Отчет по практическому заданию: Градиентные методы обучения линейных моделей

Выполнено студентом 317 группы, Филимоновым Владиславом

1 Введение

Данный отчет имеет три части:

- теоретическая часть (здесь приводится вывод формул, требуемых в условии задания [1]);
- особенности реализации (тут описаны особенности реализации моего метода обучения относительно стохастического ГС);
- эксперименты;

2 Теоретическая часть

В данном разделе будут приведены основные формулы для задачи логистической регрессии. Запишем задачу бинарной логистической регрессии в матричном виде:

$$Q(X, w) = \frac{1}{l}\{e^l, ln(e^l + exp(-y \odot Xw^T))\} + \frac{\lambda}{2}ww^T \to \min_w,$$

где $X \in R^{l\times d}$, $w \in R^{d\times 1}$, $y \in B^{l\times 1}$, $B = \{-1, 1\}$, $\lambda \in R$, а e^l - вектор-столбец длины l, составленный из единиц. Под $\{u, v\}$ понимается скалярное произведение векторов $u, v \in R^k, k \in N$. Запишем градиент функции потерь $\nabla_w Q(X, w)$:

$$\nabla_w Q(X, w) = \frac{1}{l}X^T\sigma((-y) \odot Xw^T) \odot (-y) + \lambda w,$$

где $\sigma(t) = \frac{1}{1+e^{-t}}$ применяется к вектору покомпонентно.

Теперь рассмотрим мультиномиальный случай. Заметим, что теперь $w \in R^{k\times d}$, где $k \in N$ - количество классов, а $y \in \{1, 2, \ldots, k\}^{l\times 1}$. Введем дополнительно матрицу $I \in R^{l\times k}$ так, что $I_{i,j} = [y_i = j]$. - индикатор того, что i-тый объект имеет класс j.

Тогда справедливо:

$$Q(X, w) = \frac{1}{l}\{e^l, (-Xw^T \odot I)e^k + ln((Xw^T)e^k)\} + \frac{\lambda}{2}((ww^T) \odot E_k)e^k,$$

где $E_k \in R^{k\times k}$ - квадратная единичная матрица порядка k.

Аналогично запишем градиент функции потерь $\nabla_w Q(X, w) \in R^{k\times d}$:

$$\nabla_w Q(X, w) = \frac{1}{l}(-I^T X + (exp(Xw^T))^T X \odot N) + \lambda w,$$

где $N \in R^{l\times k}$ так, что j-й столбец матрицы M состоит из l чисел $\frac{1}{exp(Xw^T)}e^k$(все столбцы одинаковы).

3 Особенности реализации

Так как при реализации обучения модели методом стохастического ГС необходимо на каждом шаге случайно выбирать небольшое количество объектов для подсчета градиента, то встает вопрос о реализации этого случайного выбора. Если просто на каждом шаге генерировать необходимое число целых чисел, то время обучения возрастает значительно, так как генерация случайных чисел достаточно затратная операция. Естественно возникает идея о том, что нужно генерировать простую перестановку индексов и работать с ней. Было рассмотрено два варианта:

- генерировать перестановку размером с обучающаю выборку один раз в эпоху;
- генерировать перестановку повторяющихся индексов для худшего случая(это возможно т.к. максимальное количество итераций, batch size и размер обучающей выборки известен).

Во втором случае рассматривается набор перестановок чисел от 1 до l, длины k (k перестановок $(1, \ldots, l)$) так, что $k * l > B * M$, где B - batch size, M - максимальное число итераций, l - размер обучающей выборки.

У каждого подхода есть свои недостатки:

- В случае первого подхода каждый объект будет учтен ровно один раз за эпоху, что является почти невозможным (показано ниже) в том случае, если бы мы каждый раз генерировали случайные числа (как и указано в исходном алгоритме стохастического ГС). Получается, что мы отходим от теоретического алгоритма;
- В случае второго подхода мы все равно не решаем указанную выше проблему, так как каждый объект может встретиться не более чем k раз, но это уже значительно ближе к теоретическому алгоритму.

Покажем, что вероятность того, что хотя бы 1 объект встретится дважды за эпоху, стремится к 1 при увеличении размера обучающей выборки.

Для определенности рассмотрим случай $batchsize = 1$. Через A обозначим вероятность данного события, через $\bar{A}$ событие, состоящее в том, что каждый объект встретился ровно 1 раз за эпоху, при этом будем считать, что выбор объектов происходит независимо и равновероятно.

$$P(A) = 1 - P(\bar{A}) = 1 - \frac{l!}{l^l} \longrightarrow 1$$

Последний предел следует из формулы Стирлинга.

В данной работе было принято решение выбрать первый подход, так как, несмотря на отхождение от теоретического алгоритма, данный подход заставит нас рассмотреть все объекты обучающей выборки почти одинаковое (отличающееся не более чем на 1) количество раз, что не должно ухудшить сходимость алгоритма.

4 Эксперименты

В каждом эксперименте (если не указано отдельно) параметры алгоритмов соответствуют параметрам, описанным в задании [1].

4.1 Сравнение работы алгоритмов полного и стохастического ГС

В данном эксперименте была исследована зависимость времени обучения, изменений значений функции потерь и точности в процессе обучения для алгоритмов полного и стохастического ГС для бинарной логистической регрессии от гиперпараметров моделей.

Из рис. 1 видно, что при использовании стохастического ГС минимизация функции потерь происходит значительно быстрее, хотя при этом допускаются небольшие осцилляции.

По рис. 2 можно сделать вывод, что наибольший вклад в минимизацию функции потерь при полном градиентном спуске вносят первые итерации алгоритма.

Из рис. 3 следует, что стохастический ГС в целом ведет себя одинаково вне зависимости от начального приближения. При этом очевидно, что начальные значения функции потерь различны для разных начальных приближений, но после достаточного числа итераций, значения функции потерь для разных начальных приближений отличаются незначительно.

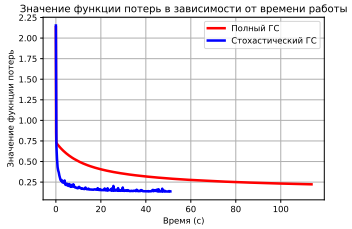

Рис. 1: Функция потерь от времени работы

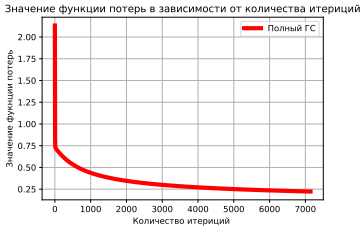

Рис. 2: Функция потерь от количества итераций

Рис. 3: Функция потерь от эпохи

Далее исследовалось поведение точности в ходе обучения алгоритма. При этом в метод fit передавалась тестовая выборка, и точность оценивалась либо на каждом шаге для полного ГС , либо раз в эпоху для стохастического ГС.

Из рис.4 следует, что стохастический ГС при значительно меньших временных затратах имеет большие значения точности, что говорит о том, что обучение с помощью стохастического ГС значительно быстрее.

По рис. 5, 6 можно сделать вывод, что наибольший вклад в прирост точности при полном и стохастическом градиентном спуске вносят первые итерации алгоритма.

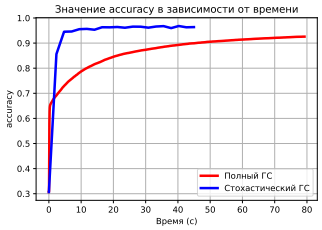

Рис. 4: Accuracy от времени работы

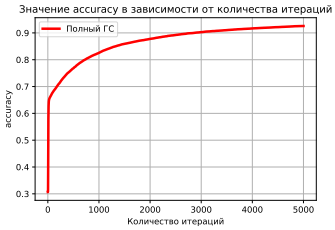

Рис. 5: Accuracy от количества итераций

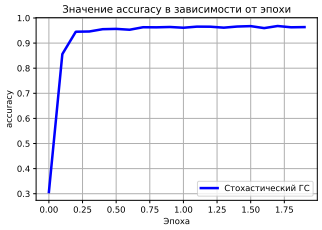

Рис. 6: Accuracy от эпохи

4.2 Подбор оптимального шага ГС

Напомним, что длина шага на k-ой итерации может быть вычислена по формуле:

$$\eta_k = \frac{\alpha}{\beta^k}.$$

В данном эксперименте сначала будет проведен подбор параметра α при фиксированном $\beta = 0$. А затем при подобранном α будет исследовано поведение алгоритмов ГС при различных β.

На рис. 7, 8 приведено несколько графиков точности от времени обучения для различных α. Из этих графиков можно сделать вывод, что наилучшим параметром является $\alpha = 1$ при фиксированном $\beta = 0$. Также стоит заметить, что $\alpha = 100$ в начале дает лучший результат, чем $\alpha = 1$, но имеет слишком сильные скачки, что мешает ему сойтись к хорошему результату, так что имеет смысл рассмотреть $\alpha = 100$ при $\beta \neq 0$.

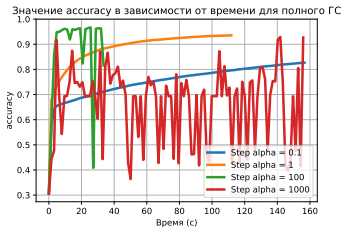

Рис. 7: Accuracy от времени работы для полного ГС

Рис. 8: Accuracy от времени для стохастического ГС

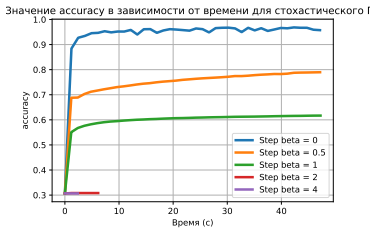

Рис. 9: Accuracy от времени для стохастического ГС

Из рис.9 можно сделать вывод, что $\beta \neq 0$ при $\alpha = 1$ приводит к тому, что алгоритм сходится раньше, но при этом обучается недостаточно, так как шаг с каждой итерацией уменьшается, а значит вклад объектов в градиент также уменьшается с каждой итерацией. Можно сделать вывод, что $\beta \neq 0$ не дает прироста в точности для $\alpha = 1$. Следует заметить, что в 4.1 было отмечено, что при $\beta = 0$ и $\alpha = 1$ наибольший прирост точности достигается на первых итерациях, и при этом в дальнейшем точность не снижалась, следовательно у нас не было предпосылок брать $\beta \neq 0$ для $\alpha = 1$.

Таблица 1: Сравнение поведения алгоритмов для различных β при $\alpha = 1$

	Полный				Стохастический			
β	кол. итераций	время	точность	loss	кол. эпох	время	точность	loss
0	7149	114.9	0.94	0.22	1.95	47.5	0.95	0.14
0.5	2392	38.41	0.68	0.68	1.95	47.7	0.79	0.51
1	3018	48.1	0.55	0.80	1.95	47.7	0.61	0.75
2	178	2.81	0.31	1.68	0.25	6.1	0.30	1.85
4	15	0.22	0.30	2.08	0.10	2.4	0.30	2.11

На рис.9 графики не очень хорошо различимы, аналогичная ситуация будет с графиками для полного ГС, поэтому анализ различных β проведем по таблице 1. Из этой таблицы можно сделать вывод, что при $\beta \neq 0$ и $\alpha = 1$ время обучения снижается, но при этом значительно падает качество алгоритма. Объяснение этому приведено выше(анализ рис.9). Заметим, что при $\beta = 0, 0.5, 1$ время обучения с помощью стохастического ГС не уменьшается, в то время как качество падает. Это можно объяснить тем, что критерий выхода на k-ой эпохе в данной работе представляет собой

$$|L(x^{k+1}, w^{k+1}) - L(x^k, w^k)| < \epsilon,$$

где ϵ - точность, x^k - объекты обучающей выборки, выбранные в начале k-ой эпохи, w^k - веса в начале k-ой эпохи, $L(x, w)$ - функция потерь. Таким образом, мы делаем маленькие шаги, но берем другие объекты(возможно не просмотренные), что может приводить к достаточно большой разнице функции потерь. Поэтому мы не всегда останавливаемся раньше и при этом всегда обучаемся недостаточно.

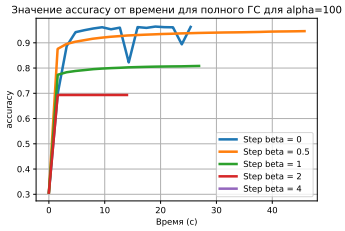

Рис. 10: Accuracy от времени работы для полного ГС для $\alpha = 100$

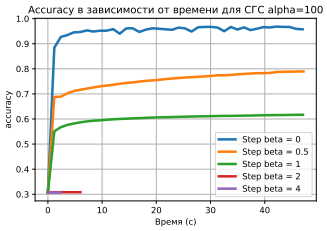

Рис. 11: Accuracy от времени для стохастического ГС для $\alpha = 100$

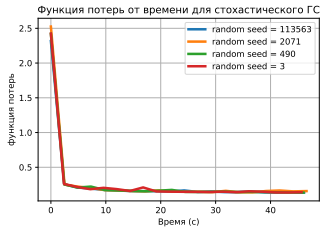

Рис. 12: Функция потерь от времени для стохастического ГС

Рассмотрим теперь $\alpha = 100$ и $\beta \neq 0$. На рис 10, 11 приведены графики точности в процессе обучения для различных β. Из этих графиков можно сделать вывод, что полный ГС дает хороший результат на $\alpha = 100$ и $\beta = 0.5$. Время обучения и точность при данных параметрах почти такие же, как и у стохастического ГС при $\beta = 0$.

В дальнейших экспериментах, если не обговорено отдельно, $\beta = 0$, $\alpha = 1$

4.3 Влияние случайности выбора объектов на качество стохастического ГС

В данном эксперименте исследуется влияние random seed на качество стохастического ГС. Было выбрано небольшое множество случайных чисел в качестве random seed и проведено обучение стохастическим ГС. На рис. 12 видно, что уменьшение функции потерь не зависит от random seed. Но так как эти графики очень похожи и пересекаются во многих точках, анализ не очень удобен. Для того, чтобы окончательно убедиться, что различные random seed приводят к одному результату, можно просмотреть таблицу 2 (все значения были округлены до второго знака после запятой).

Таблица 2: Характеристики обучения СГС с различным random seed

random seed	количество эпох	время	точность	loss
113563	1.95	47.23	0.96	0.14
2071	1.95	47.15	0.96	0.14
490	1.95	46.98	0.96	0.14
3	1.95	46.99	0.96	0.14

4.4 Влияние batch size на качество алгоритма

В данном эксперименте были исследованы batch size $\in \{1, 5, 10, 20, 50, 100\}$. Причем для $batchsize = 1$ был взят $logfreq = 0.1$, а для остальных $logfreq = 0.5$. Это было сделано по той причине, что количество эпох для $batchsize = 1$ значительно меньше, чем при остальных значениях (см таблицу 3). Графики функции потерь и точности от времени обучения оказались достаточно похожими, поэтому приводить их все на одной картинке нет смысла. На рис. 13, 14 приведены данные графики для $batchsize \in \{1, 100\}$. Для анализа остальных значений batch size приведена таблица 3.

Из приведенных данных можно сделать вывод, что большие размеры batch size приводят к снижению времени обучения, но при этом конечное значение функции потерь может оказаться немного больше, чем при меньших batch size(порядка 0,01). Так же по графикам можно сделать вывод, что при увеличении batch size функция потерь убывает более гладко и равномерно (не имеет резких скачков и осцилляций). На точность batch size оказывает аналогичное действие (только точность возрастает в процессе обучения, а не убывает, и конечное значение точности может оказаться немного ниже (так же порядка 0,01))

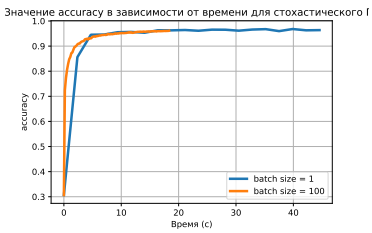

Рис. 13: Accuracy от времени работы для стохастического ГС

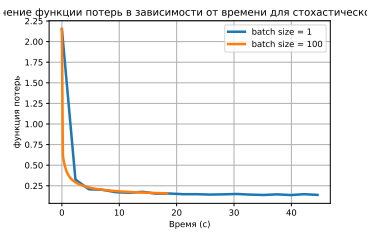

Рис. 14: Функция потерь от времени для стохастического ГС

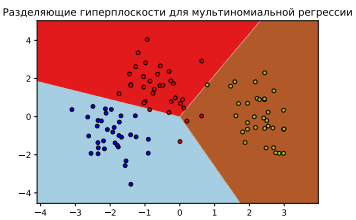

Рис. 15: Разделяющие гиперплоскости для прямого обобщения

Таблица 3: Характеристики обучения СГС с различным batch size

batch size	количество эпох	время	точность	loss	количество итераций
1	1.90	44.55	0.96	0.14	96177
5	9.50	46.40	0.97	0.13	96178
10	19.50	50.07	0.97	0.13	98709
20	39.01	52.63	0.97	0.136	98748
50	48.075	29.65	0.96	0.14	48671
100	49.67	18.33	0.96	0.15	25145

4.5 Исследование алгоритмов обобщения бинарной регрессии

В данном разделе будет проведено сравнение алгоритмов one-vs-all, all-vs-all, а так же прямого обобщения логистической регрессии на многоклассовый случай с помошью softmax. Была сгенерирована

выборка (для визуализации и генерации я пользовался примером, указанным в [2]) с 2 признаками, состоящая из 100 объектов, также была сгенерирована тестовая выборка размера 30.

На рис. 15, 16,17 визуализирована выборка и разделяющие гиперплоскости для каждого из алгоритмов. Разделяющие гиперплоскости для one-vs-all и для прямого обобщения похожи, что достаточно логично, ведь в обоих алгоритмах обучается одинаковое количество бинарных классификаторов (в случае прямого обобщения имеется ввиду настройка соответствующего вектора весов). Граница, разделяющая классы, для обоих подходов должна иметь вид ломанной, составленной из прямых отрезков (так как каждые два класса разделяются прямой). Существенным отличием этих двух алгоритмов является то, что прямое обобщение имеет возможность оценивать вероятности принадлежности объекта к каждому из классов.

В случае all-vs-all гиперплоскость может иметь более сложную природу(больше изломов, но отрезки также должны быть прямые), что заметно на рис.17. Также если какой-то объект наберет одинаковое количество голосов для i,j классов, то моя реализация всегда выберет класс с меньшим номером. Таким образом, можно сделать вывод, что в случае большого количества объектов с маленьким отступом от границы, all-vs-all будет работать хуже, чем другие реализации. Так же я считаю, что one-vs-all будет работать хуже, чем прямое обобщение в случае наличия объектов, имеющих одинаковое признаковое описание, но разные метки классов. Примера, в котором прямое обобщение будет работать хуже других реализаций, я построить не смог.

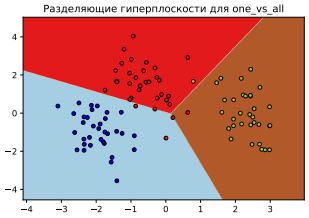

Рис. 16: Разделяющие гиперплоскости для one-vs-all

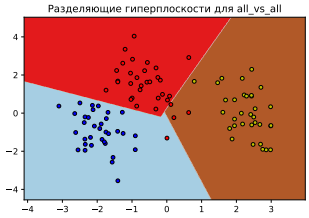

Рис. 17: Разделяющие гиперплоскости для all-vs-all

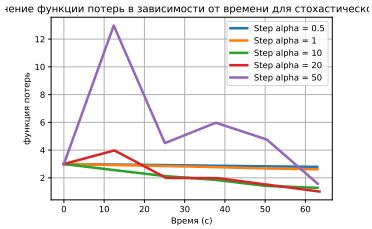

Рис. 18: Поведение функции потерь для различных step alpha

Также стоить заметить, что all-vs-all требует обучения большего количества классификаторов, но обучающие выборки при этом меньше, так как каждый классификатор обучается лишь на соответствующей части обучающей выборки. Это показывает еще одну проблему all-vs-all: в случае сильно несбалансированных классов all-vs-all будет иметь как минимум один классификатор, который будет почти константным.

4.6 Эксперименты с текстом, tf-idf преобразование.

В данном разделе будут проводиться эксперименты с текстовым датасетом 20newsgroups. Классификация будет производиться с помощью прямого обобщения логистической регрессии. Была проведена предобработка данных: все цитаты, заголовки и подписи были убраны из датасета, все символы отличные от букв и цифр были заменены на пробелы, все пробельные последовательности были заменены на единичные пробелы, каждый текст был переведен в sparce matrix, причем признаками являются все различные слова в датасете, а значение признака - количество вхождений данного слова в текст. Затем было применено tf-idf преобразование.

Для начала настроим гиперпараметры классификатора. Так как время обучения на данном датасете занимает значительное время, проверка точности для различных параметров будет сделана на отложенной выборке.

Step alpha выбирался из множества $\{[0.5, 5, 10, 20, 50\}$. На рисунке 17 показано поведение функции потерь при различных step alpha. Другие параметры были взяты такими же, как в 4.1. Исходя из графика 18, было выбрано $alpha = 10$.

Следующим из гиперпараметров выбирался batch size. На рисунке 19 показано поведение функции потерь для различных batch size. Был выбран batch size = 100, так как большее значение

требует значительно большего времени обучения притом, что значение функции потерь уменьшилось незначительно.

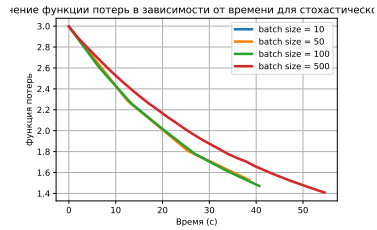

Рис. 19: Поведение функции потерь для различных batch size

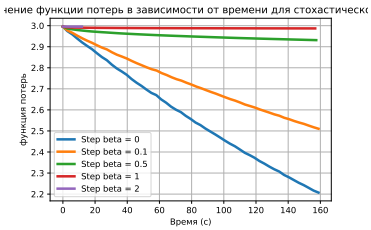

Рис. 20: Поведение функции потерь для различных step beta

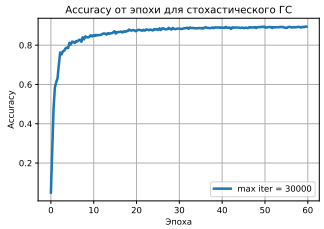

Рис. 21: Точность при большом числе итераций

Далее исследовалось влияние step beta на сходимость. Любое значение step beta отличное от 0 приводит к преждевременной сходимости из-за сильного убывания длины шага рис. 20)

Для подбора максимального числа итераций алгоритма использовался анализ 2 показателей - убывание функции потерь и возрастание точности на тестовой выборке. Было сделано предположение, что ситуация, при которой функция потерь убывает на обучающей выборке, а точность не растет, свидетельствует о переобучении и снижении обобщающей способности алгоритма. На рис. 21, 22 приведены графики точности и функции потерь. Выбор максимального числа итераций был сделан так, чтобы максимально алгоритм мог сделать 30 эпох при размере обучающей выборки - примерно 10 тысяч. Это значение (30 эпох) было выбрано, потому что точность растет незначительно на дальнейших эпохах, в то время как функция потерь продолжает убывать. Таким образом, максимальное число итераций - 3000.

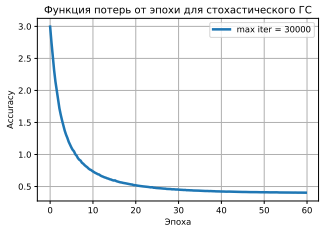

Рис. 22: Функция потерь при большом числе итераций

Далее исследовалось влияние tf-idf преобразования на качество алгоритма. Для этого было рассмотрено две обучающих выборки: до tf-idf преобразования и после него. Затем в каждой из этих выборок была отложена часть выборки, а обучение шло по оставшейся части. При этом были использованы подобранные выше параметры. Точность на отложенной выборке с tf-idf преобразованием оказалась выше на 8%, чем без применения данного преобразования. Таким образом, был сделан вывод, что tf-idf преобразование значительно повышает точность.

4.7 Анализ типичных ошибок алгоритма

Перед началом описания этого эксперимента стоит отметить одну особенность. Полученная точность после подбора параметров была примерно 0,8. Я достаточно долго пытался найти ошибку или причину такой высокой точности (так как в конференции группы отмечалось, что на линейной модели нельзя получить точность больше 0,74). Так как ни причина, ни ошибка не были найдена, я запустил sklearn'овский алгоритм, с параметрами: multi_class='multinomial', solver='sag', max_iter=3000. Он имел точность примерно 0,78. (код для sklearn'овского алгоритма приложен в отдельном файле skl.ipynb). Возможно, мной использовалась другая версия датасета (sklearn был обновлен когда была обнаружена данная особенность, но после обновления ничего не изменилось).

Точность на тестовой выборке оказалась ниже примерно на 7,5%, чем на отложенной. Этому есть несколько причин:

- Признаковое описание отложенной выборки полностью совпадало с обучающей (все токены отложенной выборки являлись признаками), а признаковое описание тестовой было подогнано под признаковое описание обучающей (если слово не входит в обучающую выборку, то оно просто игнорируется, если слово входит в обучающую выборку, но не входит в тестовую, то у всех объектов будет нулевой признак)
- Размер тестовой выборки примерно в 5 раз больше, чем размер отложенной.

Рассмотрим несколько объектов, на которых была допущена ошибка:

- 'please subscribe me e mail rpicas porto inescn pt '
- ' i heard that there is a vesa driver for the xga 2 card available on compuserve i just got this card and i am wondering if this driver is available on a ftp site anywhere my news service has beeen erratic lately so please e mail me at walsh stolaf edu thanks in advance '
- 'has anyone ever heard of the x professional organization is anyone a member is the membership worth the 100 or so that they charge tim bomgardner '
- 'shut up andi'
- 'sorry forgot to add its a jap import andy '

Может показаться, что ошибка допускается в основном на коротких статьях, но это не так, так как в отчет были специально включены статьи с короткой длиной. Средняя длина текста, на котором допускается ошибка, является 1200 символов(включая пробелы). Можно заметить, что короткие статьи содержат сленг, аббревиатуры и опечатки, что естественно осложняет их классификацию. Для анализа типичных ошибок на рис.23 приведена матрица ошибок. Так как метки классов не являются достаточно информативными приведем дополнительно таблицу соответствия меток темам статьи:

Таблица 4: Соответствие меток классов темам

Метка класса	тема
0	alt.atheism
1	comp.graphics
2	comp.os.ms-windows.misc
3	comp.sys.ibm.pc.hardware
4	comp.sys.mac.hardware
5	comp.windows.x
6	misc.forsale
7	rec.autos
8	rec.motorcycles
9	rec.sport.baseball
10	rec.sport.hockey
11	sci.crypt
12	sci.electronics
13	sci.med
14	sci.space
15	soc.religion.christian
16	talk.politics.guns
17	talk.politics.mideast
18	talk.politics.misc
19	talk.religion.misc

Можно заметить, что наибольшое количество ошибок совершается на смежных темах:

- 19 классифицруется как 0 35 раз (тема религии и тема атеизма);
- 5 как 1 45 раз (компьютерная графика и windows.x);
- 2 как 3 34 раза (ms windows и оборудование ibm);
- 19 как 15 44 раза(христианство и религия);
- 18 как 16 88 раз(политика в общем и политика на тему оружия);
- 0 как 15 (атеизм и христианство).

Таким образом, данных ошибок достаточно сложно избежать, так как темы действительно имеют много общего и достаточно много слов могут быть использованы в контексте смежных тем равновероятно.

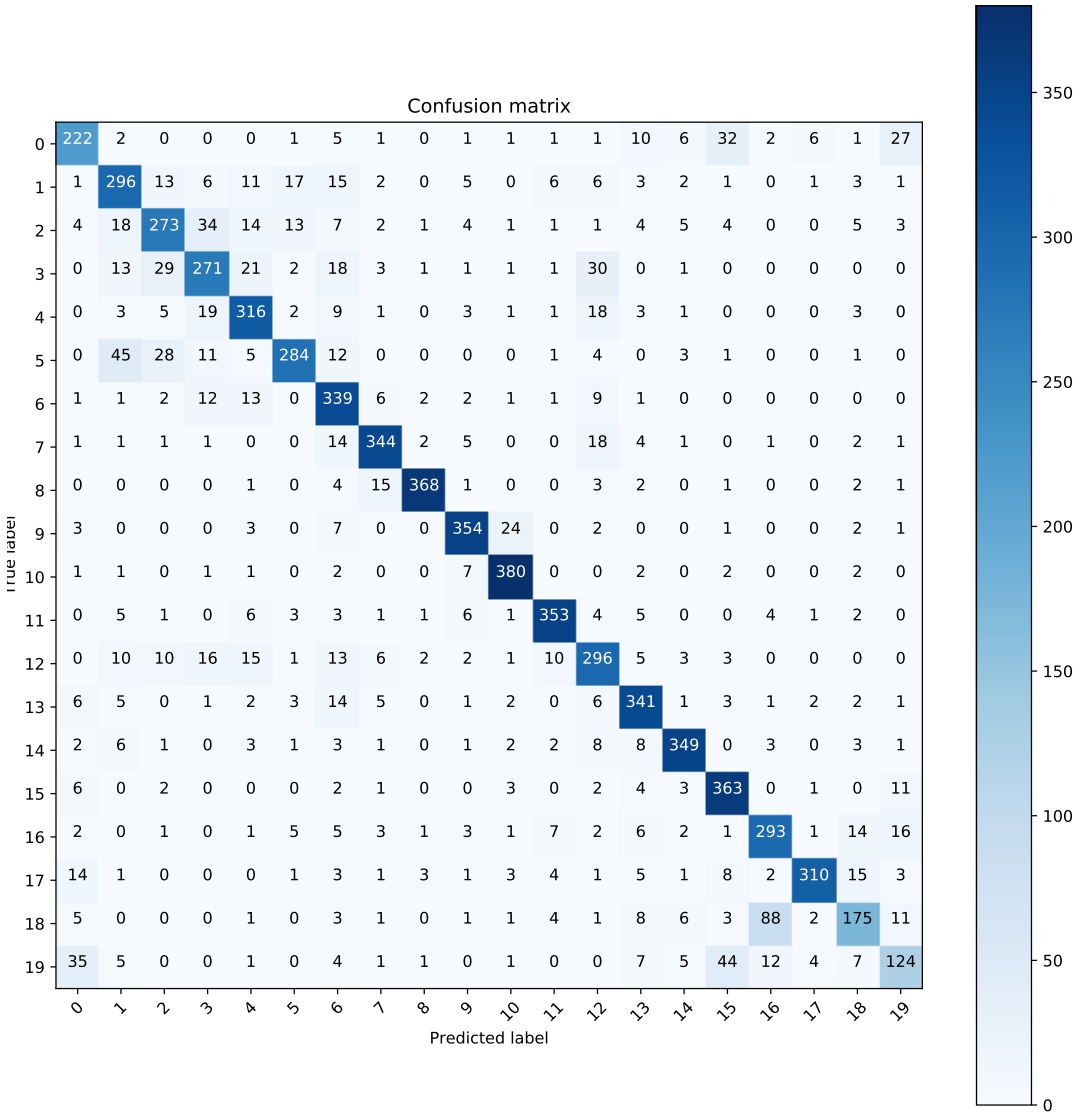

Рис. 23: Матрица ошибок

4.8 Лемматизация и стемминг.

В данном разделе исследуется влияние на работу алгоритма таких предобработок, как стемминг и лемматизация. Было исследовано влияние на точность, время работы и размер признакового пространства. В таблице 5 приведены результаты данного исследования. При этом под временем предобработки понимается суммарное время на предобработку тестовой и обучающей выборок.

Таблица 5: Влияние предобработок на работу алгоритма

Предобработка	предобработка(сек)	обучение(сек)	количество признаков	точность
Нет	0	410	121590	0.803
Лемматизация	746	370	113260	0.799
Стемминг	76	347	102833	0.804
Сокращение словаря	3	80	24335	0.802
Stop words	3	82	24301	0.803

Для иллюстрации работы данных алгоритмов предобработки приведем пример:

- Без предобработки:' i was wondering if anyone out there could enlighten me on this car i saw the other day it was a 2 door sports car looked to be from the late 60s early 70s it was called a bricklin the doors were really small in addition the front bumper was separate from the rest of the body this is all i know if anyone can tellme a model name engine specs years of production where this car is made history or whatever info you have on this funky looking car please e mail thanks il brought to you by your neighborhood lerxst ';
- Лемматизация: 'i be wonder if anyone out there could enlighten me on this car i saw the other day it be a 2 door sport car look to be from the late 60 early 70 it be call a bricklin the door be really small in addition the front bumper be separate from the rest of the body this be all i know if anyone can tellme a model name engine spec year of production where this car be make history or whatever info you have on this funky look car please e mail thanks il bring to you by your neighborhood lerxst';
- Стемминг:'i was wonder if anyon out there could enlighten me on this car i saw the other day it was a 2 door sport car look to be from the late 60s earli 70s it was call a bricklin the door were realli small in addit the front bumper was separ from the rest of the bodi this is all i know if anyon can tellm a model name engin spec year of product where this car is made histori or whatev info you have on this funki look car pleas e mail thank il brought to you by your neighborhood lerxst'

Из этого примера становится понятно, почему лемматизация занимает значительно больше времени: поиск начальной части речи значительно сложнее, чем простое отрезание окончания, причем не всегда корректное (anyon хотя у anyone нулевое окончание). Но при этом стемминг может приводить разные части речи, имеющие общий корень, к одному слову(reality и realy к reali). Этим можно объяснить небольшой прирост точности, так как корень слова обычно играет более семантическую роль, в то время как часть речи имеет, в основном, грамматическую роль.

4.9 Сокращение словаря.

Перед проведением эксперимента было проверено, что все классы примерно сбалансированы по количеству объектов. Далее было сделано предположение, что слова, встречающиеся более чем в 15% документов, не несут в себе информации о теме (так как каждый класс занимает примерно 5% документов). Такие слова будем считать частотными и выбросим их из словаря. При такой операции было удалено 108 слов, большая часть из них являлась союзами, местоимениями и предлогами.

Также были выброшены слова, которые встречаются менее чем в 5 документах. Было посчитано, что в среднем на каждую тему приходится примерно 500 текстов, поэтому если слово встречается реже 5 раз во всей обучающей выборке, то это слово скорее всего не является типичным словом для какой-либо темы. При таком преобразовании было выброшено 97147 слов. В таблице 5 приведена информация о том, как данное преобразование повлияло на работу алгоритма.

Теперь проверим, как изменится качество работы алгоритма, если вместо частотных слов выбрасывать стоп слова. Для этого была использована функция stopwords.words из модуля nltk. При этом были также выброшены редкие слова алгоритмом, описанным выше. В таблице 5 показано влияние на время работы и точность алгоритма данного преобразования (Данная предобработка была названа 'stop words'). Можно заметить, что точность при таком преобразовании оказалась немного выше (порядка 0,001). Это можно объяснить тем, что не все слова, которые не несут в себе никакой информации о теме (предлоги, частицы, междометия, цифры и т.д.) являются частотными. Они могли встречаться реже, но тем не менее не нести в себе реальной информации о теме.

Таким образом, можно сделать вывод, что сокращение словаря является эффективным средством снижения времени работы алгоритма и размера признакового пространства, которое при этом снижает точность незначительно.

4.10 Выделение n-грамм (бонус)

В данном разделе будут выделены n-граммы различной длины (2, 3, 4). При этом также будут выкинуты редкие слова(встречающиеся меньше, чем в 5 документах) и стоп-слова. В таблице 6 показано, как изменялись показатели работы алгоритма для различных длин n-грамм.

Таблица 6: Влияние предобработок на работу алгоритма

Длина n-граммы	предобработка(сек)	обучение(сек)	количество признаков	точность
1	3	82	24301	0.803
2	9	213	59995	0.808
3	18	269	76971	0.807
4	27	316	91297	0.806

Таким образом, из приведённой выше таблицы можно сделать вывод, что выделение n-грамм улучшает качество работы алгоритма, но при этом время работы растет. Можно заметить, что на данном датасете выделение 2-грамм дало наибольший прирост точности при меньших затратах времени. Это можно объяснить тем, что устойчивых выражений длины 2 значительно больше, чем устойчивых выражений длины 3, но при этом, в силу того, что датасет написан на английском языке (языке со строгим порядком слов) 3 и 4-граммы могут быть частью грамматических конструкций (например, 'it is said', 'it is reported' , что можно перевести как 'сообщается'. Эти словосочетания, очевидно, могут встречаться достаточно часто, но не нести в себе информации о теме). Таким образом, n-граммы длины 3 и более могут быть не очень эффективны для английского языка.

4.11 Заключительные выводы

Из всех проведенных выше экспериментов можно сделать следующие общие выводы о градиентных методах обучения моделей и методах обработки текстов:

- стохастический градиентный спуск является хорошей альтернативой полному градиентному спуску при большом размере обучающей выборки и большом размере признакового пространства в силу того, что обучение этим методом проходит в среднем значительно быстрее, при этом функция потерь сходится к тому же значению минимума;
- подбор параметров метода обучения играет большую роль в скорости сходимости (и наличия сходимости вообще);
- прямое обобщение логистической регрессии работает быстрее, чем one-vs-all и all-vs-all, при этом для one-vs-all и all-vs-all были предложены условия, при которых классификация этими методами будет иметь меньшую точность, чем прямым обобщением;
- граница разделяющих гиперплоскостей может иметь более сложную форму(больше изломов) для all-vs-all, но при этом все равно остается ломанной, составленной из отрезков или лучей (как и для one-vs-all и для прямого обобщения);

- tf-idf преобразование дает значительный прирост в точности при анализе текстов;
- большая часть признаков при работе с текстами является избыточной (при выбрасывании редких слов точность падает незначительно, а скорость обучения вырастает в разы);
- лемматизация занимает значительно больше времени, чем стемминг, который занимает значительно больше времени, чем выбрасывание редких и частотных слов (или стоп слов);
- выбрасывание редких и стоп слов снижает время обучения без снижения точности, а если дополнительно выделить n-граммы, то точность вырастает, а время обучения остается меньше;
- при стемминге, сокращении словаря и выделении n-грамм время на предобработку уходит значительно меньше времени, чем на обучение модели;
- предобработка текстов значительно помогает при работе с ними.

Список литературы

[1] https://github.com/arti32lehtonen/mmp_prac_2017/blob/master/Tasks/task2/task2.pdf

[2] http://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_multinomial.html