Московский Государственный Университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Фоминская Галина
кафедра ММП, группа 317
2017

Градиентные методы обучения линейных моделей

1 Введение

В данной работе был проведен анализ методов логистической и мультиномиальной регрессии для задач классификации. Рассмотренная реализация методов была написана на языке *Python*. Эксперименты были проведены на датасетах real-sim и 20newsgroups. В качестве меры качества использовалась accuracy. Также были реализованы методы one-vs-all и all-vs-all как альтернативные варианты обобщения бинарных классификаторов на многоклассовый случай.

2 Вывод формул

2.1 Логистическая регрессия

Представим формулы для функции потерь в задаче логистической регрессии и для её градиента через матричные операции.

$$Q(X, w) = \frac{1}{l}\sum_{i=1}^{l} \log(1 + \exp\left(-y_i * \langle X_i, w\rangle\right)) + \frac{\lambda}{2} w^T w =$$

$$= \frac{1}{l}\langle \log\left(e + \exp\left(-y * Xw\right)\right), e\rangle + \frac{\lambda}{2} w^T w,$$

где e — вектор из единиц, размер которого совпадает с размером y, $*$ — операция поэлементного перемножения матриц, операции log и exp также применяются поэлементно. Тогда градиент функции потерь равен:

$$\nabla_w Q(X, w) = \frac{1}{l}\left(X^T\left(-y * \frac{1}{e + \exp(-y * Xw)}\right)\right)^T + \lambda w,$$

где операция деления тоже поэлементная.

2.2 Мультиномиальная регрессия

Теперь запишем формулы функции потерь и градиента для задачи мультиномиальной регрессии.

$$Q(X, w) = -\frac{1}{l}\sum_{i=1}^{l} \log \frac{\exp\langle X_i, w_{y_i}\rangle}{\sum_{k=1}^{K} \exp\langle X_i, w_k\rangle} + \frac{\lambda}{2}\sum_{k=1}^{K} ||w_k||^2 =$$

$$= \frac{1}{l}(trace(-Xw^T[y]) + \log(\exp(wX^T).sum(axis = 0)).sum()) + \frac{\lambda}{2} w^2.sum()$$

Тогда градиент выглядит так:

$$\nabla_w Q(X,w) = -\frac{1}{l}\left(MX - \left(\frac{\exp\left(Xw^T\right)}{\exp\left(Xw^T\right).sum(axis=1)}\right)^T X\right) + \lambda w,$$

где $M \in \mathbb{R}^{l,K}$, $M_{ij} = 1$, если $y_i = j$, иначе $M_{ij} = 0$.

2.3 Эквивалентность логистической и мультиномиальной регрессии при $k = 2$

Запишем формулу функции потерь мультиномиальной регрессии при $k = 2$:

$$Q(X,w) = -\frac{1}{l}\sum_{i=1}^{l}\log\frac{\exp\left(\langle X_i, w_1\rangle I[y_i = 1] + \langle X_i, w_{-1}\rangle I[y_i = -1]\right)}{\exp\langle X_i, w_1\rangle + \exp\langle X_i, w_{-1}\rangle} + \frac{\lambda}{2}\left(w_1^T w_1 + w_{-1}^T w_{-1}\right) =$$

$$= \frac{1}{l}\sum_{i=1}^{l}\log\left(1 + \exp\left(-y_i * \langle X_i, w_1 - w_{-1}\rangle\right)\right) + \frac{\lambda}{2}\left(w_1^T w_1 + w_{-1}^T w_{-1}\right)$$

Положим $w = w_1 - w_{-1}$, тогда:

$$Q(X,w) = \frac{1}{l}\sum_{i=1}^{l}\log\left(1 + \exp\left(-y_i * \langle X_i, w\rangle\right)\right) + \frac{\lambda}{2}\left((w + w_{-1})^T(w + w_{-1}) + w_{-1}^T w_{-1}\right)$$

$$(w + w_{-1})^T(w + w_{-1}) + w_{-1}^T w_{-1} = w^T w + 2w^T w_{-1} + 2w_{-1}^T w_{-1} = w^T w + 2w_1^T w_{-1}$$

Заметим теперь, что итоговые вероятности можно выразить через w:

$$P(y = 1|x) = \frac{\exp\langle X_i, w_1\rangle}{\exp\langle X_i, w_1\rangle + \exp\langle X_i, w_{-1}\rangle} = \frac{1}{1 + \exp\left(-\langle X_i, w_1 - w_{-1}\rangle\right)} = \frac{1}{1 + \exp\left(-\langle X_i, w\rangle\right)}.$$

Следовательно, ответ не зависит от w_{-1} и можем положить его равным нулевому вектору. Тогда видно, что задача эквивалентна задаче логистической регрессии.

3 Эксперименты

3.1 Сравнение работы алгоритмов

В первом эксперименте сравнивается работа алгоритмов полного и стохастического градиентного спуска. На графиках 1, 2 приведены зависимости функции потерь и точности (accuracy) от времени работы и от итерации для обоих алгоритмов. Для стохастического алгоритма рассматривались следующие несколько начальных приближений: нулевой вектор весов, единичный, а также векторы, заполненные значениями 0.25, 0.5 и 0.75.

Из графиков видно, что стохастический спуск быстрее всего сходится и дает лучшее качество при нулевом начальном приближении. Это можно объяснить тем, что алгоритм штрафует за большую норму вектора весов, и поэтому нулевое начальное приближение оказывается ближе к искомому решению, чем остальные. Также из графиков можно сделать вывод, что полный градиентный спуск сходится более устойчиво, чем стохастический, но гораздо медленнее.

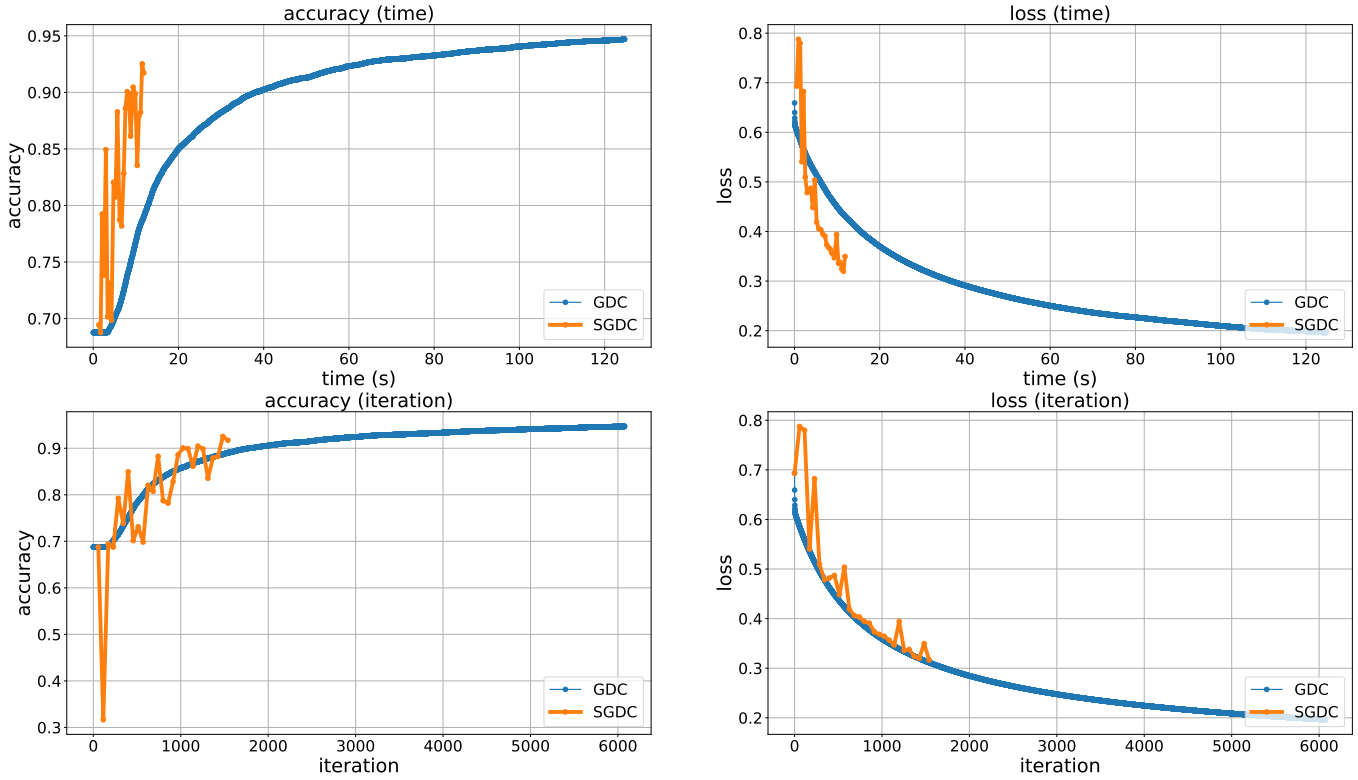

Рис. 1: SGDC vs. GDC

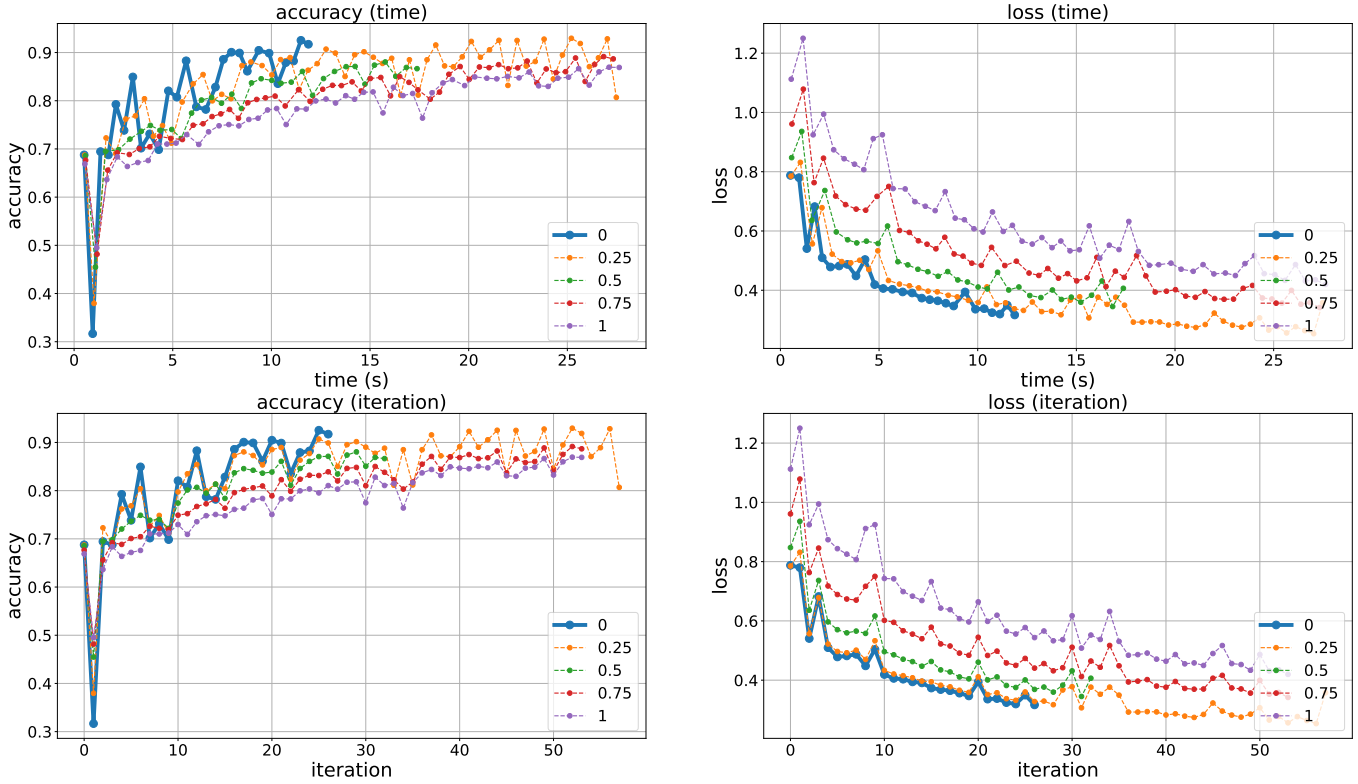

Рис. 2: SGDC with different w0

3.2 Подбор параметров для бинарной классификации

В таблицах 1, 2, 3, 4, 5, 6 показана зависимость качества модели от параметров шага градиентного спуска *step_alpha* и *step_beta*. Лучшее значение accuracy для полного алгоритма равно 0.971 и достигается при $\alpha = 1000, \beta = 0.5$ и при $\alpha = 100, \beta = 0.1$. Лучшее значение accuracy для стохастического алгоритма равно 0.935 и достигается также при $\alpha = 1000, \beta = 0.5$. При таких значениях параметров шага алгоритм начинает сходиться быстро засчет большого α и чем ближе он подходит к решению, тем меньше становится шаг и тем точнее получается результат. Также в целом можно заметить, что при больших значениях β лучшее качество достигается при больших значениях α.

Таблица 1: $\beta = 0$

α	**loss, full**	**accuracy, full**	**loss, stochastic**	**accuracy, stochastic**
0.1	0.359	0.857	0.608	0.688
1	**0.196**	0.947	**0.563**	0.692
100	0.239	**0.963**	14.429	**0.754**
1000	47.938	0.704	124.642	0.709

Таблица 2: $\beta = 0.5$

α	**loss, full**	**accuracy, full**	**loss, stochastic**	**accuracy, stochastic**
0.1	0.618	0.688	0.611	0.688
1	0.569	0.688	**0.609**	0.688
100	0.172	0.956	2.134	0.701
1000	**0.137**	**0.971**	1.815	**0.935**

Таблица 3: $\beta = 0.1$

α	**loss, full**	**accuracy, full**	**loss, stochastic**	**accuracy, stochastic**
0.1	0.457	0.759	0.601	0.688
1	0.239	0.929	**0.424**	0.786
100	**0.133**	**0.971**	2.808	**0.902**
1000	15.059	0.716	59.487	0.745

Таблица 4: $\beta = 1$

α	**loss, full**	**accuracy, full**	**loss, stochastic**	**accuracy, stochastic**
0.1	0.672	0.688	0.681	0.688
1	0.616	0.688	0.622	0.688
100	**0.383**	0.839	**0.524**	0.761
1000	0.1782	**0.941**	4.706	**0.789**

Таблица 5: $\beta = 2$

α	**loss, full**	**accuracy, full**	**loss, stochastic**	**accuracy, stochastic**
0.1	0.687	0.688	0.687	0.688
1	0.647	0.688	0.651	0.688
100	0.552	0.688	**0.559**	0.725
1000	**0.251**	**0.879**	2.683	**0.741**

Таблица 6: $\beta = 0.01$

α	**loss, full**	**accuracy, full**	**loss, stochastic**	**accuracy, stochastic**
0.1	0.369	0.851	0.611	0.688
1	**0.199**	0.945	**0.349**	**0.859**
100	0.217	**0.967**	9.324	0.842
1000	55.234	0.694	110.715	0.708

random_seed	**accuracy**	**loss**
2	0.924	0.191
10	0.947	0.175
50	0.948	0.193
150	0.949	0.175
1000	**0.954**	**0.173**

Таблица 7: Accuracy, loss (*random_seed*)

batch_size	**accuracy**	**loss**
1	0.929	0.188
5	**0.955**	**0.171**
10	0.923	0.188
20	0.933	0.181
50	**0.953**	**0.171**
100	0.945	0.173

Таблица 8: Accuracy, loss (*batch_size*)

В таблице 7 показано, какое получается качество модели при различной степени случайности выбора объектов в стохастическом алгоритме. Как видно из таблицы, чем больше значение *random_seed*, тем лучше обучается модель. Это может объясняться тем, что при большей степени случайности алгоритм меньше учитывает, какие именно объекты попали в подвыборку на текущем шаге, и засчет этого лучше обобщает.

В таблице 8 показано, как зависит качество модели от размера подвыборки в стохастическом алгоритме. Из таблицы можно сделать вывод, что качество не сильно зависит от этого параметра. В таблицах 9 и 10 показана зависимость качества модели от параметров величины шага при размерах подвыборки 10 и 100. Можно сделать вывод, что наше наблюдение о том, что при больших значениях β лучшее качество достигается при больших значениях α, остается справедливым и при этих значениях размера подвыборки.

	$batch\_size = 10$		$batch\_size = 100$	
α	**loss**	**accuracy**	**loss**	**accuracy**
0.1	0.603	0.688	0.611	0.688
1	**0.480**	0.754	0.559	0.688
100	0.820	**0.953**	**0.339**	**0.953**
1000	17.599	0.908	22.536	0.662

Таблица 9: $\beta = 0$

	$batch\_size = 10$		$batch\_size = 100$	
α	**loss**	**accuracy**	**loss**	**accuracy**
0.1	0.677	0.688	0.673	0.688
1	0.629	0.688	0.623	0.688
100	**0.415**	0.806	0.455	0.742
1000	0.627	**0.875**	**0.233**	**0.925**

Таблица 10: $\beta = 1$

3.3 Сравнение трех методов многоклассовой классификации

Теперь рассмотрим задачу многоклассовой классификации. В приведенной реализации есть три метода ее решения: one-vs-all, all-vs-all и мультиномиальная регрессия. Сравним их между собой. К плюсам методов one-vs-all и all-vs-all можно отнести то, что они оба простые и легко реализуются, а также за их основу можно брать любой бинарный классификатор. Но при этом метод one-vs-all хорошо работает в случаях, когда каждый класс отделим от остальных, и плохо работает в других случаях. В методе all-vs-all итоговое решение принимается на основе подсчета числа голосов всех алгоритмов. Из-за того, что число голосов у разных классов может оказаться одинаковым, ответ зависит от способа выбора класса в таких случаях. Это делает метод all-vs-all неустойчивым. Также число классификаторов, равное $\frac{k(k-1)}{2}$, может быть довольно большим, и придется обучить и хранить много векторов в памяти. Так же в этих двух методах обучение классификаторов происходит независимо друг от друга, и следовательно нет общей функции потерь, по которой можно оценивать качество модели.

Третий способ решения задачи многоклассовой классификации — это обобщение логистической регрессии на случай $k \geq 2$, называемое мультиномиальной регрессией. К плюсам этого метода можно отнести те же плюсы, что есть у логистической регрессии: он корректно оценивает вероятности принадлежности объектов к классам, может быть обучен с помощью известных методов, а в случае $k = 2$ сводится к бинарной логистической регрессии, как показано в теоретической части.

На рисунке 3 изображена работа этих методов для случая классификации объектов, описываемых 2 признаками, на 3 класса.

4 Эксперименты на датасете 20newsgroups

Здесь мы попробуем решить задачу многоклассовой классификации для текстов из датасета 20newsgroups с помощью алгоритма мультиномиальной регрессии.

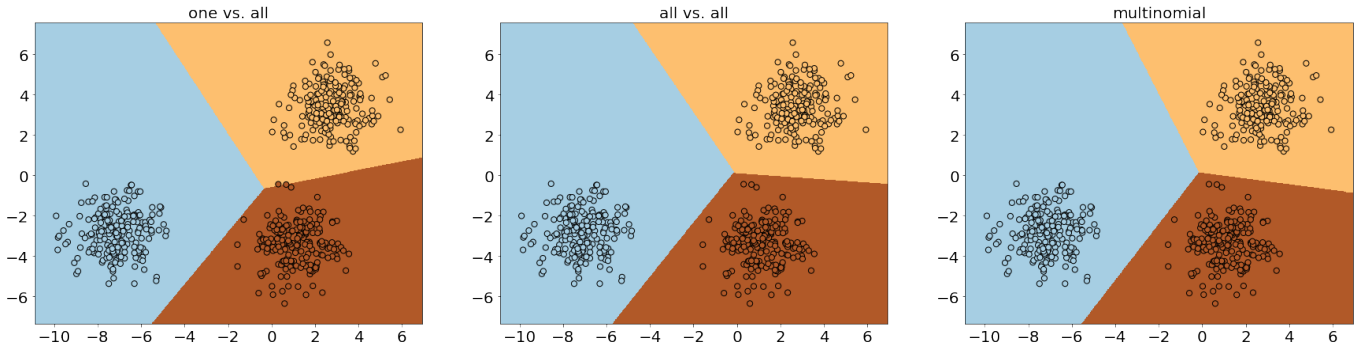

Рис. 3: Multiclass

4.1 Подбор параметров

Рассмотрим два способа векторных представлений текстов:

1. Вектор из счетчиков частот слов
2. Вектор из счетчиков tf-idf

В таблицах 11, 12 и 13 приведены результаты экспериментов, в которых тексты были представлены этими способами. Видно, что представление текстов с использованием tf-idf дает значительно лучшие результаты accuracy на отложенной выборке, поэтому в дальнейшем будет использовать его.

α	**accuracy**
0.1	0.615
1	**0.665**
10	0.653
100	0.645

Таблица 11: Без tf-idf, $\lambda = 0$, maxiter=1000

λ	**accuracy**
0	**0.645**
10^{-6}	0.041
10^{-1}	0.106
1	0.053
100	**0.641**

Таблица 12: Без tf-idf, $\alpha = 100$, maxiter=1000

α	**accuracy**
1	0.589
10	0.714
100	**0.744**

Таблица 13: С tf-idf, $\lambda = 0$, maxiter=1000

При $\alpha = 100$ и $\lambda = 10^{-6}$ accuracy получилось равным 0.744, как и при $\lambda = 0$, поэтому в дальнейшем было выбрано $\lambda = 10^{-6}$, $\alpha = 100$.

4.2 Применение алгоритма

После обучения полного градиентного спуска с подобранными в предыдущем эксперименте параметрами на коллекции 20newsgroups, сделаем предсказание на тестовой выборке и проанализируем результаты. Полученная accuracy равна 0.684. Матрица ошибок приведена на рисунке 4.

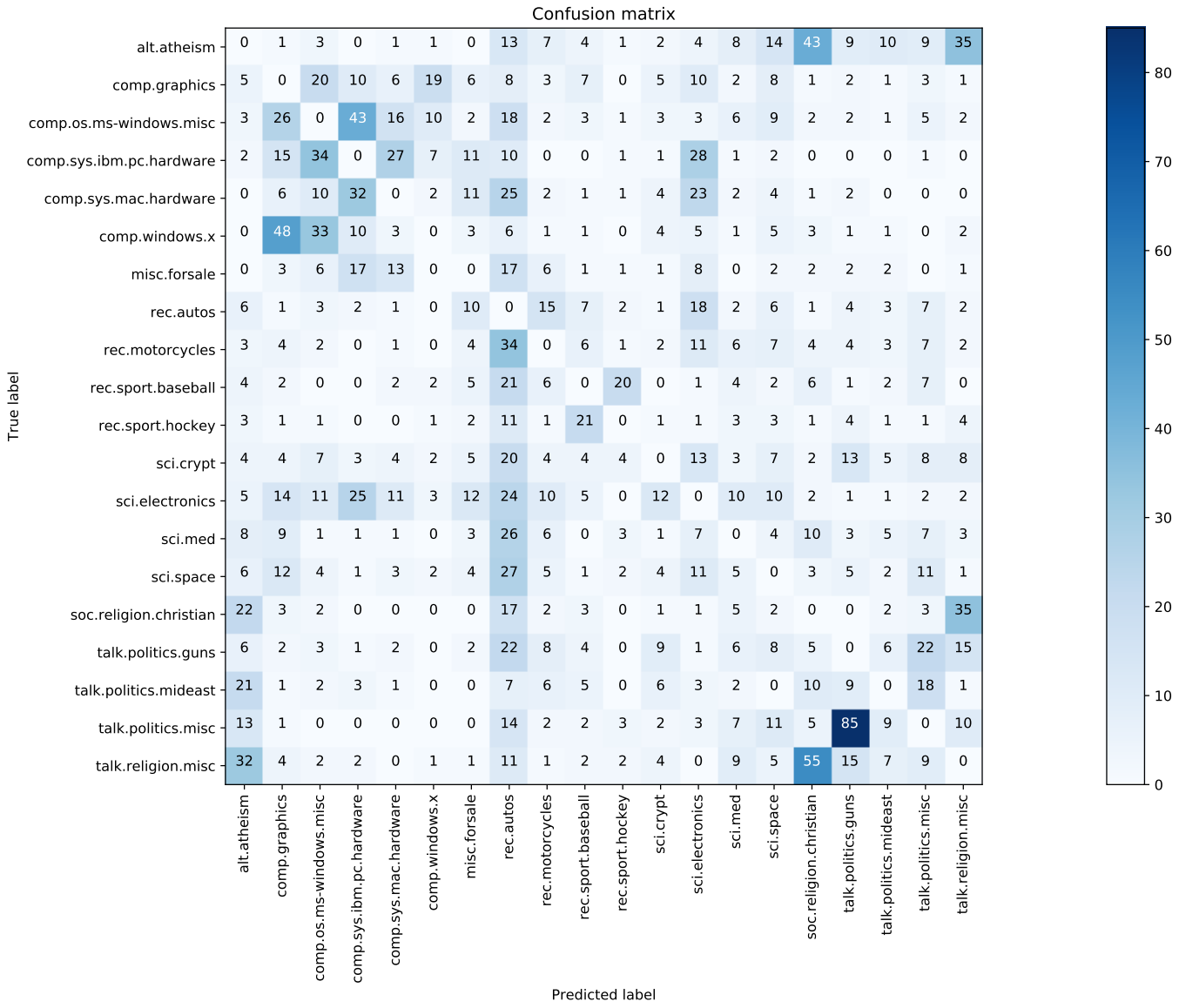

Рис. 4: Матрица ошибок при применении tf-idf

Рассмотрим теперь 2 примера ошибочно классифицированных текстов (таблицы 14 и 15. Первый текст действительно содержит слова $headers, displays, uncompressed$, которые могут указывать на его отношение к компьютерной графике. При этом при предобработке текстов слово x-face, которое должно повлиять на отнесение текста к классу $comp.windows.x$, было разбито на части x и $face$, которые по отдельности могут встречаться в очень разных по тематике текстах. Поэтому алгоритм мог ошибся при классификации этого документа. Второй текст просто очень маленький и не содержит слов, указывающих на его класс.

i m not familiar at all with the format of these x face thingies but after seeing them in some folks headers i ve got to see them and maybe make one of my own i ve got dpg view on my linux box which displays uncompressed x faces and i ve managed to compile un compface too but now that i m looking for them i can t seem to find any x face s in anyones news headers could you would you please send me your x face header i know i ll probably get a little swamped but i can handle it i hope

Таблица 14: True label is *comp.windows.x*, predicted label is *comp.graphics*

in a word yes

Таблица 15: True label is *alt.atheism*, predicted label is *talk.politics.misc*

4.3 Применение лемматизации и стемминга

Применение лемматизации и стемминга может помочь сократить словарь, то есть понизить размерность пространства признаков. В таблице 16 приведены результаты. Из нее видно, что применение стемминга ухудшило качество предсказаний, а применение лемматизации улучшило. При этом в обоих случаях размерность пространства признаков была понижена, но про стемминге различных слов осталось меньше, чем при лемматизации. Поэтому падение качества может быть связано с потерей большой части информации при стемминге.

алгоритм	**accuracy**	**время (мин)**	**размерность**
GDC	0.684	13	101631
GDC + Stemming	0.681	**11**	**84435**
GDC + Lemmatizing	**0.686**	12	93650

Таблица 16: Исходный алгоритм vs. стемминг vs. лемматизация

На рисунках 5, 6 приведены матрицы ошибок для выполненных предсказаний. Видно, что распределение ошибок практически не изменилось.

4.4 Сокращение словаря

Попробуем добиться ускорения работы алгоритма и улучшения его качества, удалив из коллекции слова, которые встречаются в либо очень часто, либо очень редко. Из таблицы 17 видно, что при лучшая точность достигается при удалении слов, частота которых выше 90%. При этом при лучшем времени, в 10 раз меньшем времени обучения начальной модели, достигается точность 0.677 при дополнительном удалении слов, которые встретились реже 5 раз. Если вместо этого удалять стоп-слова, accuracy итогового предсказания на тесте составит 0.649.

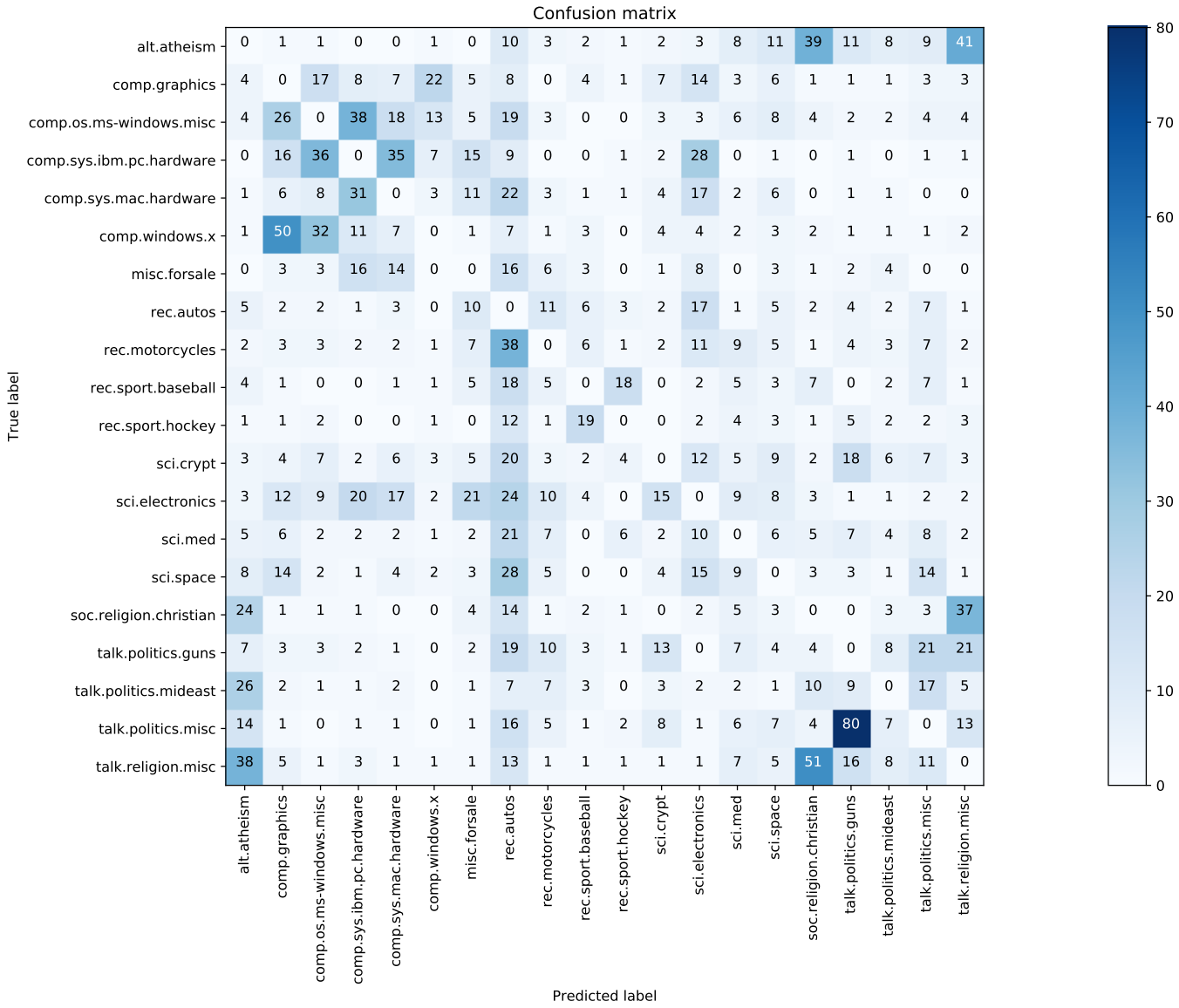

Рис. 5: Матрица ошибок при применении стемминга

min_df (**количество**)	*max_df* (%)	**accuracy**	**время (мин)**
0	99	0.685	12
0	90	**0.685**	11
0	80	0.684	11
10	90	0.668	1
5	90	0.677	**1**

Таблица 17: Зависимость accuracy и времени работы от количества удаленных слов

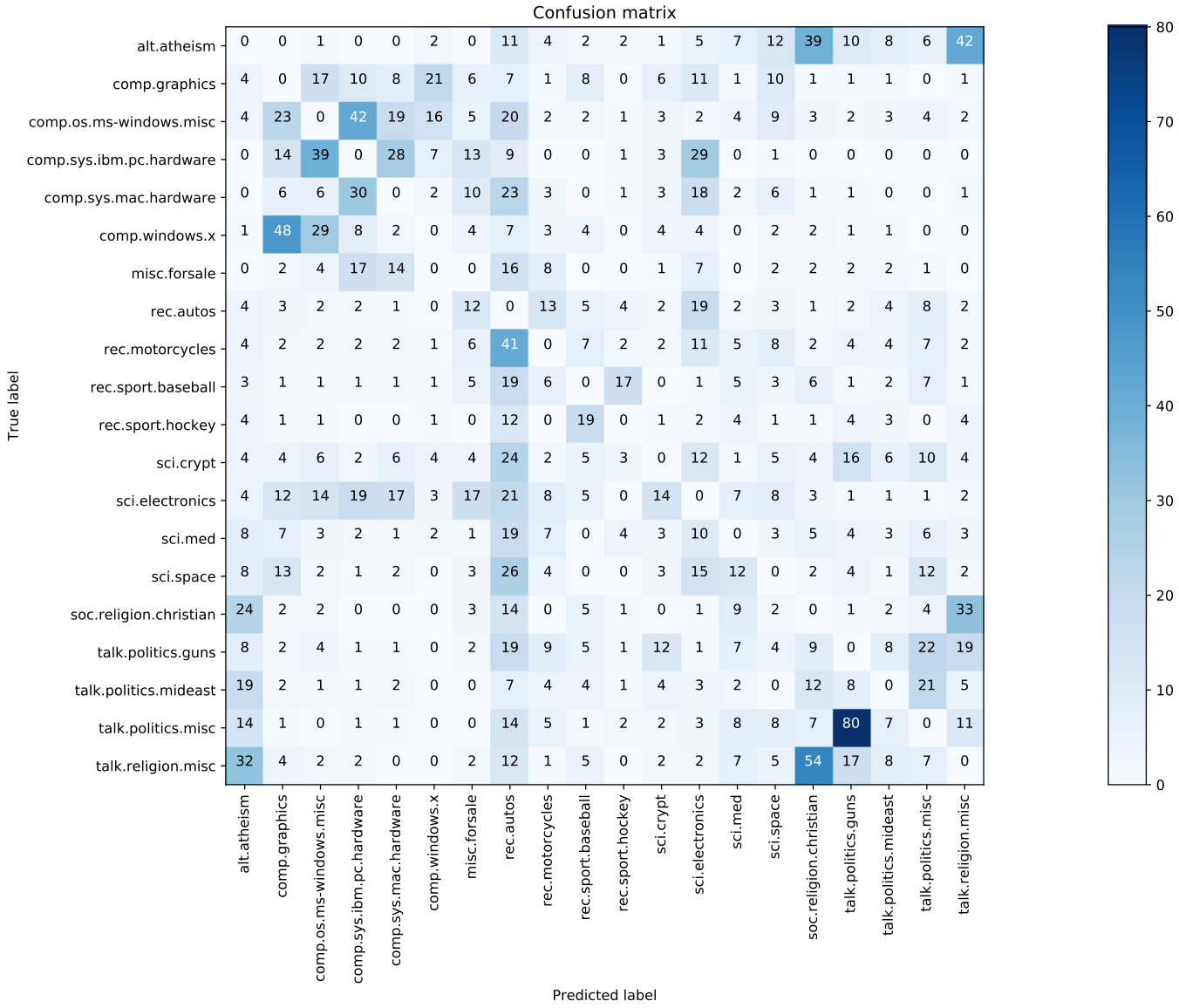

Рис. 6: Матрица ошибок при применении лемматизации