Исследование работы линейных моделей

Тыцкий Владислав

Ноябрь 2020

Градиент в логистической регрессии

Пусть $X \in R^{N \times F}$ – матрица объектов-признаков, $y \in \{-1, 1\}^N$ – метки соответствующих объектов, $w \in R^F$ – вектор весов, x_i – i-ый объект y_i – метка класса i-ого объекта, T - стохастическая подвыборка($|T|$ и T будут обозначаться T в зависимости от контекста). Везде будем считать, что добавлен константый признак, но соответствующий вес не регуляризуется.

Дана задача оптимизации:

$$Q(X, y, w) = \mathcal{L}(X, y, w) + \frac{\lambda}{2}||w||_2^2 \to \min_w$$
$$\mathcal{L}(X, y, w) = \frac{1}{T}\sum_{i=1}^{T} \log(1 + \exp(-y_i\langle x_i, w\rangle)), T \leq N$$

Для решения этой задачи с помощью градиентных методов необходимо знать градиент функционала $Q(X, y, w)$

$$dQ = d\mathcal{L} + \frac{\lambda}{2}d\langle w, w\rangle = d\mathcal{L} + \lambda\langle w, dw\rangle$$
$$d\mathcal{L} = \frac{1}{T}\sum_{i=1}^{T} \frac{d(\exp(-y_i\langle x_i, w\rangle))}{1 + \exp(-y_i\langle x_i, w\rangle)} = \frac{1}{T}\sum_{i=1}^{T} \frac{\exp(-y_i\langle x_i, w\rangle)d\langle -y_i x_i, w\rangle}{1 + \exp(-y_i\langle x_i, w\rangle)} =$$
$$-\frac{1}{T}\sum_{i=1}^{T} \frac{\langle y_i x_i, dw\rangle}{1 + \exp(y_i\langle x_i, w\rangle)}$$

Заметим, что $dQ = \langle \nabla Q, dw\rangle$. Окончательно получаем:

$$\nabla Q(X, y, w) = \lambda w - \frac{1}{T}\sum_{i=1}^{T} \frac{y_i x_i}{1 + \exp(y_i\langle x_i, w\rangle)}$$

Случай для K классов

Пусть $X \in R^{N \times F}$ – матрица объектов-признаков, $y \in K^N$ – метки соответствующих объектов, где $K = \{1 \dots k\}$ – множество классов, $w_i \in R^F$ – вектор весов

соответствующий k-ому классу, x_i – i-ый объект y_i – метка класса i-ого объекта соответственно, T - стохастическая подвыборка($|T|$ и T будут обозначаться T в зависимости от контекста).

Дана задача оптимизации — максимизация правдоподобия:

$$Q(X,y,w) = -\frac{1}{T}\sum_{i=1}^{T}\log \mathbb{P}(y_i|x_i) + \frac{\lambda}{2}\sum_{k=1}^{K}||w_k||_2^2 \to \min_{w_1 \dots w_k}$$

$$\mathbb{P}(y=c|x) = \frac{\exp\langle w_c, x\rangle}{\sum_{k=1}^{K}\exp\langle w_k, x\rangle}$$

Найдем градиент по w_m.

$$dQ(X,y,w) = d(-\frac{1}{T}\sum_{i=1}^{T}\log \mathbb{P}(y_i|x_i)) + d(\frac{\lambda}{2}\sum_{k=1}^{K}||w_k||_2^2) = -\frac{1}{T}d(\sum_{i=1}^{T}\log \mathbb{P}(y_i|x_i)) + \lambda(w_m, dw_m)$$

$$d(\sum_{i=1}^{T}\log \mathbb{P}(y_i|x_i))_{w_m} = \sum_{i=1}^{T} d(\log\exp\langle w_{y_i}, x_i\rangle) - \sum_{i=1}^{T} d(\log\sum_{k=1}^{K}\exp\langle w_k, x_i\rangle) =$$

$$= \sum_{\substack{i:y_i \leftrightarrow w_m \\ i\in T}} \langle x_i, dw_m\rangle - \sum_{i=1}^{T}\frac{\exp\langle w_m, x_i\rangle\langle x_i, dw_m\rangle}{\sum_{k=1}^{K}\exp\langle w_k, x_i\rangle}$$

Отсюда получаем:

$$\nabla Q_{w_m} = \lambda w_m + \frac{1}{T}\sum_{i=1}^{T}\frac{\exp\langle w_m, x_i\rangle x_i}{\sum_{k=1}^{K}\exp\langle w_k, x_i\rangle} - \frac{1}{T}\sum_{\substack{i:y_i \leftrightarrow w_m \\ i\in T}} x_i$$

Эквивалентность бинарной логистической регрессии и мультиномиальной при K=2

Доказательство.
Пусть w_+ — вектор весов соответствующий первому классу, а w_- — -1 классу.
Введем $w = w_+ - w_-$. Рассмотрим задачу мультиномиальной регрессии при K=2:

$$Q(X,y,w) = -\frac{1}{T}\sum_{i=1}^{T}\log\frac{\exp\langle w_c, x\rangle}{\sum_{k=1}^{K}\exp\langle w_k, x\rangle} == -\frac{1}{T}\sum_{\substack{i:y_i=w_+ \\ i\in T}}\log\frac{\exp\langle w_+, x_i\rangle}{\exp\langle w_+, x_i\rangle + \exp\langle w_-, x_i\rangle}$$

$$-\frac{1}{T}\sum_{\substack{i:y_i=w_- \\ i\in T}}\log\frac{\exp\langle w_-, x_i\rangle}{\exp\langle w_+, x_i\rangle + \exp\langle w_-, x_i\rangle} = -\frac{1}{T}\sum_{\substack{i:y_i=w_+ \\ i\in T}}\log\frac{1}{1+\exp\langle -w, x_i\rangle} -$$

$$-\frac{1}{T}\sum_{\substack{i:y_i=w_- \\ i\in T}}\log\frac{1}{\exp\langle w, x_i\rangle + 1} == \frac{1}{T}\sum_{i=1}^{T}\log(1+\exp(-y_i\langle w, x_i\rangle))$$

То есть функции потерь для бинарной логрегрессии и мультиномиальной регрессии при K=2 эквиваленты.
ч.т.д

Задание №1

Предобработка документов делается с помощью модуля re и метода apply из pandas.

```
    X_train_df.apply(lambda x: re.sub("[^a-zA-z0-9]", " ", x.lower()))
    X_test_df.apply(lambda x: re.sub("[^a-zA-z0-9]", " ", x.lower()))
```

Листинг 1: Clear documents

Задание №2

Использование CountVectorizer для представления слов с помощью bag of words.

```
vectorizer = CountVectorizer(lowercase=True, min_df=50)
X_train_v = vectorizer.fit_transform(X_train["comment_text"])
X_test_v = vectorizer.transform(X_test["comment_text"])
```

Листинг 2: Vectorizer

Min_df = 50 имеет под собой основание. "Оскорбительные" слова в данном датасете обычно встречаются чаще 100 раз. Листинг ниже (3) демонстрирует код для подсчета.

```
text = X_train["comment_text"]
count = 0
count_bad = 0
for i in range(text.size):
    if text[i].find("very very bad word") != -1:
        count += 1
        if X_train["is_toxic"][i]:
            count_bad += 1
```

Листинг 3: Count bad words

Задание №3, №4, №5

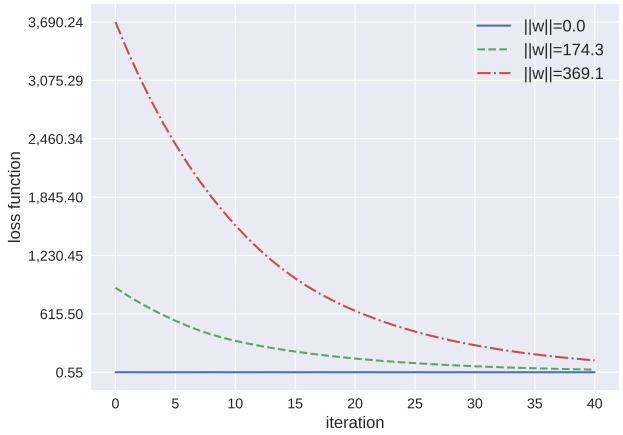
Исследуем как ведет себя метод (стохастического) градиентного спуска. Я посчитал, что 3, 4, 5 задания можно совместить в одно большое задание — легче прослеживается логика повествования. [1]

Начальная инициализация

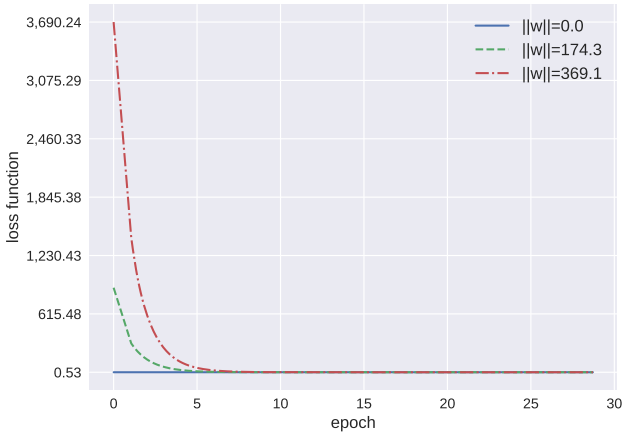
Интересно взглянуть как влияет начальная инициализация весов на функцию потерь (Таблица 1) [2]

[1] Все графики строились на весьма урезанной по количеству признаков выборке (2300). Это сделано для того, чтобы вычислительной мощности компьютера Тыцкого.В.И. хватило построить их за разумное время.

[2] Я так и не понял как поменять тип caption с Таблицы на Рисунок



GD



SGD

Таблица 1: Зависимость функции потерь от начальной инициализации и итераций

Можно заметить, что вне зависимости от начальной инициализации спустя небольшое количество итераций(эпох) функция потерь становится примерно одинаковой.

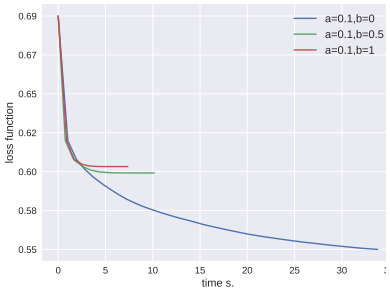
В других экспериментах будем использовать нулевой вектор в качестве начальной инициализации

Параметры задающие скорость обучения

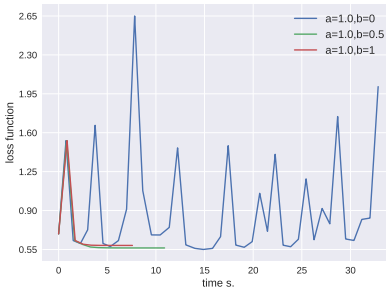
В экспериментах используется линейный классификатор, который вычисляет новый вес w^{i+1} по формуле:

$$w^{i+1} = w^i - \eta \nabla Q,\ \eta = \frac{\alpha}{i^\beta}$$

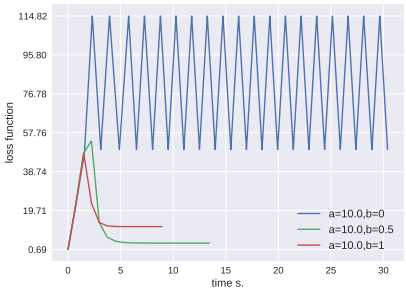
где Q - градиент функции потерь.
Необходимо понять как влияют гиперпараметры α и β на алгоритм. В Таблице 2 представлены графики зависимости функции потерь от параметров α и β.



$\alpha = 0.1$



$\alpha = 1.0$



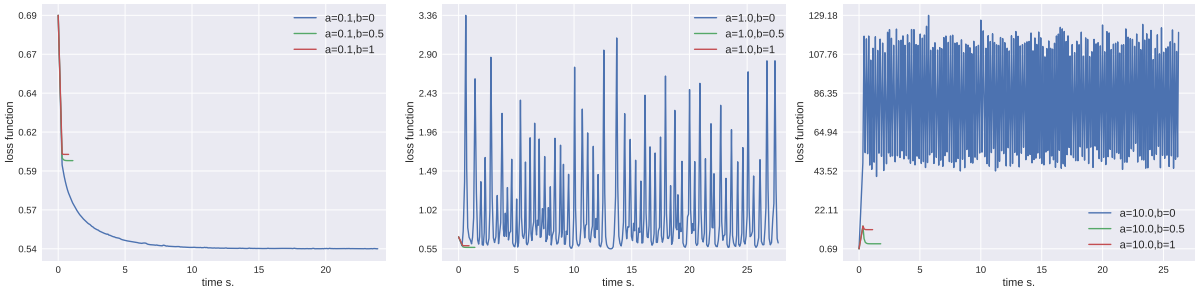
$\alpha = 10.0$

Таблица 2: Зависимость функции потерь от alpha и beta для GD

Заметим как быстро алгоритм останавливается при $\beta > 0$. Если и имеет смысл использовать ненулевые β, то только для больших значений параметра α. В то же время при сильно больших α и $\beta = 0$ градиентный спуск может не спуститься в точку экстремума, что плохо сказывается на качестве модели.

В дальнейших экспериментах будем брать $\alpha \approx 0.5$ и $\beta = 0$.

Для стохастического градиентного спуска картина такая же (Таблица 3) за исключением того, что при больших α поведение еще более непредсказуемо.



$\alpha = 0.1$ $\alpha = 1.0$ $\alpha = 10.0$

Таблица 3: Зависимость функции потерь от alpha и beta для SGD

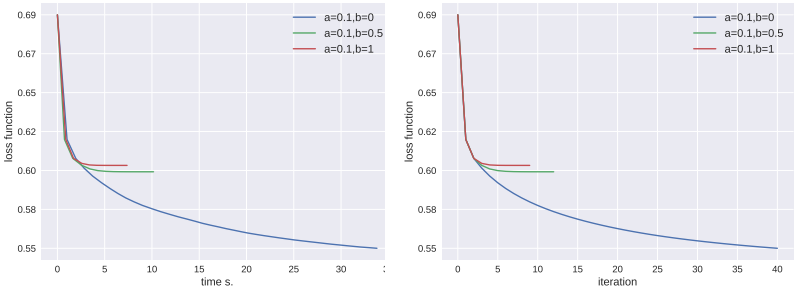
Сравнение GD и SGD в скорости

На предыдущих графиках(Таблица 2, Таблица 3) можно пронаблюдать скорость обучения GD и SGD классификатора. GD делает более "точные" шаги градиентного спуска, но скорость выполнения этого шага довольно медленная. Хоть SGD менее точен, но из-за того, что он делает больше шагов градиентного спуска за эпоху, он быстрее сходится к локальному экстремуму. Заметим, что выбор в пользу SGD сделан конкретно для данного датасета. Для других задач поведение GD и SGD может быть совсем иным.

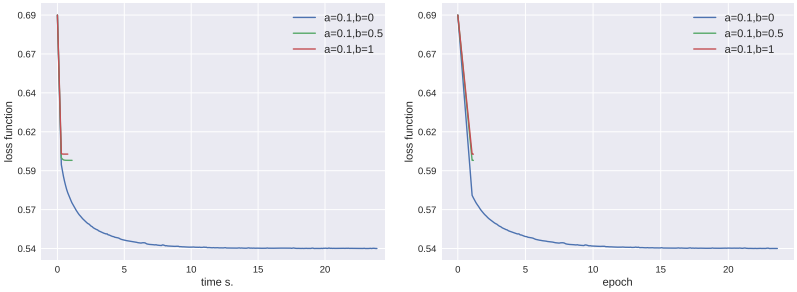
В дальнейших экспериментах будем использовать SGD классификатор.

Время на одну итерацию (эпоху)

Важно оценить время работы классификатора в зависимости от итераций (эпох), чтобы подобрать оптимальное по соотношению качество скорость итераций (эпох). Таблица 4. Одна итерация (эпоха) делается чуть меньше, чем за секунду.



Time iterations GD



Time epoch SGD

Таблица 4: Зависимость между временем и итерацией (эпохой)

Размер батча для SGD

В случае выбора SGD в качестве основного алгоритма необходимо понять какой размер батча (подвыборки) оптимален. Важна и скорость работы, и точность шагов градиентного спуска. Справа представлен график Рис.1.

Можно заметить, что уже при размерах батча 2000 скорость сходимости и точность шагов градиентного спуска приемлемы — не возникает больших скачков, как у размера 500 и скорость гораздо выше, чем для размера 10000 или всей выборки.

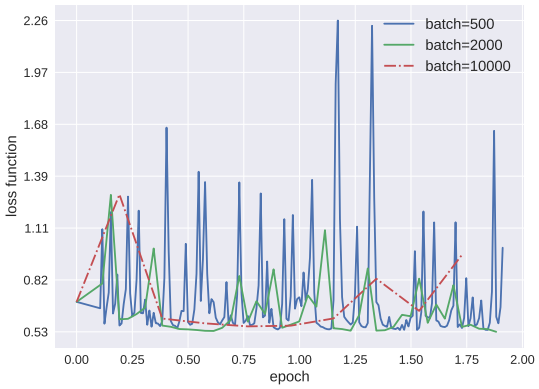
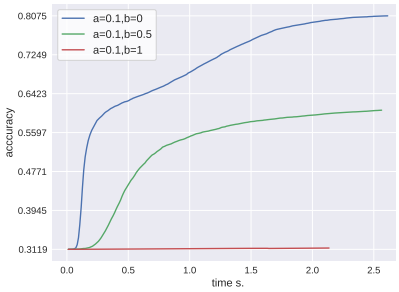


Рис. 1: Зависимость функции потерь от размера батча

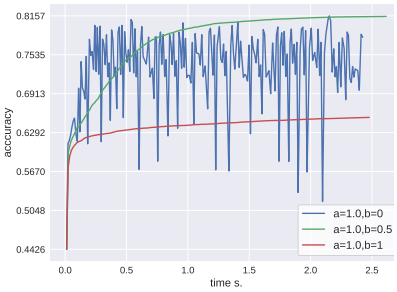
В будущих экспериментах будет использоваться размер батча 2000-3000.

Качество алгоритма

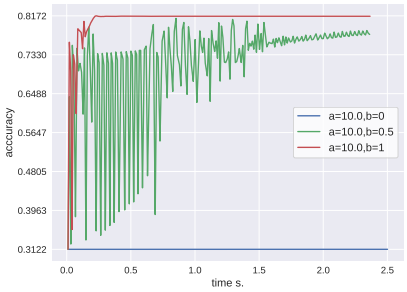
Уменьшение функции потерь — не самое важное для нас. Необходимо взглянуть как меняется мера качества в идеале на отложенной выборке.[3] Ниже представлены графики Таблица 5 для SGD классификатора(batch_size=2000, l2_coef=0.1)



$\alpha = 0.1$



$\alpha = 1.0$



$\alpha = 10.0$

Таблица 5: Зависимость функции потерь от alpha и beta для GD

Лучшее качество на данном этапе экспериментов для тестовой выборки: **roc-auc– 0.928, accuracy– 0.860**

Обучение проводилось на выборке, векторизованной Bag of words с $min\_df = 50$, с помощью SGD классификатора с параметрами:
$batch\_size = 2000, \alpha = 0.5, \beta = 0, max\_iter = 1500, l2\_coef = 0.01, tolerance = 1e-7, fit\_intercept = True$.

Задание №6

При обработке текстов возникает проблема — одно и то же слово может присутствовать в коллекции в разных формах (например play, played). Все слова стоит привести в начальную форму – это может существенно уменьшить признаковое пространство и улучшить качество модели. Возьмем за основу алгритм лемматизации слов. Ниже представлен Листинг 4, демонстрирующий применение алгоритма лемматизации с помощью библиотеки nltk.

```
# https://webdevblog.ru/podhody-lemmatizacii-s-primerami-v-python/
def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)


```

[3] Не стал делать отложенную выборку и замерил на обучающей

```
lem= WordNetLemmatizer()

X_train.apply(lambda x: " ".join([lem.lemmatize(w, get_wordnet_pos(w))
        for w in nltk.word_tokenize(x)]))
X_test.apply(lambda x: " ".join([lemm.lemmatize(w, get_wordnet_pos(w))
        for w in nltk.word_tokenize(x)]))
```

Листинг 4: Lemmatizer

Также перед лемматизацией удалим стоп-слова.

Размерность признакового пространства снизилась с 3930 до 3090. На скорость работы уменьшение размерности несильно повлияло[4]. Accuracy и roc-auc на тестовой выборке существенно возросло: **roc-auc – 0.935, accuracy – 0.865**

Обучение проводилось на лемматизированной выборке, векторизованной Bag of words с $min\_df = 50, stop\_words = english$, с помощью SGD классификатора с параметрами:
$batch\_size = 2000, \alpha = 0.8, \beta = 0, max\_iter = 1800, l2\_coef = 0.01, tolerance = 1e-7, fit\_intercept = False$.

Задание №7

Посмотрим как влияет предобработка данных на качество, а конкретно параметры min_df, max_df. В Таблице 6 представлены зависимости времени, размерности, качества (BoW, TF-IDF) от min_df, max_df.

		shape	time s.	accuracy BoW	accuracy TFIDF
max_df = max	min_df = 10	11000	23	0.851	0.741
	min_df = 100	2080	7.5	0.850	0.831
	min_df = 1000	150	2.0	0.784	0.784
min_df = 50	max_df = 300	2957	8.27	0.723	0.698
	max_df = 800	3452	9.59	0.801	0.725
	max_df = max	3660	10.79	0.848	0.766

Таблица 6: Зависимость некоторых величин от min_df, max_df и типов векторизации

Размерность пространства сильно увеличивается, если min_df близко к нулю, но качество в свою очередь несильно меняется. Как было упомянутно в начале исследования, обычно "плохие" слова встречаются чаще, чем 100 раз на коллекцию, поэтому уменьшение min_df приводит только к увеличению размерности и времени работы.

Max_df неразумно ставить не максимальным т.к. мы можем потерять важную информацию, а самые распространенные слова (стоп-слова) английского языка мы предварительно убрали.

[4] в виду того, что и так размерность не самая большая

Интересно, что качество после обработки с помощью TF-IDF стабильно хуже, чем BoW. **Лучший алгоритм показал roc-auc – 0.935, accuracy – 0.853 на тестовой выборке.(TF-IDF)**

Задание №8

Самый лучший алгортим показал **auc-roc – 0.937, accuracy – 0.876 на тестовой выборке**. Использовался лемматизированный BoW с 2-gramm, стоп-словами, и mid_df=50.

Параметры алгоритма: batch_size=2000, step_alpha=0.4, step_beta=0, tolerance=1e-7, max_iter=3000, l2_coef=0.0001, fit_intercept=True

Проанализируем ошибки алгоритма:

False positive

1. "i will burn you to hell if you revoke my talk page access"
2. "what the hell justin"
3. "black mamba it be ponious snake of the word and but it not kill many people but king cobra kill many people in india"

Для некоторых из FP объектов мог ошибиться даже человек (1),(2)[5], в некоторых документах использовались слова характерные для токсичных комментариев, но на деле это было не так (3).

False negative

Примеры:

1. "hey shithead stop vandilizing article"
2. "i think you should delete the whole fuckin wikipedia"
3. "shut up please this consensus be over"
4. "if ya not still fu k u"
5. "x box 360 sukcs big bumm and like it up the as"

Все ошибки можно разделить на два типа: алгоритм объективно ошибся (1),(2),(3) или в документе были допущены ошибки (намерено или случайно), которые не позволили правильно классифицировать документ (4),(5).

[5]По крайней мере я

Вывод

Алгоритм хорошо научился распознавать токсичные комметарии, в которых нет намеренных ошибок в "плохих" словах, но сложности возникают, когда слова специально меняют – добавляют пробелы, меняют буквы местами, заменяют буквы другими символами. Также алгоритм плохо понимает контекст и ложно относит к токсичным комментарии, которые были на определенную тему(например комментарий про кобр).

Возникает гипотеза, что в разметке могли быть допушены ошибки, потому что некоторые из комментариев явно относятся к токсичным, но в датасете отмечены как не токсичные.