Московский Государственный Университет имени М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Математических Методов Прогнозирования

Практикум осень 2022
Градиентные методы обучения линейных моделей. Применение линейных моделей для определения токсичности комментария. Отчет GD

Авдеев Роман Артемович
Группа 317

Ноябрь, 2022

Оглавление

1 Введение

В данной работе была поставлена задача:

1. реализовать на языке Python модель линейной логистической регрессии с градиентными методами оптимизации
2. обосновать используемые формулы/методы

Немного о градиентных методах оптимизации:

Стохастический градиентный спуск относится к оптимизационным алгоритмам и нередко используется для настройки параметров модели машинного обучения.

При стандартном градиентном спуске для корректировки параметров модели используется градиент. Вектор параметров изменяется в направлении антиградиента с заданным шагом, поэтому стандартному градиентному спуску требуется один проход по обучающим данным до того, как он сможет менять параметры.

При стохастическом градиентном спуске значение градиента аппроксимируются градиентом функции стоимости, вычисленном только на одном элементе обучения. Затем параметры изменяются пропорционально приближенному градиенту. Таким образом, параметры модели изменяются после каждого объекта обучения. Важным преимуществом стохастического градиентного спуска является то, что для больших массивов данных он может дать значительное преимущество в скорости по сравнению со стандартным градиентным спуском.

2 Пояснения к задаче

Некоторые формулы, используемые в решении:

Объект $\nabla f(X)$ - вектор для функции векторного агумента и матрица для функции матричного аргумента - называется градиентом

$gradf(X) = \nabla f(X) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, ..., \frac{\partial f}{\partial x_n})$, где $X = (x_1, ..., x_n)$

$df(X) =< \nabla f(X), dX >$

3 Теоретическая часть

3.1 Формула градиента функции потерь для задачи бинарной логистической регрессии

$w \in R^d, x_i \in R^d$

$a(x) = sign(< w, x > +w_0)$; Полагаем $w_0 = 0$

logloss: $L(M_i) = ln(1 + e^{-M_i})$, где $M_i(w) = y_i < w, x_i >$

$\sigma(< w, x >= \frac{1}{1+e^{-<w,x>}}$

Представим L в виде $L_i = -ln(\sigma(y_i < w, x_i >)) = ln(1 + e^{-y_i<w,x_i>})$

Тогда $\frac{\partial L_i}{\partial w_k} = \frac{1}{1+e^{-y_i<_k w,x_{ik}>}} * e^{-y_i<w_k,x_{ik}>} * (-y_i * x_{ik}) = \sigma(y_i < w_k, x_{ik} >) * \frac{1-\sigma(y_i<w_k,x_{ik}>)}{\sigma(y_i<w_k,x_{ik}>)} * (-y_i * x_{ik}) = -(1 - \sigma(y_i < w_k, x_{ik} >) * y_i * x_{ik}$

Значит, $\frac{\partial L}{\partial w_k} = \sum_{i=1}^{d} L_i = -\sum_{i=1}^{d} (1 - \sigma(y_i < w_k, x_{ik} >) * y_i * x_{ik})$

Учитываем нормировку $=> \frac{1}{d} \sum_{i=1}^{d} L_i = -\frac{1}{d} \sum_{i=1}^{d} (1 - \sigma(y_i < w_k, x_{ik} >) * y_i * x_{ik})$

Добавляем регуляризатор: $\frac{\lambda}{2} * \|\omega\|^2$;

Посчитаем $d(\frac{\lambda}{2} * \|\omega\|^2) = \frac{\lambda}{2} * d(< w, w >) = \lambda w^T dw$

Тогда финальная формула для градиента: $\nabla L = -\frac{1}{d} \sum_{i=1}^{d} (1 - \sigma(y_i < w_k, x_{ik} >) * y_i * x_{ik}) + \lambda {w_k}^T$

3.2 Формула градиента функции потерь для задачи многоклассовой (мультиномиальной) логистической регрессии

Опустим регуляризатор, тогда функция потерь представляется как $L(X, w) = -\frac{1}{d} \sum_{i=1}^{d} ln(\frac{e^{<w_{yi}, x_i>}}{\sum_{k=1}^{K} e^{<w_k, x_i>}})$

Представим $ln(\frac{e^{<w_{yi}, x_i>}}{\sum_{k=1}^{K} e^{<w_k, x_i>}})$ как разность логарифмов, т.е. $ln(e^{<w_{yi}, x_i>}) - ln(\sum_{k=1}^{K} e^{<w_k, x_i>})$

$\frac{\partial L}{\partial w_k} = -\frac{1}{d} \frac{\partial}{\partial w_k} (\sum_{i=1}^{d} (ln(e^{<w_{yi}, x_i>}) - ln(\sum_{k=1}^{K} e^{<w_k, x_i>})) = -\frac{1}{d} \frac{\partial}{\partial w_k} (\sum_{i=1}^{d} (< w_{yi}, x_i > - ln(\sum_{k=1}^{K} e^{<w_k, x_i>})) =$

$= -\frac{1}{d} (\sum_{i=1}^{d} (x_i * [y_i == j] - ln(e^{<w_{yi}, x_i>}) - \frac{\nabla e^{<w_{yi}, x_i>}}{\sum_{k=1}^{K} e^{<w_k, x_i>}})) = -\frac{1}{d} (\sum_{i=1}^{d} ([y_i == j](x_i - \frac{e^{<w_{yi}, x_i>}}{\sum_{k=1}^{K} e^{<w_k, x_i>}} x_i))) =$

$= -\frac{1}{d} \sum_{i=1}^{d} ([y_i == j](\mathbf{1} - \frac{e^{<w_{yi}, x_i>}}{\sum_{k=1}^{K} e^{<w_k, x_i>}}) x_i)$

Получили $\nabla_w j L = -\frac{1}{d} \sum_{i=1}^{d} ([y_i == j](\mathbf{1} - \frac{e^{<w_{yi}, x_i>}}{\sum_{k=1}^{K} e^{<w_k, x_i>}}) x_i)$, $\forall j = 1, ..., K$

3.3 Сводим задачу мультиномиальной логистической регрессии к бинарной логистической регрессии

Теперь сведем полученную выше формулу к бинарной задаче:

$K = 2 => -\frac{1}{d} \sum_{i=1}^{d} ([y_i == j](\mathbf{1} - \frac{e^{<w_{yi}, x_i>}}{e^{<w_j, x_i>} + e^{<w_p, x_i>}}) x_i)$,

где p и j в знаменателе стоит понимать как метки $+1$ и -1

$-\frac{1}{d} \sum_{i=1}^{d} ([y_i == j](\mathbf{1} - \frac{e^{<w_j, x_i>}}{e^{<w_j, x_i>} + e^{<w_p, x_i>}}) x_i) = -\frac{1}{d} \sum_{i=1}^{d} ((\mathbf{1} - \frac{1}{1 + \frac{e^{<w_p, x_i>}}{e^{<w_j, x_i>}}}) x_i y_i) =$

$= -\frac{1}{d} \sum_{i=1}^{d} ((\mathbf{1} - \frac{1}{1 + e^{<w_p - w_j, x_i>}}) x_i y_i) = -\frac{1}{d} \sum_{i=1}^{d} ((\mathbf{1} - \frac{1}{1 + e^{<w_s, x_i>}}) x_i y_i) = -\frac{1}{d} \sum_{i=1}^{d} (1 - \sigma(y_i < w_s, x_{is} >) * y_i * x_{is})$

ч.т.д

4 Практическая часть

4.1 Эксперимент 1 + Эксперимент 2

В исходных данных присутствовали различные символы помимо букв и цифр, поэтому, используя библиотеку re и реуглярные выражения, очистил тексты.

Были использованы следующие регулярные выражения:

1. `[\W_]+` - оставляем только буквы и цифры
2. `\s+` - находим один или несколько пробелов, чтобы далее поменять на одиночный(с помощью re.sub())

А также .lower() для приведения к нижнему регистру и .strip() для удаления лишних пробелов в начале и конце строки.

Для преобразования выборки в разреженную матрицу использовал CountVectorizer(), подобрав `min_df` и `max_df` для сокращения числа признаков до разумного.

Также в данном блоке преобразовал ответы к столбцу из (-1) - False и (+1) - True.

4.2 Эксперимент 3

Для сравнения численного подсчета градиента функции потерь из модуля utils.py с вычислением по аналитической формуле сгенерировал выборку из 10000 элементов и 5000 признаков, а также ответы (+1 и -1) к ним. Веса также были сгенерированы случайным образом.
При подсчете градиента функции потерь из модуля utils.py взял $\epsilon = 10^{-3}$

Порядок разницы между численными значениями градиента составил 10^{-11}

Точное значение среднего расхождения $= 1.0704334796329818 * 10^{-11}$

4.3 Эксперимент 4

В данном эксперименте исследовал поведение градиентного спуска для задачи логистической регрессии в зависимости от следующих параметров:

1. параметр размера шага `step_alpha`
2. параметр размера шага `step_beta`
3. начального приближения

4.3.1 Подбор α и β (1-ый этап)

Данные параметры нам нужны для задания темпа обучения(learning rate) $\eta_k = \frac{\alpha}{k^\beta}$

Сначала был проведен подбор α и β по большой сетке со значениями от 0.1 до 4 с шагом в 0.1. Итого 1600 комбинаций параметров. Исследование проводилось при `max_iter` $= 500$

Первой была проанализирована точность.

Здесь рассматривались две стратегии:

1. выбор лучшей комбинации на основе максимального среднего accuracy по итерациям
2. выбор лучшей комбинации на основе максимального accuracy, достигаемого среди итераций

По каждой из стратегий были выбраны 5 лучших комбинаций.

Для первой стратегии лучшими оказались:

1. $\alpha = 3.0$ и $\beta = 0.6$
2. $\alpha = 2.3$ и $\beta = 0.6$
3. $\alpha = 3.3$ и $\beta = 0.6$
4. $\alpha = 2.5$ и $\beta = 0.6$
5. $\alpha = 2.4$ и $\beta = 0.6$

Оценка точности для данных комбинаций (Рис.1):

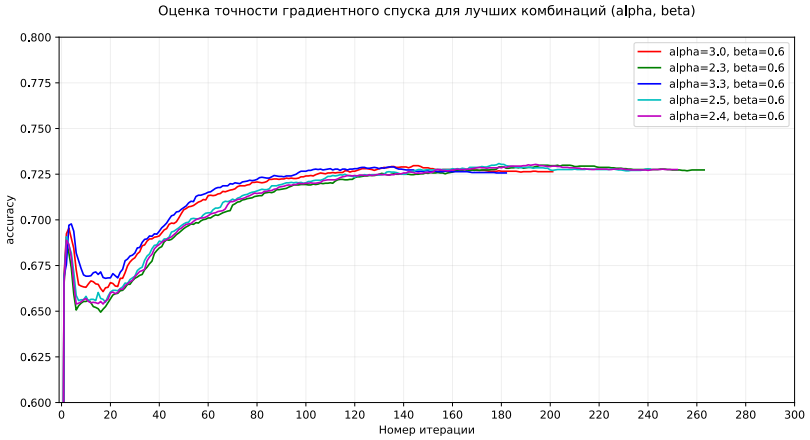
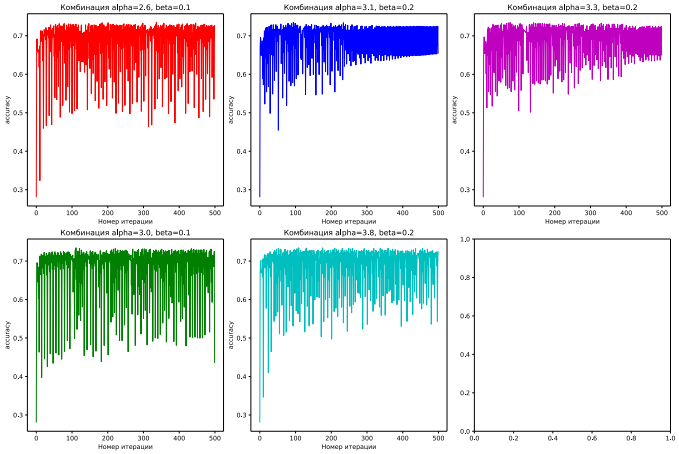


Рис.1

Для второй стратегии лучшими оказались:

1. $\alpha = 2.6$ и $\beta = 0.1$
2. $\alpha = 3.0$ и $\beta = 0.1$
3. $\alpha = 3.1$ и $\beta = 0.2$
4. $\alpha = 3.8$ и $\beta = 0.2$
5. $\alpha = 3.3$ и $\beta = 0.2$

Оценка точности для данных комбинаций (Рис. ниже):



Но наблюдаем сильную осцилляцию, значит, подобраны не лучшие α и β. Поэтому будем считать первую стратегию более удачной.

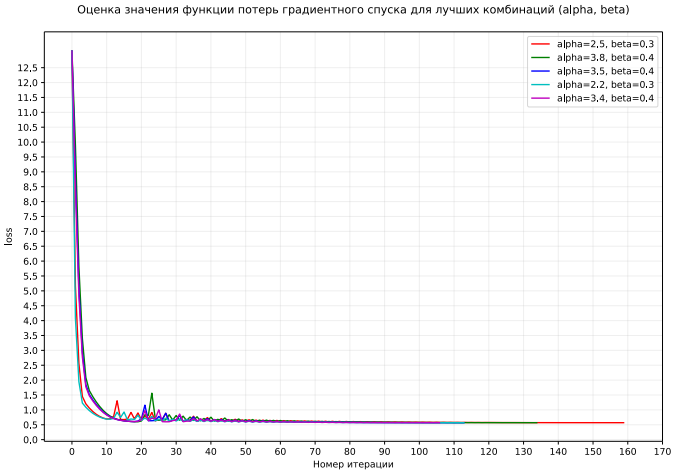
Далее были проанализированы значения функции потерь.

Здесь осуществлялся поиск комбинаций α и β с наименьшим возможным loss на итерациях.

Такими комбинациями оказались:

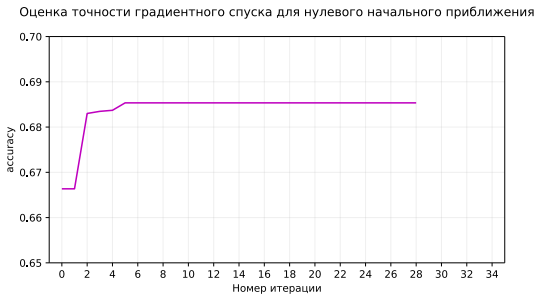
1. $\alpha = 2.5$ и $\beta = 0.3$
2. $\alpha = 3.8$ и $\beta = 0.4$
3. $\alpha = 3.5$ и $\beta = 0.4$
4. $\alpha = 2.2$ и $\beta = 0.3$
5. $\alpha = 3.4$ и $\beta = 0.4$

Оценка значений функции потерь для данных комбинаций:



4.3.2 Подбор начального приближения (1-ый этап)

Здесь рассматривалась сетка $[-1, 1]$ с шагом в 0.1.
Формировались начальные приближения вида $[-1, ..., -1], [-0.9, ..., -0.9], ..., [1, ..., 1]$
Тестирование проводилось на α и β, указанных по умолчанию. Лучшим оказалось нулевое приближение.
График точности для начального приближения:



Поэтому для 2-го этапа было решено использовать именно нулевое начальное приближение при подборе α и β (уже по новой сетке).

Главная задача первого "грубого" подбора достигнута - нашли пределы, в которых стоит рассмотреть α и β подробнее(с меньшим шагом сетки), а также определились с начальным приближением.

4.3.3 Подбор α и β (2-ой этап)

На втором этапе подбора были взяты $\alpha \in [0.1, 4)$ с шагом в 0.08 и $\beta \in [0.1, 0.6)$ с шагом в 0.08.

Дизайн эксперимента остался таким же. Итак, были получены лучшие комбинации α и β:

Первая стратегия:

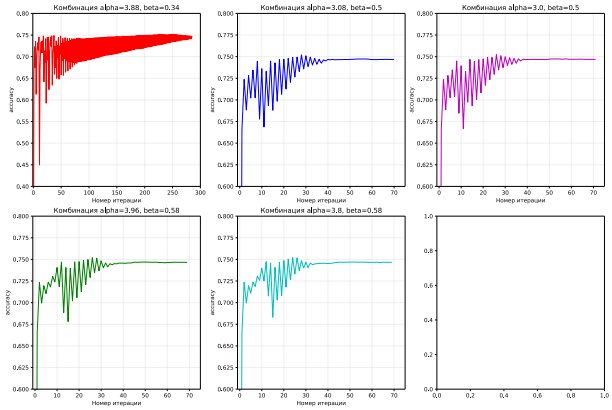
1. $\alpha = 1.32$ и $\beta = 0.58$
2. $\alpha = 1.24$ и $\beta = 0.58$
3. $\alpha = 1.16$ и $\beta = 0.58$
4. $\alpha = 1.08$ и $\beta = 0.58$
5. $\alpha = 1.0$ и $\beta = 0.5$

Оценка точности для первой стратегии:

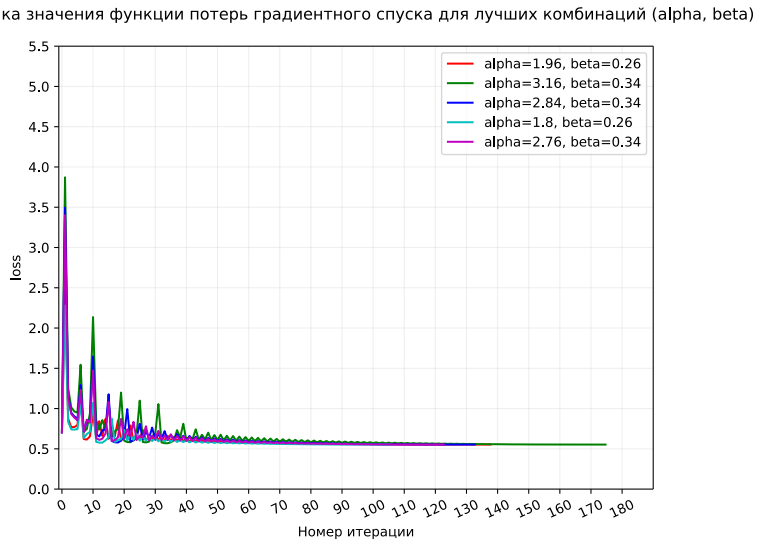
Вторая стратегия:

1. $\alpha = 3.88$ и $\beta = 0.34$
2. $\alpha = 3.08$ и $\beta = 0.5$
3. $\alpha = 3.0$ и $\beta = 0.5$
4. $\alpha = 3.96$ и $\beta = 0.58$
5. $\alpha = 3.8$ и $\beta = 0.58$

Оценка точности для второй стратегии:



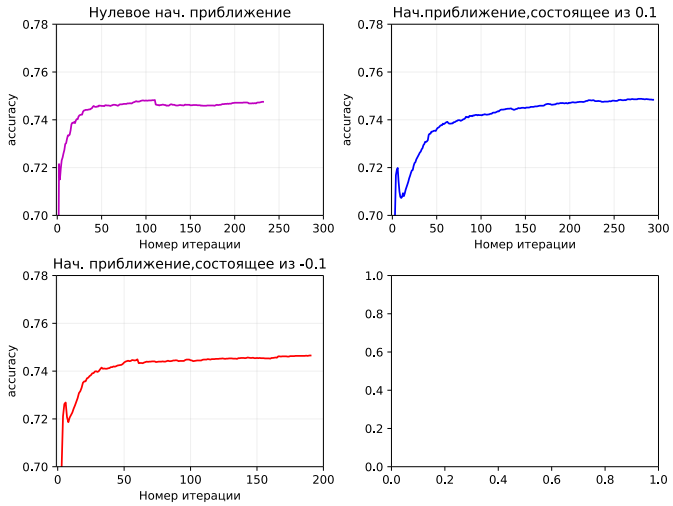
Оценка значений функции потерь для указанных комбинаций:



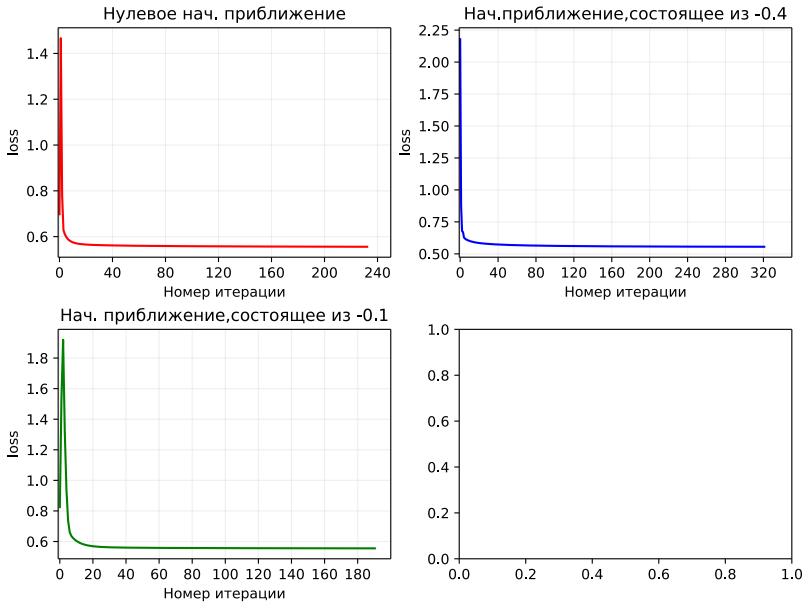
4.3.4 Подбор начального приближения (2-ой этап)

На втором этапе анализировались начальные приближения на лучших комбинациях α и β, подобранных на 1ом этапе.

Итак, для той же сетки подбора весов при $\alpha = 1.08$ и $\beta = 0.58$ получили 3 приближения, дающие наивысшую точность(по первой стратегии):



Также оценим значения функции потерь для лучших начальных приближений:



Видим, что некоторые начальные приближения дают более высокие значения точности, но более низкие показатели в функции потерь. Единственным приближением, которое входит в "топ" и по точности, и по значениям функции потерь, является нулевое. Поэтому будем считать его наиболее подходящим и использовать в следующих экспериментах.

4.4 Эксперимент 5

В данном эксперименте исследовал поведение стохастического градиентного спуска для задачи логистической регрессии в зависимости от следующих параметров:

1. параметр размера шага `step_alpha`
2. параметр размера шага `step_beta`
3. начального приближения
4. размер подвыборки `batch_size`

4.4.1 Подбор α и β

В этом эксперименте действовал, как и в предыдущем, рассматривая две стратегии оценки лучших значений точности для комбинаций α и β:

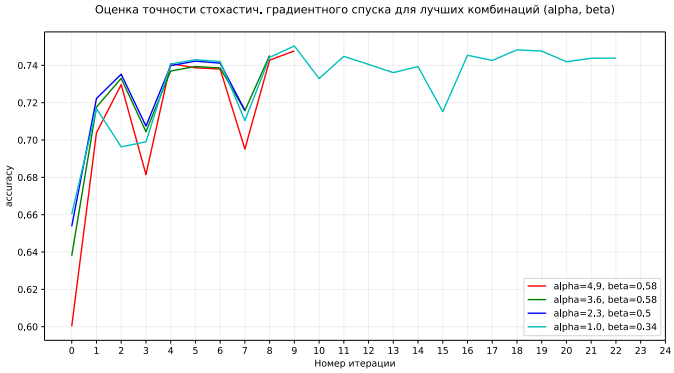
1. выбор лучшей комбинации на основе максимального среднего accuracy по итерациям
2. выбор лучшей комбинации на основе максимального accuracy, достигаемого среди итераций

Был проведен подбор α и β по сетке со значениями от 1 до 5 с шагом в 1.3 для α и от 0.1 до 0.6 с шагом в 0.08 для β. Исследование проводилось при `max_iter` $= 500$.

Рассмотрим точность моделей.

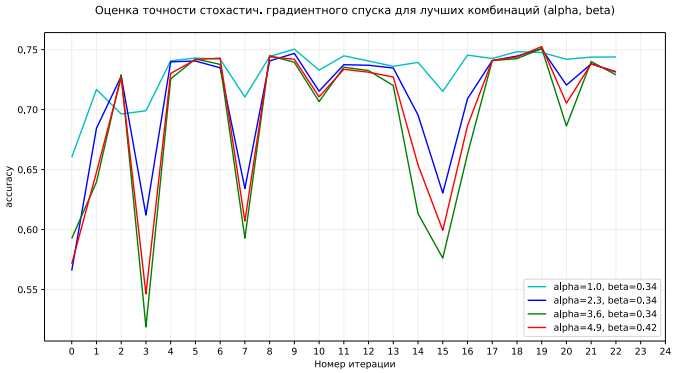
Для первой стратегии лучшими оказались:

1. $\alpha = 4.9$ и $\beta = 0.58$
2. $\alpha = 3.6$ и $\beta = 0.58$
3. $\alpha = 2.3$ и $\beta = 0.5$
4. $\alpha = 1.0$ и $\beta = 0.34$



Для второй стратегии:

1. $\alpha = 1.0$ и $\beta = 0.34$
2. $\alpha = 2.3$ и $\beta = 0.34$
3. $\alpha = 3.6$ и $\beta = 0.34$
4. $\alpha = 4.9$ и $\beta = 0.42$



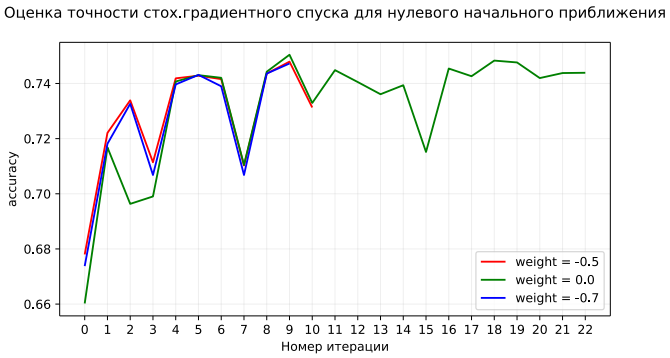
Заметим, что для каждого из двух подходов лучшей комбинацией является $\alpha = 1.0$ и $\beta = 0.34$.

Рассмотрим значения функции потерь:

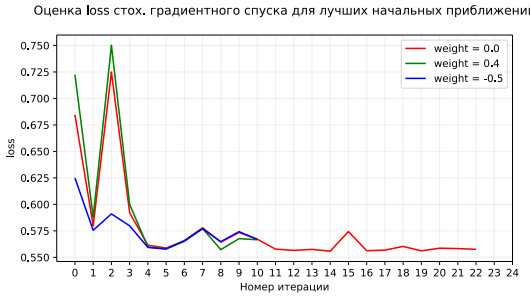
Получили, что комбинация $\alpha = 1.0$ и $\beta = 0.34$ демонстрирует не только наивысшие показатели точности, но и наименьшие значения функции потерь. Тогда зафиксируем эту комбинацию и будем использовать при подборе начального приближения и размера подвыборки.

4.4.2 Подбор начального приближения

Возьмем такую же сетку, как и в эксперименте 4: $[-1, 1]$ с шагом в 0.1.
При $\alpha = 1.08$ и $\beta = 0.34$ получили 3 приближения, дающие наивысшую точность(по первой стратегии):



Также оценим значения функции потерь для лучших начальных приближений:



Видим, что некоторые начальные приближения дают более высокие значения точности, но более низкие показатели в функции потерь. Единственным приближением, которое входит в "топ" и по точности, и по значениям функции потерь, является нулевое. Поэтому будем считать его наиболее подходящим и использовать в следующих экспериментах.

4.4.3 Подбор размера подвыборки

Рассмотрим три варианта батчей: 100, 400, 600. График точности в зависимости от номера эпохи и размера батча:

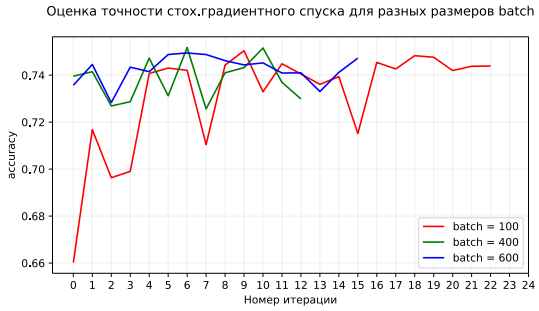


График функции потерь в зависимости от номера эпохи и размера батча:

Видим, что батч размера 100 дает более высокую точность, а также наименьший loss.

Значит, `batch_size`=100 наиболее подходящий.

4.5 Эксперимент 6

Из экспериментов 4 и 5 видим, что стандартный градиентный и стохастический градиентный спуски отличаются друг от друга.

Разница в качестве хоть и небольшая, но SGD превосходит GD. Также SGD быстрее, так как он работает не со всей выборкой, а с фрагментом. Плюсом же GD является устойчивость, а также лучшая сходимость, чего нет у SGD(т.к. на каждом шаге у нас новая подвыборка). Это можно заметить на графиках в экспериментах 4 и 5. У GD нет таких резких выбросов функции потерь, а также более плавные графики сходимости к лучшей точности.

4.6 Эксперимент 7

В данном эксперименте исследуется влияние преобработки текста на точность, время работы алгоритма и размер признакового пространства.

Были рассмотрены 4 стратегии предобработки:

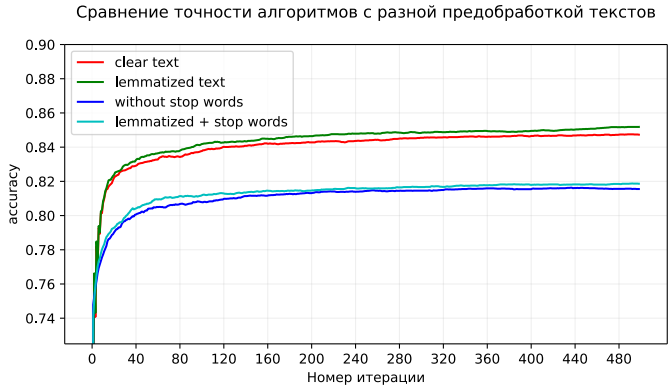
1. исходные данные, очищенные от всех символов, отличных от букв и цифр
2. лемматизация

3. удаление стоп-слов из исходных очищенных данных

4. удаление стоп-слов из лемматизированного текста

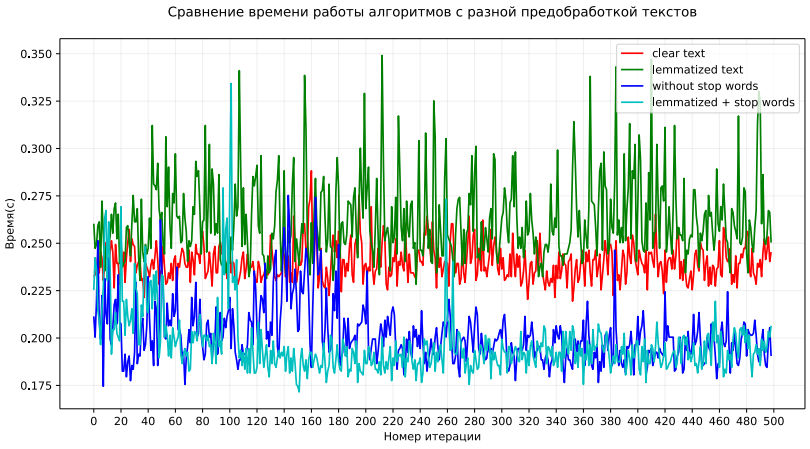
Для лемматизации текста и удаления стоп-слов была использована библиотека nltk (WordNetLemmatizer, stopwords, Tokenizer).

Сравним точность стратегий:



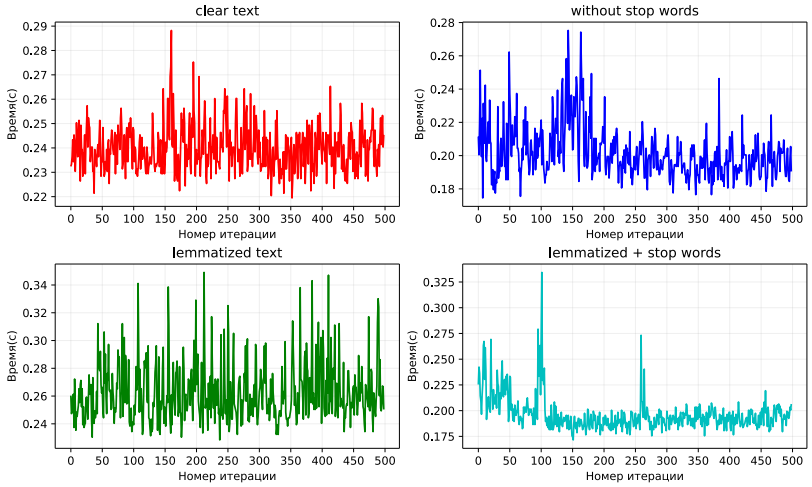
Заметим, что наивысшее качество демонстрирует лемматизированный текст.

Сравним время работы алгоритма для каждой из стратегий:



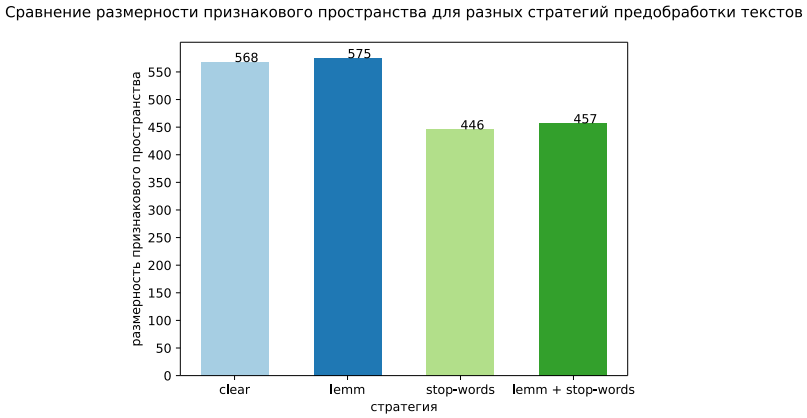
Видим, что лемматизация увеличивает время работы алгоритма.

Для наглядности рассмотрим 4 стратегии по отдельности:



Думаю, это связано с размерностью признакового пространства(рассмотрим далее). Лемматизация разделяет некоторые слова на 2 слова(например, слова с апострофом), что увеличивает число признаков. Стоп-слова наоборот уменьшают признаковое пространства, т.к. удаляют наиболее частые и непоказательные токены.

Теперь рассмотрим размерности признакового пространства подробнее:



Видим, что гипотеза была верна. И наибольшее число признаков именно у лемматизированного текста.

Также важно заметить, что данный эксперимент выполнялся при помощи градиентного спуска с нулевым начальным приближением, $\alpha = 1.0$ и $\beta = 0.34$.

4.7 Эксперимент 8

В этом эксперименте была исследована зависимость качества, времени работы алгоритма и размера признакового пространства в зависимости от используемого представления (BagOfWords или TfIdf) и параметров `max_df`, `min_df`.

Была выбрана сетка $[0.0, 0.5]$ с шагом 0.01 для `min_df` и $[0.05, 1]$ с шагом 0.1 для `max_df`

Выбраны именно такие множества рассматриваемых значений, т.к. в противном случае (при больших `min_df` и малых `max_df`) векторайзер не найдет токенов, и мы получим ValueError.

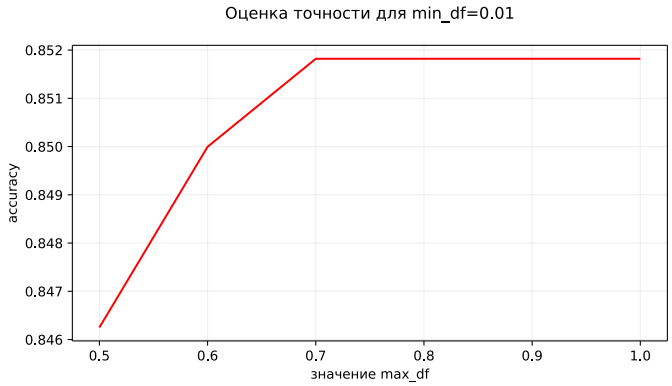
4.7.1 BOW

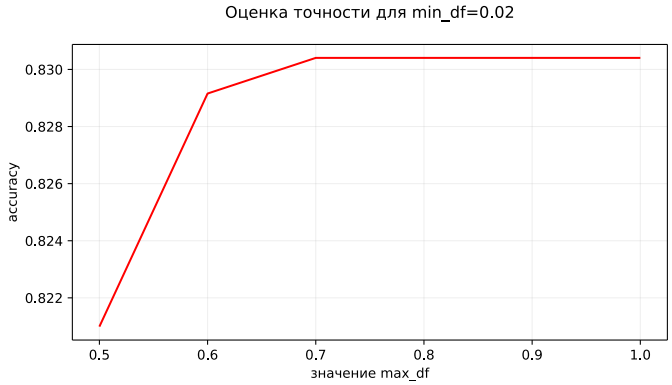
Сначала рассмотрим парамтеры `min_df` и `max_df` для BagOfWords.

Итак, лучшими комбинациями оказались:

1. `min_df` = 0.05 и `max_df` = 0.7
2. `min_df` = 0.04 и `max_df` = 0.7
3. `min_df` = 0.03 и `max_df` = 0.7
4. `min_df` = 0.02 и `max_df` = 0.7
5. `min_df` = 0.01 и `max_df` = 0.7

Лучшая точность достигается при `min_df` = 0.01 и `max_df` = 0.7: accuracy=0.852

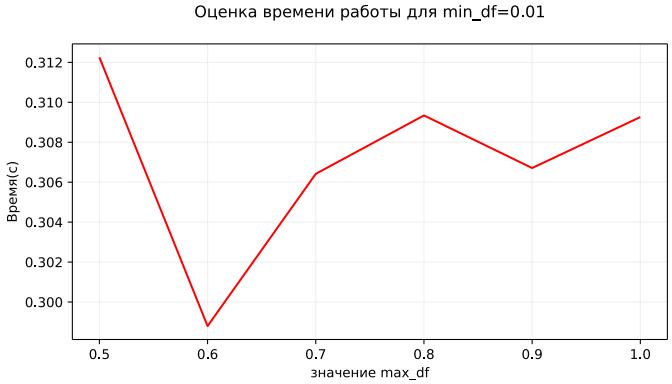


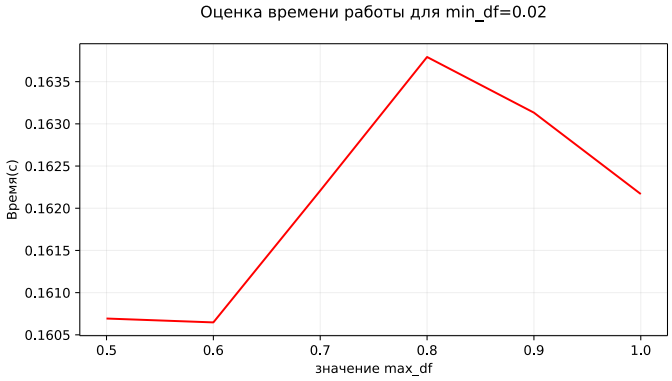


Также в ходе данного эксперимента заметил, что при `max_df`>=0.7 точность для текущего датасета не увеличивалась.

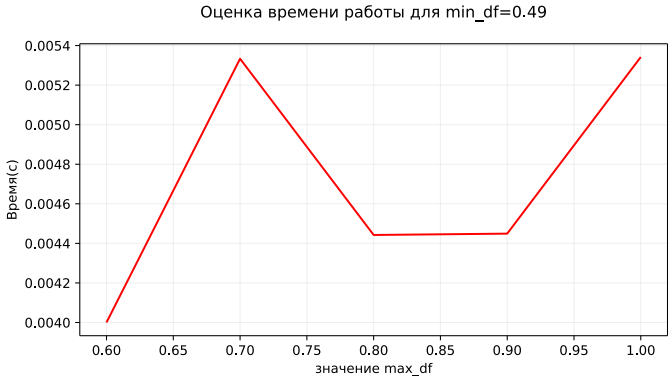
Оценим время работы алгоритма:

Самыми долгими оказались комбинации с малым `min_df` и большим `max_df`





А самым быстрым оказался алгоритм с наибольшим `min_df`



Думаю, это связано с тем, что у нас становится сильно меньше признаков.

Для `min_df` $= 0.01$ и `max_df` $= 0.7$ число признаков 575 (как и для `min_df` $= 0.02$ и `max_df` $= 0.7$).

А для `min_df` $= 0.49$ и `max_df` $= 0.7$ всего 3 признака.

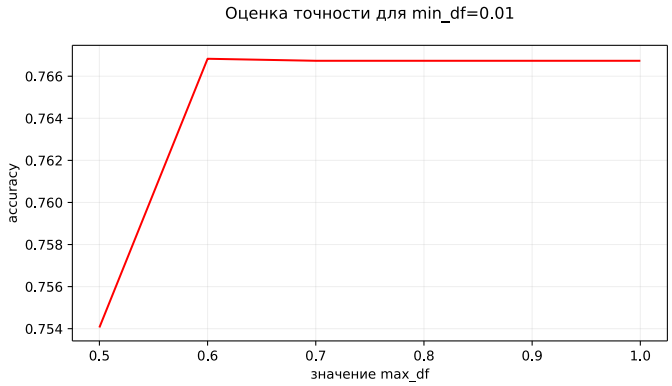
4.7.2 TF-IDF

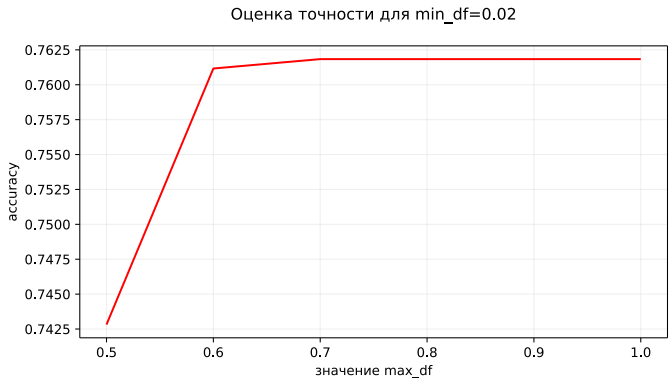
Теперь рассмотрим парамтеры `min_df` и `max_df` для TF-IDF.

Итак, лучшими комбинациями оказались:

1. `min_df` = 0.06 и `max_df` = 0.7
2. `min_df` = 0.04 и `max_df` = 0.7
3. `min_df` = 0.03 и `max_df` = 0.7
4. `min_df` = 0.02 и `max_df` = 0.7
5. `min_df` = 0.01 и `max_df` = 0.6

Лучшая точность достигается при `min_df` = 0.01 и `max_df` = 0.6: accuracy=0.767

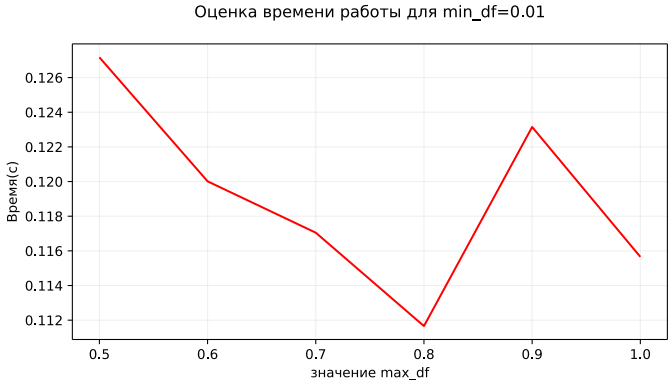


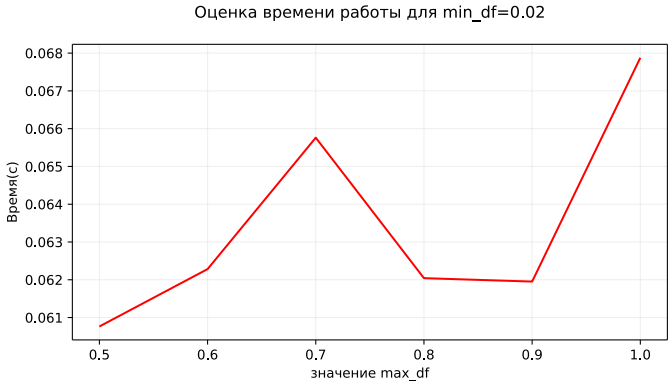


Для TF-IDF при `max_df`>=0.7 точность для текущего датасета не увеличивалась.

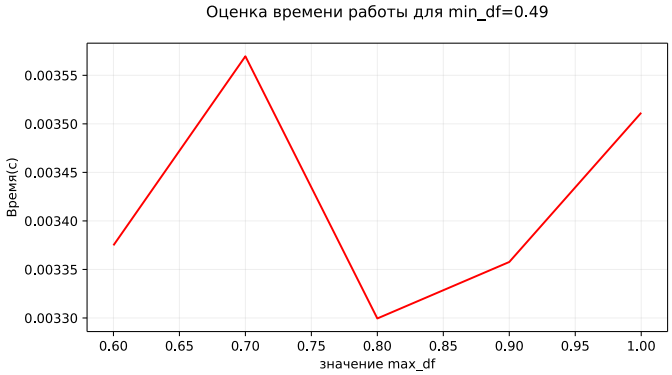
Оценим время работы алгоритма:

Самыми долгими оказались комбинации с малым `min_df` и большим `max_df`





А самым быстрым оказался алгоритм с наибольшим `min_df`



Размерности признакового пространства для BOW и TFIDF получились одинаковыми для рассмтариваемых параметров:
Для `min_df` $= 0.01$ и `max_df` $= 0.7$ число признаков 575 (как и для `min_df` $= 0.02$ и `max_df` $= 0.7$).

А для `min_df` $= 0.49$ и `max_df` $= 0.7$ - 3 признака.

Лучшее качество достигается при использовании BOW.

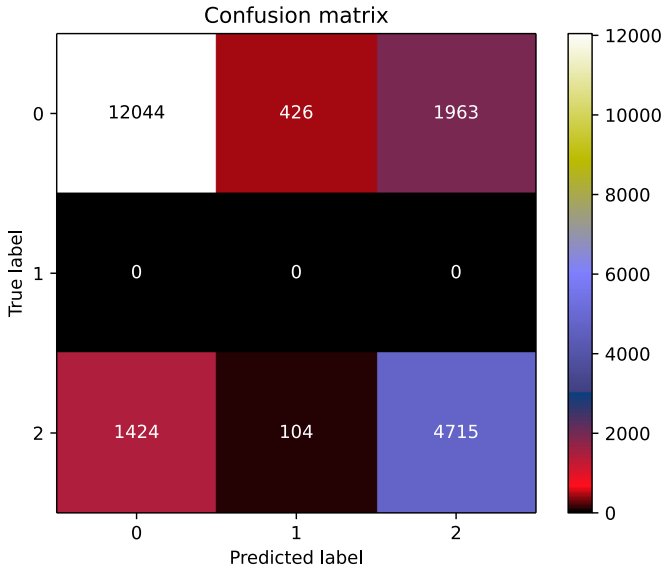
4.8 Эксперимент 9

Этот эксперимент проводился на тестовой выборке (в отличие от предыдущих экспериментов, которые ставились на валидационной).

Был выбран алгоритм градиентного спуска с $\alpha = 1.0$, $\beta = 0.34$, нулевым начальным приближением; использовано представление BagOfWords, `min_df` $= 0.01$ и `max_df` $= 0.9$. Также текст был лемматизирован.

При такой конфигурации модели точность ссоставила: 0.81

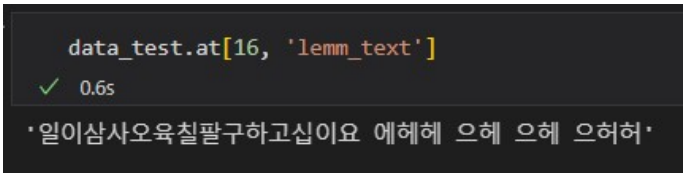
Проанализируем ошибки:



Здесь лейбл 0 - False в колонке 'is toxic' (положительно/нейтрально окрашенный текст), 1 - нулевое предсказание, 2 - True в колонке 'is toxic' (негативно окрашенный текст).

Неожиданным стало появление еще одного значения в предсказании и, как следствие, еще одной колонки в матрице ошибок.

Причина заключается в том, что после лемматизации остаются тексты на других языках.
Так, например, в 16-ой строке тестовой выборки находится корейский текст (прикреплен ниже). Требование первого эксперимента оставить только буквы и цифры выполнено, но данный текст все равно является неразличимым для модели, отсюда получаем нулевое предсказание. Всего в выборке 530 таких строк.



Что касается ошибок в предсказании False/True токсичности текста, то наибольшее число ошибок связано с "недостаточной" негавтиной окраской предложений.

К примеру, текст "dear god this site is horrible" модель определила как нормальный, хотя в тестовой выборке он отнесен к негативным. Думаю, это вызвано сочетанием как положительно окрашенных слов "dear god", так и отрицательно окрашенных "horrible". И в данном примере для модели более сильной стала положительная окраска.

Противоположная ситуация: текст "i think the origin of sagging has his roots in that human stupidity has no limits" отнесен моделью к негативным, тк "human stupidity", должно быть, является "триггером". Но в тестовой выборке этот отнесен к нейтральным, т.к. не оскорбляет кого-то, а просто излагает чей-то взгляд на мир/человечество.

4.9 Бонусное задание

В признаковое пространство были добавлены n-граммы. Рассмотрены 3 случая: биграммы, триграммы и 4-граммы.

Был выбран алгоритм градиентного спуска с $\alpha = 1.0$, $\beta = 0.34$, нулевым начальным приближением; использовано представление BagOfWords, `min_df` $= 0.01$ и `max_df` $= 0.9$. Также текст был лемматизирован.

Итак, при добавлении биграмм точность увеличилась до 0.8124 (без биграмм точность=0.81 - Эксперимент 9)

При добавлении триграмм точность незначительно снизилась - 0.8121.

При добавлении 4-грамм точность не изменилась (0.8121).

При этом время работы алгоритма на биграммах увеличилось почти на 42% в сравнении с алгоритмом на униграммах в 9-ом эксперименте.
Время работы на триграммах возросло еще на 5%, на 4-граммах на 2% (в сравнении с работой алгоритма на униграммах).

Таким образом видим, что использование n-грамм дает прирост качества (правда, незначительный), но при этом работает намного дольше на одном и том же объеме данных.

4.10 Общие выводы

В ходе работы над данным заданием были реализованы методы стандартного градиентного и стохастического спусков, изучены достоинства и недостатки каждого из данных алгоритмов. Также проанализированы зависимости точности, времени работы, размерности признакового пространства от представления, преодобработки текстов, а также от значений гиперпараметров.