, чтобы спуск остановился (разница меньше tolerance) прежде, чем он достигнет искомого минимума. С ростом же `step_beta`, напротив, качество улучшается засчет увеличения числа необходимых шагов в спуске.

$$F(w_{i+1}) = F(w_i) - \frac{\alpha}{i^\beta} \bigtriangledown F(w_i)$$

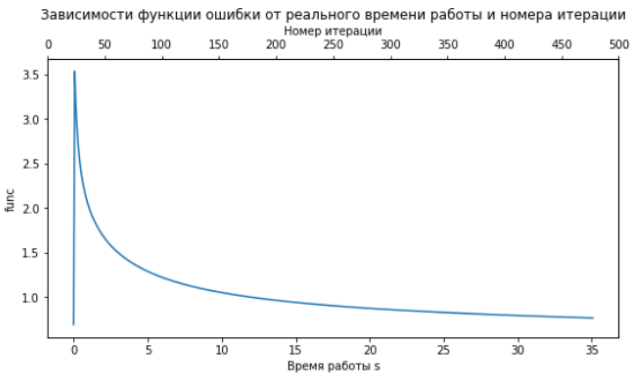
Справа карта показывает число итераций. Можно заметить, что при росте `step_beta` от 1 спуск очень рано останавливается: для `step_beta` $= 0.1$ `step_beta` $= 2.0$ он делает всего 13 итераций, в то время как для `step_beta` $= 2.6$ `step_beta` $= 0.0$ он почти достигает `max_iter` и при этом с хорошей точностью.

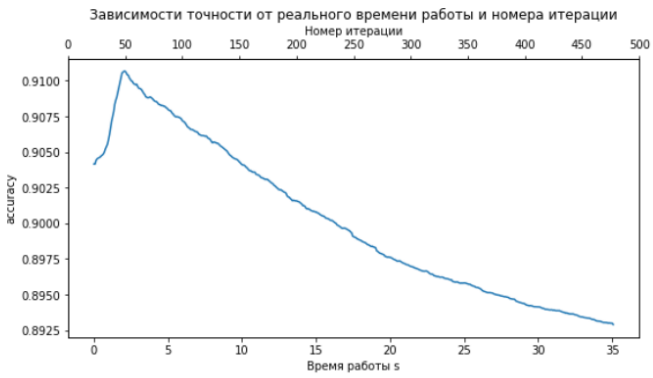
Визуализируем несколько спусков, а именно построим графики зависимости:

- Значения функции потерь от реального времени работы алгоритма
- Значения функции потерь от итерации метода
- Значения точности от реального времени работы алгоритма
- Значения точности от итерации метода

Выберем спуски с достаточным числом шагов. Начнем от самого некачественного спуска:

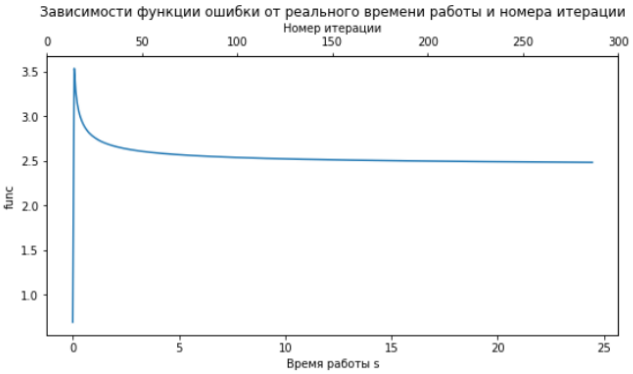
`step_alpha: 4.6, step_beta: 1.0`

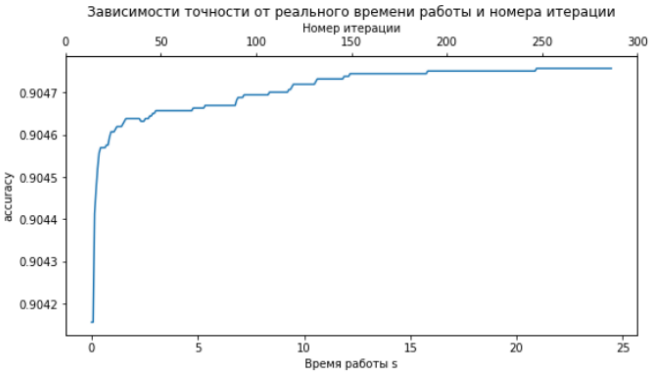




Учитывая малость коэффициента при градиенте спуск не сходится и модель неустойчива.

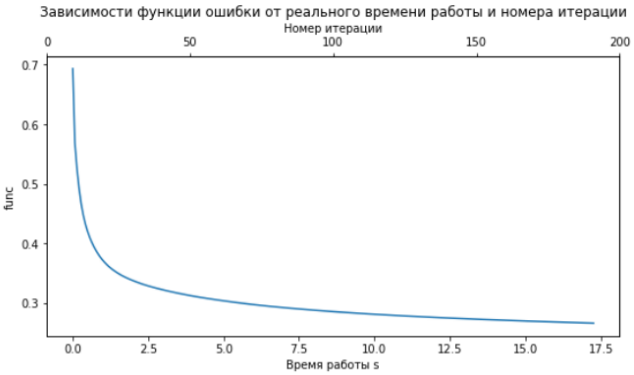
`step_alpha: 4.6, step_beta: 1.5`





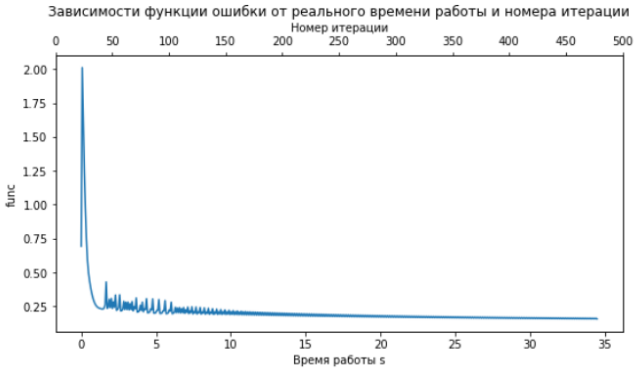
Видим, что сходимость улучшается.

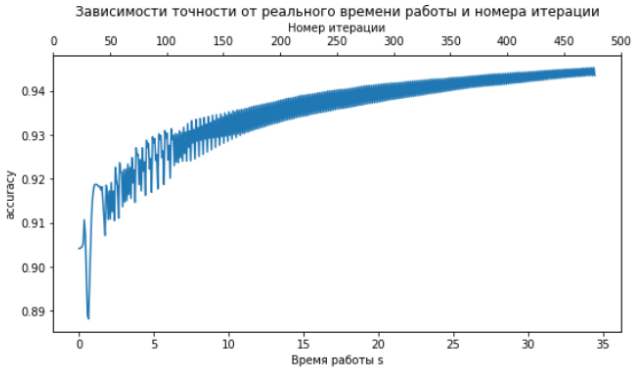
`step_alpha: 0.6, step_beta: 0.5`



Наконец, кривые стали совсем гладкими, что говорит об устойчивости модели и хорошей сходимости метода с данными параметрами.

```
step_alpha: 2.6, step_beta: 0.0
```

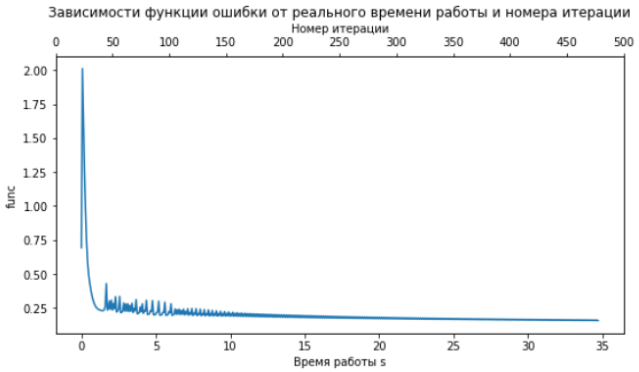


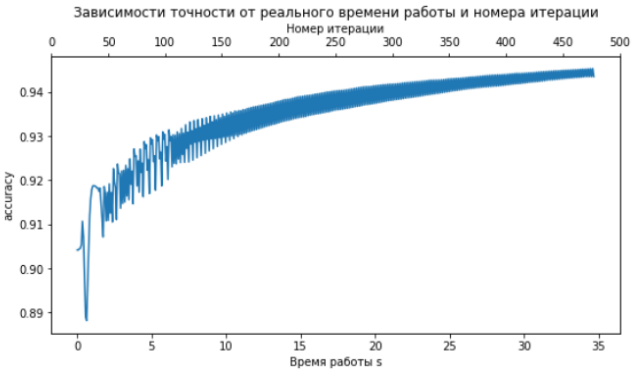


Данная модель реализует спуск без уменьшения шага, ведь $\frac{\alpha}{i^\beta} = \alpha$. И α служит осциллятором кривой ошибки и кривой качества, отвечая за амплитуду отклонений (ширину "штрихованной"линии).

4.2.2 Зависимость от начального приближения

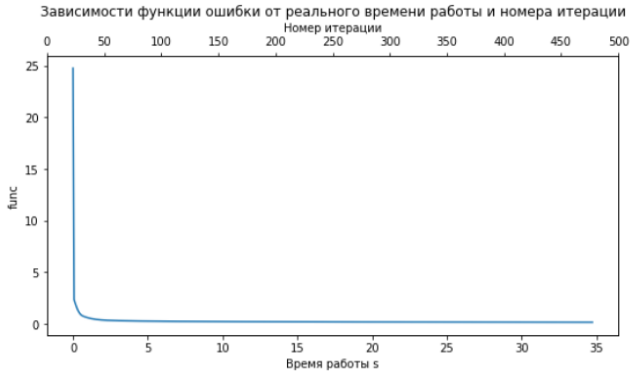
Шаги `step_alpha` и `step_beta` возьмем равными 2.6, 0.0 на протяжении всего эксперимента.

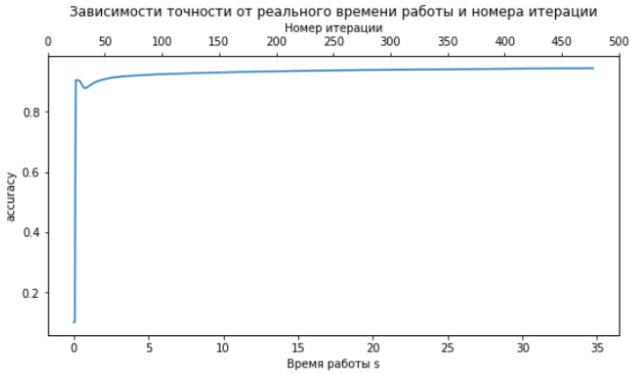




Из графика видно, что при нулевом начальном приближении значение функции потерь на первом шаге резко возрастает и только после этого скачка начинает убывать. Это свидетельствует о том, что начальное приближени находится достаточно далеко от точки оптимального старта для спуска.

Попробуем подобрать w_0 лучше. Возьмем хотя бы среднее арифметическое между нулевым приближением (из 1) и вектором весов, которые получили в предыдущем спуске.





Действительно, скачок функции ошибки исчез, что говорит о близости точки запуска к искомому минимуму. К тому же вся кривая стала гладкой, то ест ьприближение происходит равномерно. Более того, трудно не заметить, как резко возросла точность на первых шагах спуска и далее не опускалась. Вывод: точка запуска важна для хорошего спуска.

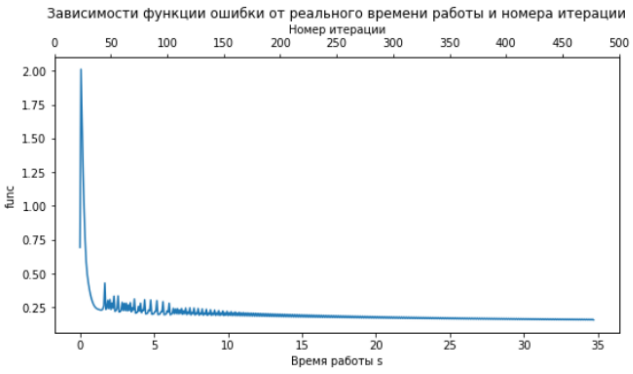
4.3 Исследование стохастического градиентного спуска

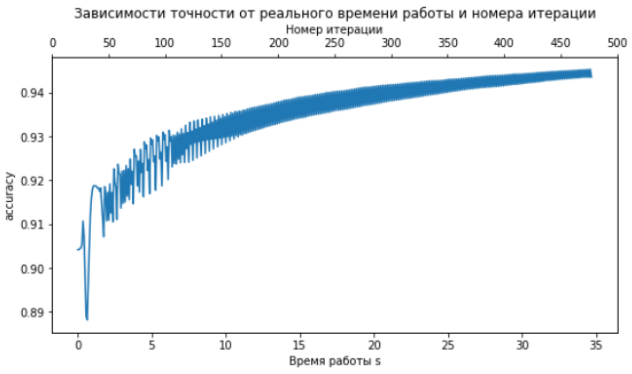
4.3.1 Зависимость от размера батча

Шаги `step_alpha` и `step_alpha` и начальное приближение w_o возьмем равными 2.6, 0.0, нулевое на протяжении всего эксперимента.

Приведем график для этих параметров у GDC:

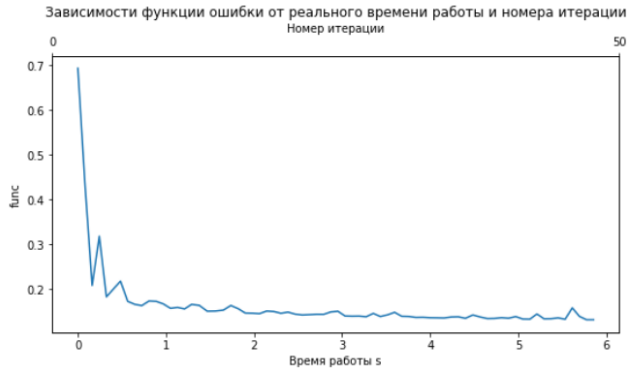
```
Wall time: 46.9 s
```





А теперь у SGDC с размером батча в 5000 объектов:

```
Wall time: 7.11 s
```

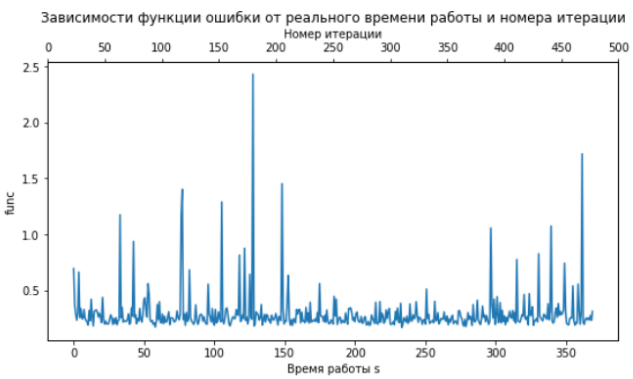


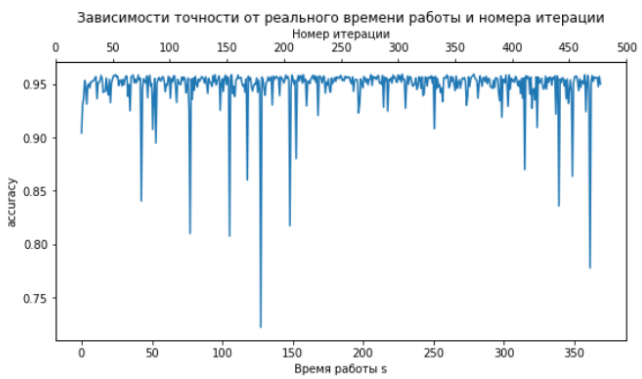
Видно, что модель стала устойчивее и спуск сходится плавнее, об этом говорит бОльшая гладкость кривых. Во всяком случае, нет таких резких колебаний вокруг истинного значения. Так происходит, потому что коэффициент градиента становится всё меньше и меньше с каждым батчем, реализуя уменьшающийся итерационный шаг. Однако убывает он не настолько быстро, чтобы прекратить спуск до достижения `max_iter`

```
Wall time: 3.96 s
```

Для 100 объектов кривая стала гораздо более аккуратной ломаной, что говорит о неком кусочно-равномерном приближении. Однако при уменьшении размера батча результат окажется таким:

```
Wall time: 6min 16s
```





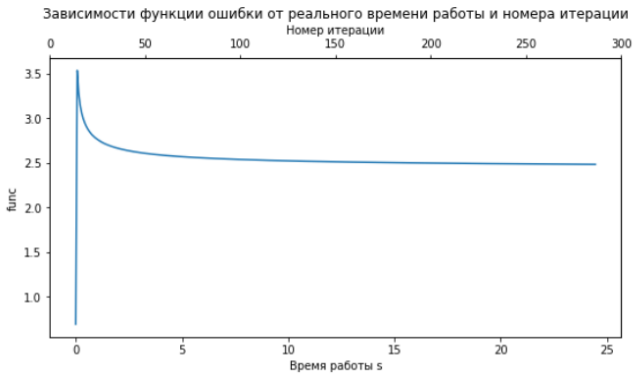
Так происходит из-за длительного перемешивания выборки на каждой итерации, а так же по причине использования внутри стохастического спуска не numpy-методы, а for - простые циклы, занимающие по сравнению с предыдущим гораздо больше времени.

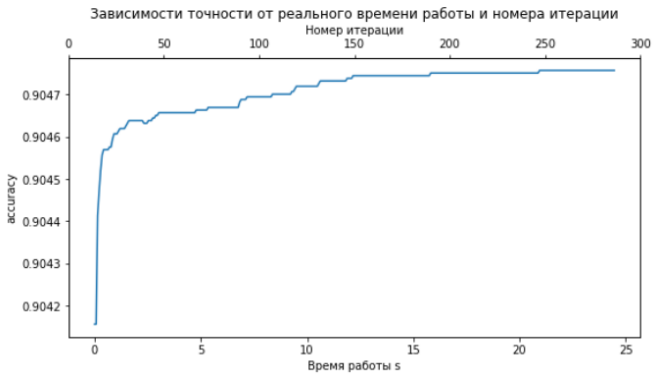
4.3.2 Зависимость от параметров шага

В сущности, стохастический градиентный спуск дает более гладкие кривые приближения по функции ошибок и точности, однако характер зависимости и вид графиков остается таким же, как и при градиентном спуске. Важно понимать, что коэффициент перед градиентом теперь зависит не только от параметров шага `step_alpha` и `step_alpha`, но и от размера батча. Потому зависимости получаются более гладкими. При увеличении размеров батча поведение стохастического градиентного спуска стремится к поведению обычного градиентного спуска, что логично, ведь если размер батча достигнет размеров выборки, то различий между стохастическим и обычным спуском не будет.

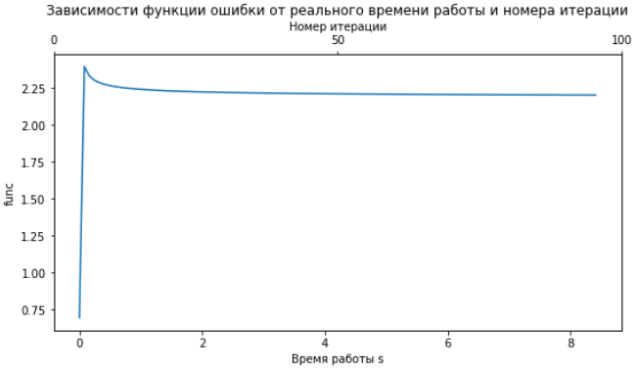
Зафиксируем размер батча на 5000 и визуализируем еще одну пару спусков для параметров шага:

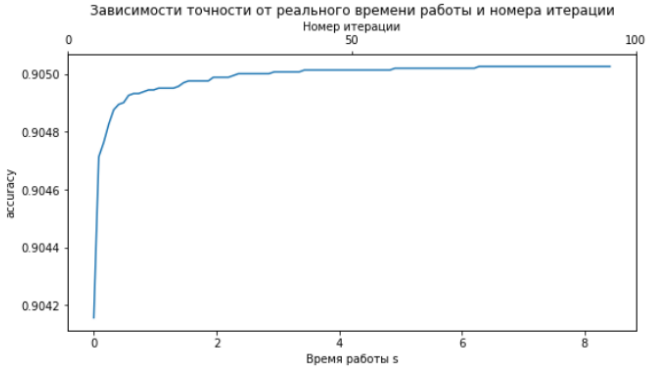
```
step_alpha: 4.6, step_beta: 1.5
```





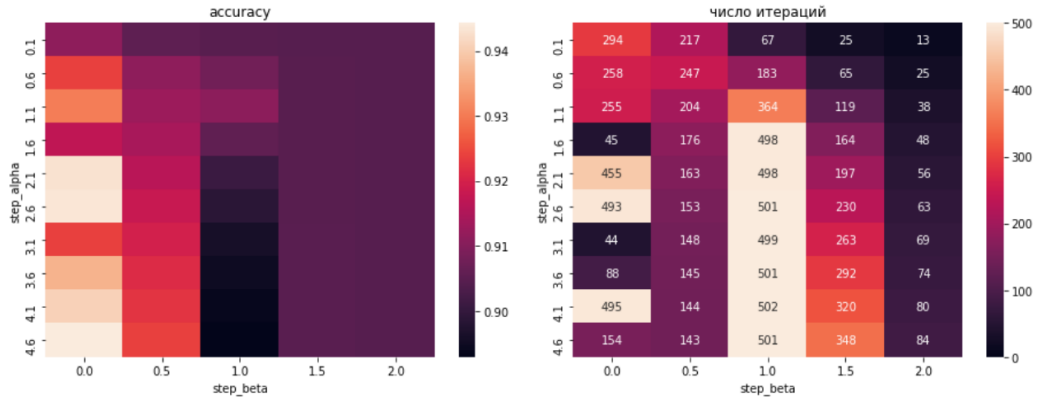
- обычный спуск



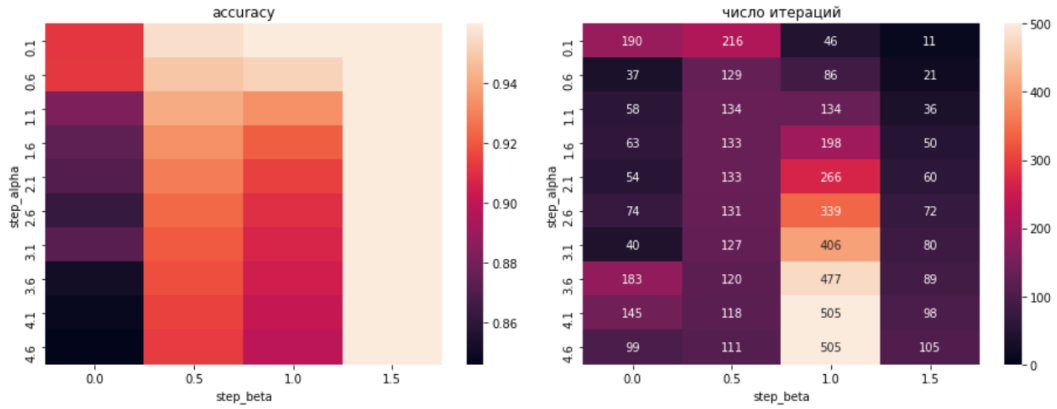


-стохастический

Как и утверждалось, кривые похожи, но стохастическая более гладкая. Убедимся, что помимо гладкости улучшается качество на обучающей выборке: Для сравнения, тепловые карты для обычного спуска:



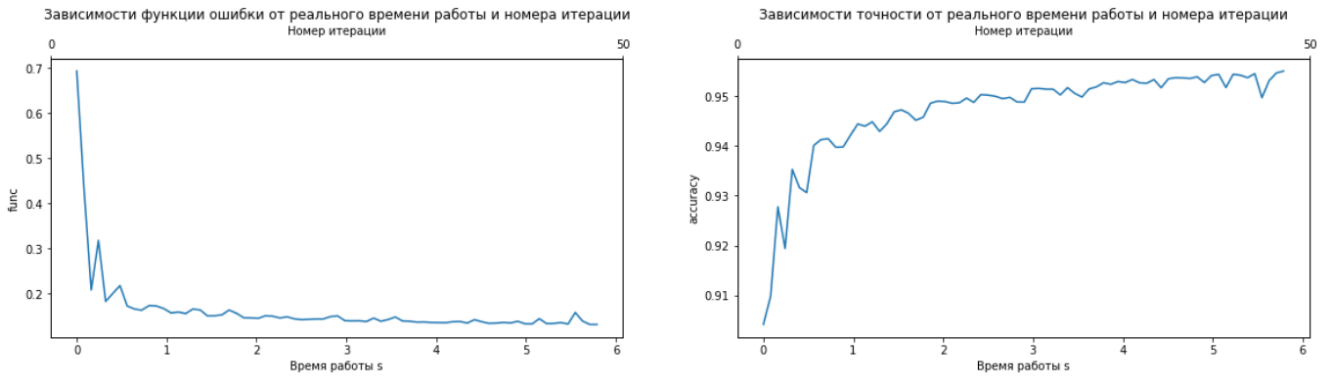
Для стохастического спуска:



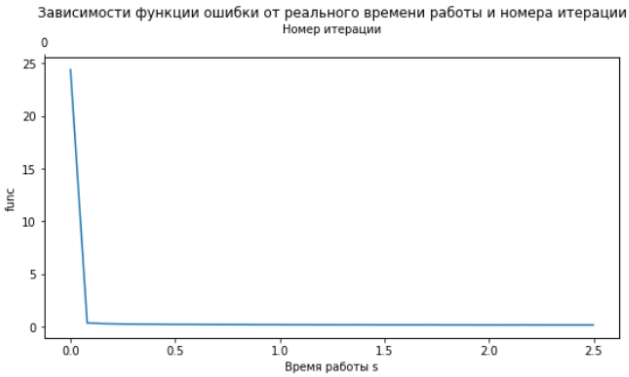
Видно, что карта стала существенно ярче, то есть общая точность метода повысилась.

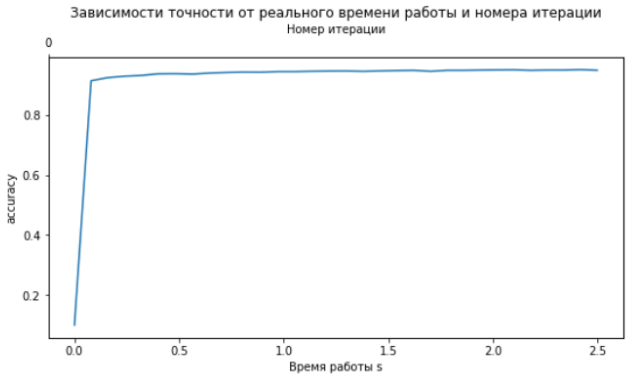
4.3.3 Зависимость от начального приближения

Поскольку суть метода спуска не меняется, стохастический спуск так же чувствителен к начальному приближению. Убедимся в этомб зафиксировав размер батча и шаги.
Начальное приближение 0:



Начальное приближение w = среднее арифметическое между нулевым приближением (из 1) и вектором весов, которые получили в предыдущем спуске.





Как видим, кривая снова стала более линейной (менее ломаной), что свидетельствует о большей равномерности подхода к минимуму с лучшим значением запуска w_0

4.4 Сравнение GD и SGD

Основываясь на предыдущих результатах, стохастический градиентный спуск показал себя более гибким, нежели классический градиентный спуск. Обобычный спуск является частным случаем стохастического при размере батча = размеру выборки. При небольших размерах батча стохастический спуск, несмотря на увеличение времени показывает лучшие результаты, нежели обычный, кривые всегда более линейны, с меньшим числом изломов. То есть можно сказать, что метод стабильнее. К тому же заметим, что при использовании стохастического спуска нет необходимости хранить всю обучающую выборку в ОП. Это большое преимущество при работе с большими данными или дорогой памятью.

4.5 Лемматизация

Лемматизация - преобразование входного текста, такое что каждое слово приводится к своей нормальной форме. Для этого используется библиотека nltk и лексическая база данных wordnet. После того как все комментарии приведены к нормальной форме повторяется процедура с Count.Vectorize. Поскольку объем выборки очень велик, процесс леммматизациии занимает много времени. Семплируем датасет и вместо 160 тысяч оставим случайные 10%.

размерность векторного пространства после приведения к разреженному виду

`min_df`	без лемматизации	с лемматизацией	%снижения
1	45903	39093	15.0%
0.0001	18821	15167	19.4%

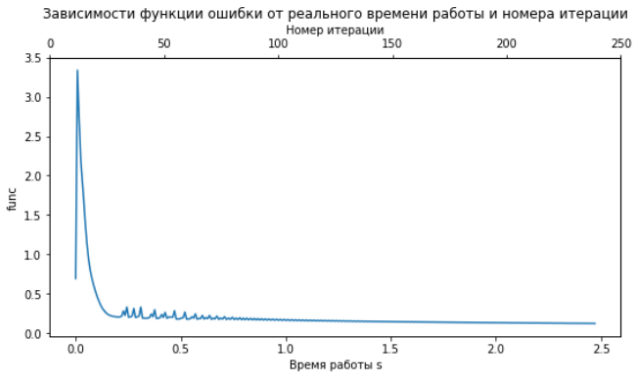
Таким образом, лемматизация вкупе с `min_df` = 0.0001 хорошо понижает размерность. Так происходит, потому что после лемматизации слова, имеющие разную форму одного и того же нормального слова, будут сгруппированы в один признак.

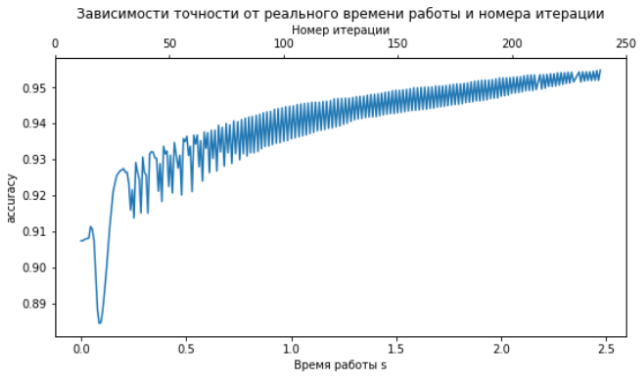
accuracy логистической регрессии с градиентным спуском

выборка	без лемматизации	с лемматизацией
train	0.955	0.963
test	0.905	0.885

Качетсво алгоритма заметно улучшилось для тренировочной выборки, и уменьшилось для тестовой. Посмотрим на графики функции ошибки и точности для градиентного спуска без и с лемматизацией:

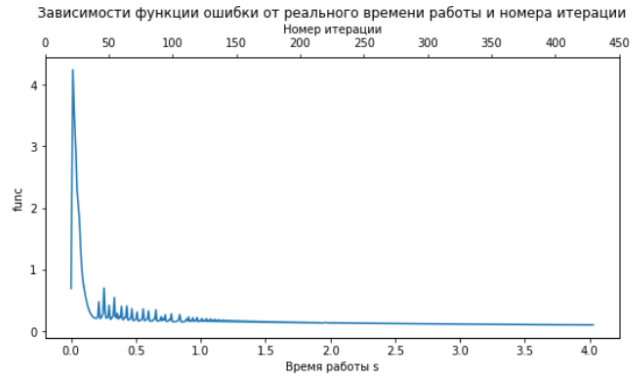
```
accuracy for train: 0.955 accuracy for test: 0.905
CPU times: user 14.7 s, sys: 39.7 s, total: 54.4 s
Wall time: 3.45 s
```

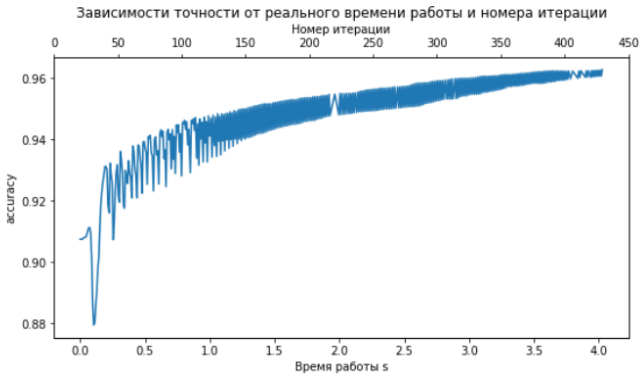




- без

```
accuracy for train: 0.963 accuracy for test: 0.885
CPU times: user 23.7 s, sys: 1min 3s, total: 1min 27s
Wall time: 5.54 s
```

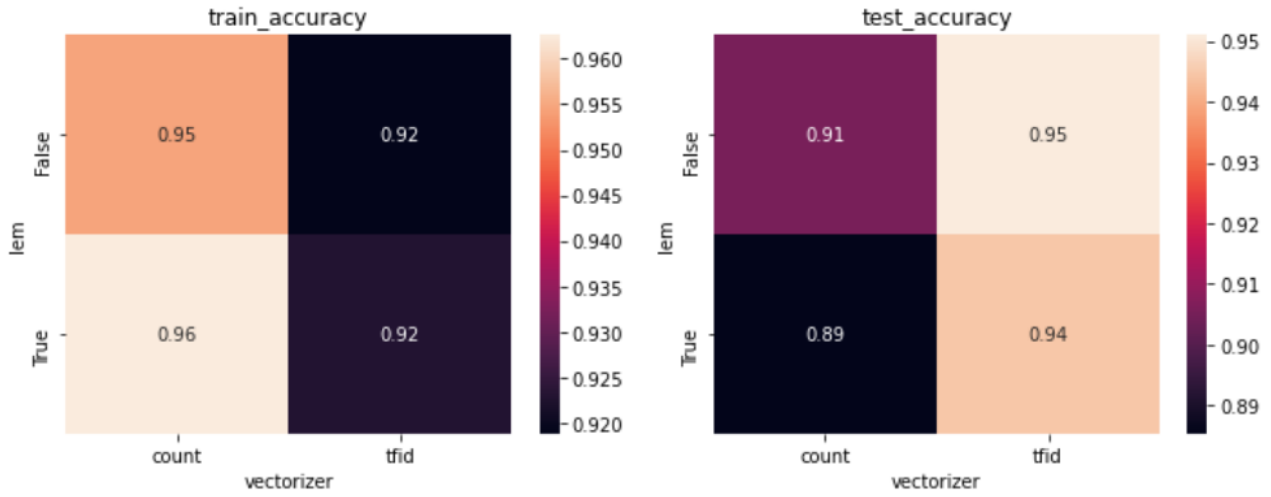




- с лемматизацией Помимо возросшего качества стоит отметить увеличение числа итераций и, слдеовательно, лучшую сходимость.

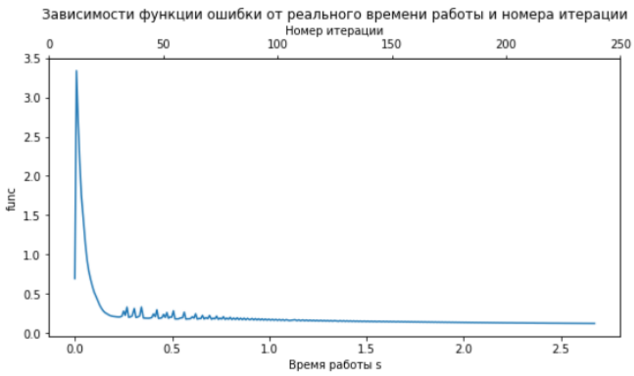
4.6 Bag of words или TF-IDF

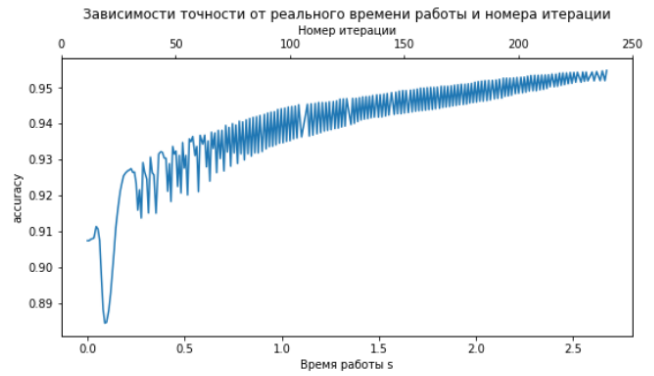
Классическое кодирование текстов Bag of words: в i-той позиции признакового вектора стоит число раз, которое слово i встретилось в данном комментарии. Кодирование TF-IDF учитывает популярность слова во всем корпусе. Чем популярнее слово, тем, что логичнее, меньшую смысловую нагрузку оно несет. Как вводные слова или частицы.
Взглянем на итоговую карту точностей для 4 комбинаций градиентных спусков: с лемматизацией и без, методом Count.Vectorize и TfidfVectorizer:

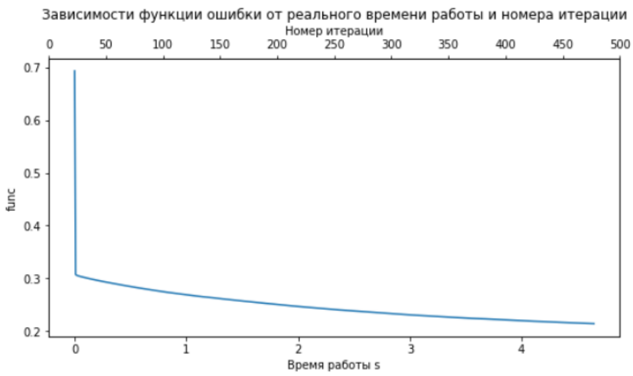


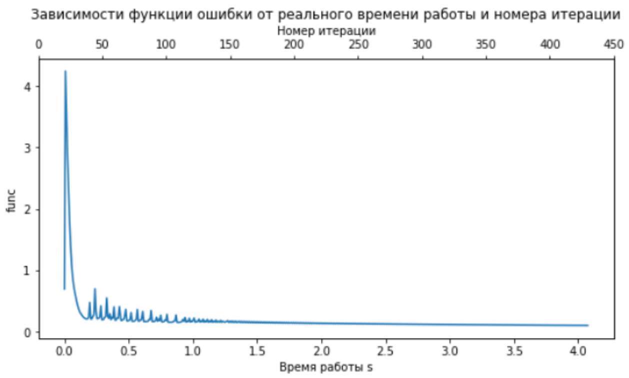
Видно, что применение TfidfVectorizer дает меньшую точность на обучающей выборке, но выигрывает на тестовой.

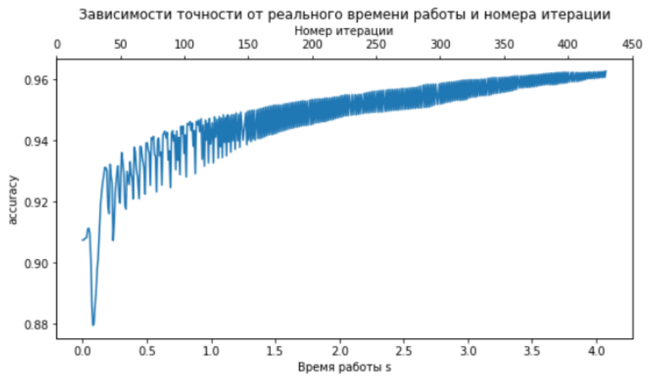
Посмотрим на графики зависимости функции ошибки и точности от применения лемматизации и использования TfidfVectorizer:
vectorizer = 'count': Count.Vectorize, 'tdif': TfidfVectorizer
lem = True: применяется лемматизация, False: не применяется
Графики:
1) Без лемматизации, Count.Vectorize
1) Без лемматизации, TfidfVectorizer
1) С лемматизацией, Count.Vectorize
1) Без лемматизацией, TfidfVectorizer

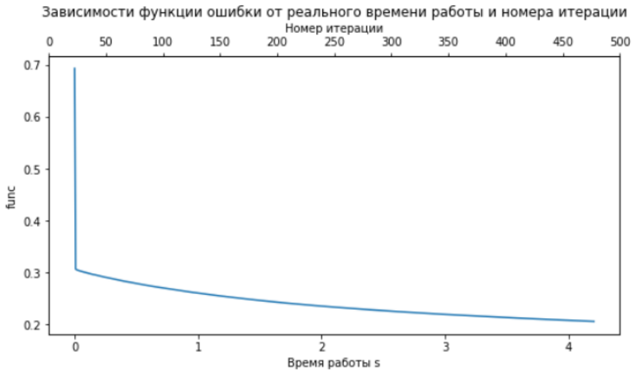








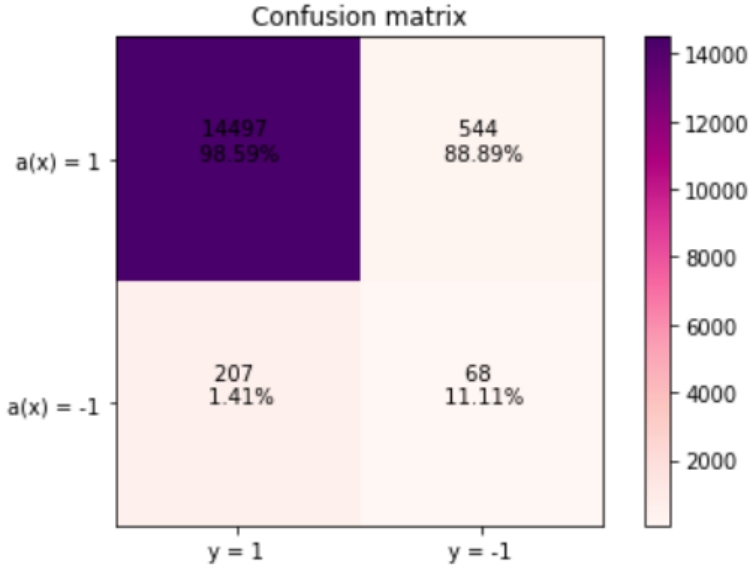




4.7 Анализ ошибок

В завершение применим наилучший алгоритм, дающий наибольшую точность на тестовой выборке: `step_alpha` $= 5.0$, `step_alpha` $= 0.1$ без начального приближения w_o с выключенной лемматизацией, методом TfidfVectorizer. Качество $= 0.95$

Визуализируем матрицу ошибок




```
precision: 0.247, recall: 0.111
```