Логистическая регрессия для классификации текстов

Практикум ММП ВМК МГУ, `Алексеев Илья`

Составлено на основе материалов `Находнова Максима`

План

`Логистическая регрессия`

`Обработка естественного языка`

Логистическая регрессия

- Определение и мотивация логистической регрессии
- Алгоритмы GD и SGD
- Вариации GD

Логистическая регрессия

Постановка задачи

Обучающая выборка $X = (x_i, y_i)_{i=1}^{\ell}$, где $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{Y} = \{1, -1\}$.

Как определяется линейная модель классификации?

Линейная модель классификации

Определяется следующим образом:

$$a(x) = \text{sign}\left(\langle w, x\rangle + b\right),$$

где $w \in \mathbb{R}^d$ — вектор весов, $b \in \mathbb{R}$ — сдвиг.

Что такое отступ?

Отступ

Величина $M_i(w, b) = y_i(\langle w, x_i \rangle + b)$ называется отступом (margin) объекта x_i относительно алгоритма $a(x)$.

Основное свойство?

Отступ: основное свойство

Если $M_i(w, b) < 0$, алгоритм допускает ошибку на объекте x_i. Чем больше отступ $M_i(w, b)$, тем более надёжно и правильно алгоритм классифицирует объект x_i.

Обучение

Пусть $\mathcal{L}(M(w.b))$ — монотонно невозрастающая функция, такая что $[M(w.b) < 0] \leqslant \mathcal{L}(M(w.b))$.

$$Q(X, w, b) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}(M_i(w, b)) \to \min_{w,b}$$

Логистическая регрессия

Это линейный классификатор со следующей функцией потерь:

$$\mathcal{L}(M) = \log(1 + \exp(-M)).$$

Отличительное свойство?

Логистическая регрессия: отличительное свойство

Оценивание вероятности принадлежности объекта к каждому из классов:

$$p(y = 1 \,|\, x) = \frac{1}{1 + \exp(-\langle w, x \rangle + b)} = \sigma(\langle w, x \rangle + b)$$

Борьба с переобучением

Один из примеров регуляризации — L_2 регуляризатор:

$$Q(X, w, b) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}(M_i(w, b)) + \frac{\lambda}{2} \|w\|_2^2 \to \min_{w,b}$$

Итог: логистическая регрессия

- постановка задачи
- линейная модель классификации
- отступ (margin)
- логистическая функция потерь
- регуляризация

Градиентный спуск

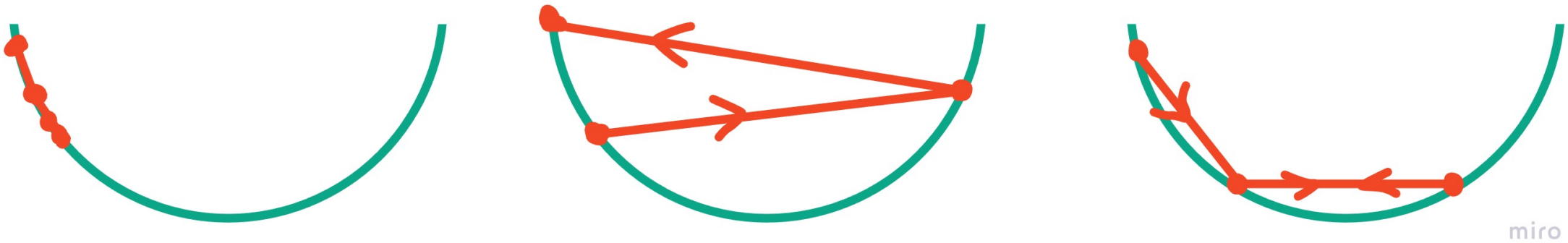
Оптимизация

Метод градиентного спуска для минимизации функционала $Q(X, w)$:

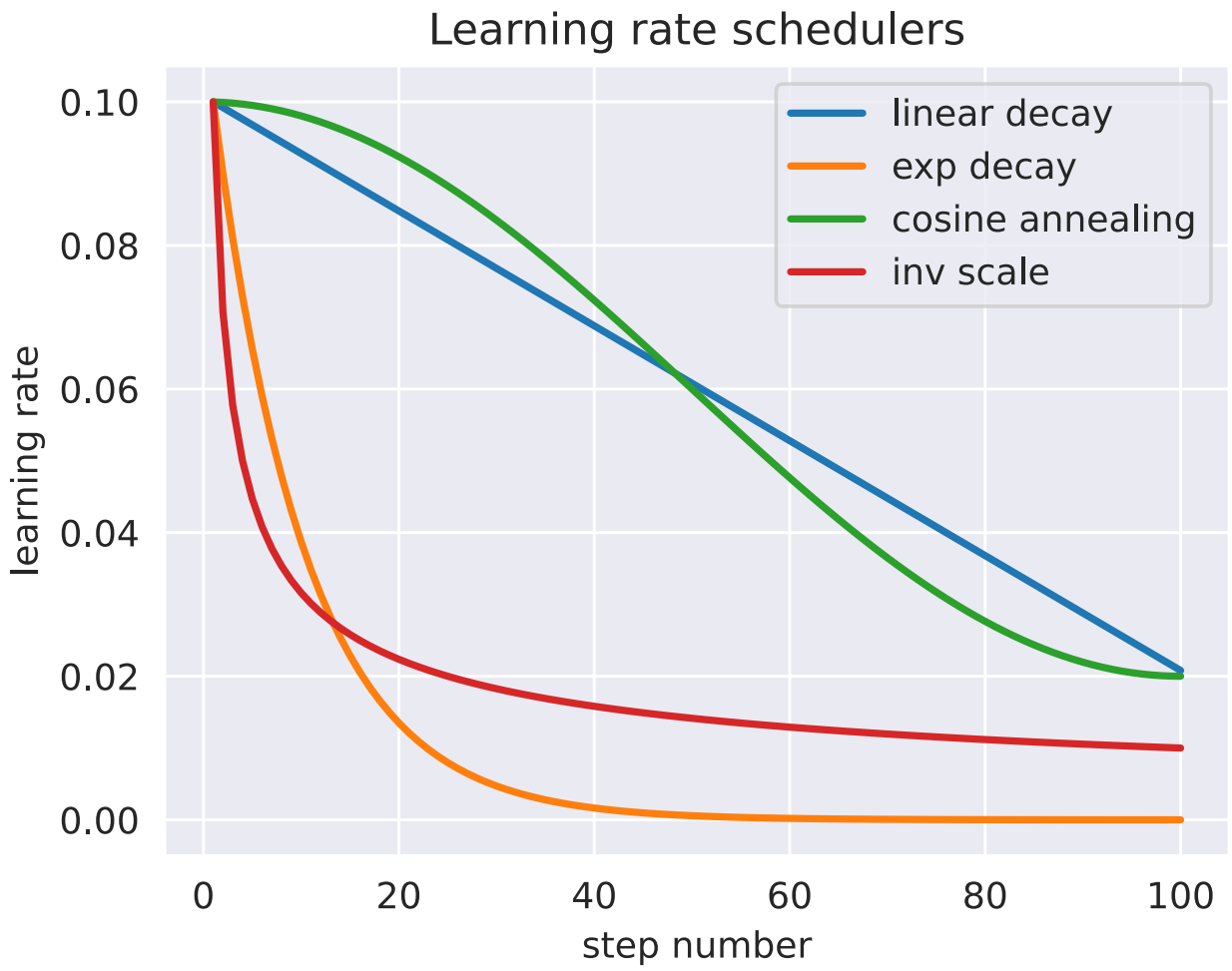
$$w^{(k+1)} = w^{(k)} - \eta_k \cdot \frac{1}{\ell} \sum_{i=1}^{\ell} \nabla_w \mathcal{L}_i$$

Learning Rate

Параметр $\eta_k > 0$ должен быть достаточно мал, чтобы не разойтись, и достаточно большим, чтобы дойти до оптимума.



Learning Rate Schedulers



Learning Rate Schedulers

- exp: $\eta_k = \eta_0 \cdot \gamma^k$
- lin: $\eta_k = \eta_{\max} - (\eta_{\max} - \eta_{\min}) \cdot (k-1)/n$, где n -- макс. число итераций
- cos: $\eta_k = \eta_{\min} + \frac{1}{2} \cdot (\eta_{\max} - \eta_{\min}) \cdot (1 + \cos(\frac{\pi k}{n}))$
- inv scale: $\eta_k = \alpha / k^\beta$, где $\alpha, \beta > 0$

Алгоритм градиентного спуска

```
вход: выборка X
выход: веса w, b
гиперпараметры: MAX_ITER, alpha, beta

1.  w, b := некоторое начальное приближение
2.  для k := 1 до MAX_ITER
3.      gw := dQ(w,b)/dw
4.      gb := dQ(w,b)/db
5.      eta := alpha / k ** beta
6.      w := w - eta * gw
7.      b := b - eta * gb
```

Общий вид алгоритма оптимизации

```
вход: выборка X
выход: веса w

1.  w := Initialize()
2.  пока not Stop(w, X):
3.      R := Oracle(w, X)
4.      w := Update(w, X, R)
```

Oracle

- градиент $\nabla Q(w)$ или $Q'(w) \Rightarrow$ методы первого порядка (GD, SGD)
- гессиан $\nabla^2 Q(w)$ или $Q''(w) \Rightarrow$ методы второго порядка (Newton)
- аппроксимация гессиана $\Rightarrow$ квазиньютоновские методы (L-BFGS)

Update

- GD:

$$w_{k+1} := w_k - \eta_k Q'(w_k)$$

- GD with momentum:

$$w_{k+1} := w_k - \eta_k Q'(w_k) + \beta_k (w_k - w_{k-1})$$

- GD with Nesterov momentum:

$$m_{k+1} := \gamma \cdot m_k + \eta \cdot Q'(w_k - \gamma \cdot m_k)$$
$$w_{k+1} := w_k + m_{k+1}$$

Stop

ε — точность решения (tolerance)

- число итераций
- сходимость параметров: $\|w_{k+1} - w_k\| < \varepsilon$
- сходимость функционала: $|Q(w_k) - Q(w_{k+1})| < \varepsilon$
- ноль градиента: $\|\nabla Q(x)\| < \varepsilon$
- *зазор двойственности: $Q(w_k) - Q(\lambda_k, \mu_k) < \varepsilon$

Initialize

- $w, b \sim \mathrm{Normal}(0, \sigma)$
- $w, b \sim \mathrm{Uniform}[-a, a]$
- $w \sim \mathrm{Normal}(0, \sigma), b = 0$
- $w = 0, b = \mathrm{const}$
- $w_j = \langle f_j, y \rangle / \langle f_j, f_j \rangle$, где y - вектор меток, f_j - столбец признака
- обучение по небольшой случайной подвыборке

Итог: градиентный спуск

- ограничения на learning rate
- learning rate schedulers
- алгоритм GD
- общий вид алгоритма оптимизации
- oracle, update, stop, initialize

Mini-batch SGD

Алгоритм

```
вход: выборка X
выход: веса w, b
гиперпараметры: MAX_ITER, alpha, beta, B

1.  w := некоторое начальное приближение
2.  пока not Stop(w,b)
3.      shuffle(X)
4.      для i in range(0, len(X), B):
5.          batch := X[i:i+B]
6.          R := Oracle(w, batch)
7.          w := Update(w, batch, R)
```

Посмотрим в `sklearn`

Мультиномиальная логистическая регрессия

Теперь $\mathbb{Y} = \{1, \ldots, K\}$. Вероятность k-ого класса можно выразить так:

$$P(y = j \,|\, x) = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)},$$

где $z_k = \langle w_k, x \rangle + b$ выдает сырой скор принадлежности k-му классу (логит).

Обучение производится с помощью метода максимального правдоподобия:

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

Рекомендации по выполнению задания

Grad check

Проверить правильность реализации подсчета градиента можно с помощью конечных разностей:

$$[\bigtriangledown f(w)]_i \approx \frac{f(x + \varepsilon e_i) - f(x)}{\varepsilon}$$

$e_i = (0, 0, \ldots, 0, 1, 0, \ldots, 0)$ — базисный вектор, $\varepsilon > 0$ — небольшое положительное число.

Сравнение GD и SGD

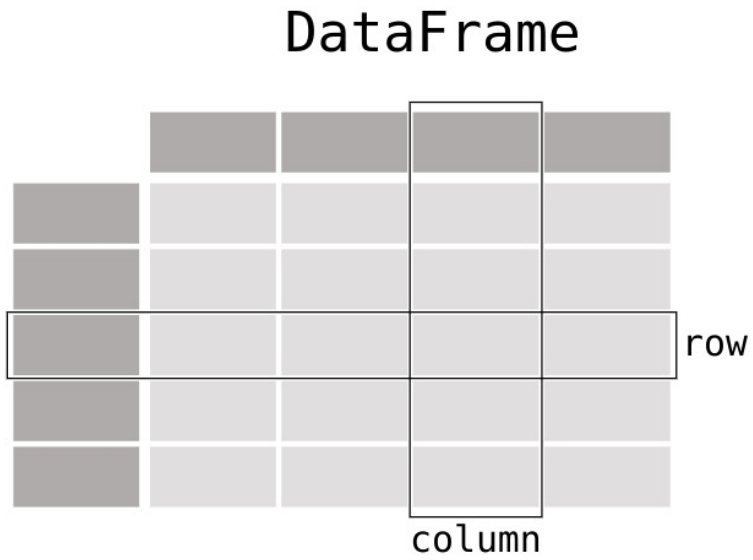
1 итерация GD != 1 итерация SGD, поэтому сравнивать итерации некорректно!

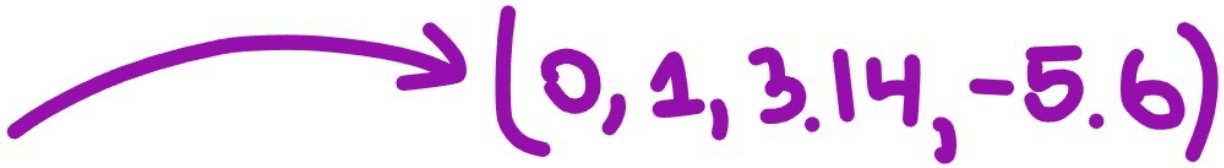
Сравнивать нужно либо по времени, либо по эпохам.

Обработка естественного языка

- векторизация текста
- предобработка текста
- регулярные выражения

Векторизация





[Припев: Big Baby Tape]
Свет мой, зеркало, скажи, покажи мне, кто тут G
Кто был на районе, поставлял барыгам кирпичи?
Передайте мне ключи, передайте мне ключи
Mirror, mirror on the wall, крадусь на них — я вор в ночи

Векторизация

1. Базовые подходы:
    - **Мешок слов (Bag Of Words)**
    - **Tf-Idf**
2. Матричные разложения:
    - LSA/LDA
    - BigARTM
3. Нейросетевые подходы:
    - Word2Vec (Skip-gram, CBoW, FastText, Glove, ...)
    - BERT

Что такое текст?

Обучающая коллекция документов (текстов):

$$D = \{d_1, d_2 \dots d_N\}$$

Документ:

$$d_i = (w_1, w_2, \dots w_n),$$

где w_i — токен (слово) из вокабулярия (словаря) V.

Токенизация — разделение текста на токены, элементарные единицы текста

В большинстве случае токен это слово!

Если пользоваться методом `.split()`, токен — последовательность букв, разделённая пробельными символами

Можно использовать регулярные выражения и модули `re`, `regex`.

Можно использовать специальные токенизаторы, например из `nltk`:

- `RegexpTokenizer`
- `BlanklineTokenizer`
- И ещё около десятка штук

```
from nltk.tokenize import word_tokenize

example = 'Но не каждый хочет что-то исправлять:('
word_tokenize(example, language='russian')
```

```
['Но', 'не', 'каждый', 'хочет', 'что-то', 'исправлять', ':(']
```

```
from nltk.tokenize import sent_tokenize

sent = 'Hey! Is Mr. Bing waiting for you?'
nltk.tokenize.sent_tokenize(sent)
```

```
['Hey!', 'Is Mr. Bing waiting for you?']
```

Bag of Words (Count Vectorizer)

Предположим:

- Порядок токенов в тексте не важен
- Важно лишь сколько раз токен w входит в текст d

Term-frequency, число вхождений слова в текст: $\mathrm{tf}(w, d)$

Векторизация:

$$v(d) = (\mathrm{tf}(w_i, d))_{i=1}^{|V|}$$

Bag of Words (Count Vectorizer)

```
from sklearn.feature_extraction.text import CountVectorizer

s = [
    'my name is Joe',
    'your name are Joe',
    'my father is Joe'
]
vectorizer = CountVectorizer()
vectorizer.fit_transform(s).toarray()
```

```
array([[0, 0, 1, 1, 1, 1, 0],
       [1, 0, 0, 1, 0, 1, 1],
       [0, 1, 1, 1, 1, 0, 0]])
```

Проблемы

- Нет учёта контекста и порядка слов
- Учет слов, которые не несут дискриминативной информации
- Огромное признаковое пространство

Как учесть важности каждого признака?

(почти) Inverse document frequency:

$$\mathrm{idf}(w) = \frac{1 + |D|}{1 + \mathrm{df}(w)},$$

где $\mathrm{df}(w) = \sum_{d \in D} [w \in d]$.

Отражает, насколько редко данное слово встречается в корпусе.

IDF

Inverse document frequency:

$$\mathrm{idf}(w) = 1 + \log \frac{1 + |D|}{1 + \mathrm{df}(w)},$$

TF-IDF

Всё вместе:

$$\mathrm{df}(w) = \sum_{d \in D} [w \in d]$$

$$\mathrm{idf}(w) = 1 + \log \frac{1 + |D|}{1 + \mathrm{df}(w)}$$

$$v(d) = (\mathrm{tf}(w_i, d) \cdot \mathrm{idf}(w_i))_{i=1}^{|V|}$$

TF-IDF

```
from sklearn.feature_extraction.text import TfidfVectorizer
s = [
    'my name is Joe',
    'your name are Joe',
    'my father is Joe'
]
vectorizer = TfidfVectorizer(norm=None)
vectorizer.fit_transform(s).toarray()
```

```
array([[0. , 0. , 1.3, 1. , 1.3, 1.3, 0. ],
       [1.7, 0. , 0. , 1. , 0. , 1.3, 1.7],
       [0. , 1.7, 1.3, 1. , 1.3, 0. , 0. ]])
```

Промежуточный итог

- что такое текст
- токенизация
- bag of words
- tf-idf

Проблемы

- Нет учёта контекста и порядка слов
- ~~Учет слов, которые не несут дискриминативной информации~~
- Огромное признаковое пространство

Удаление стоп-слов

Слова, которые почти гарантированно встречаются в каждом тексте и которые не отличают один текст от другого.

- a the is are
- то не и а но

Удаление стоп-слов

```
from nltk.corpus import stopwords
en_stop_words = set(stopwords.words('english'))
ru_stop_words = set(stopwords.words('russian'))
```

Удаление слишком/частых редких слов

Которые встречаются в $\leqslant 5\%$ или в $\geqslant 90\%$ текстах, к примеру.

Стемминг

Удаление меняющихся окончаний слов:

- бегу бегут бегущий -> бег
- опрошенных считают налоги необходимыми -> опрошен счита налог необходим
- полк полка -> полк
- write wrote written

Стемминг

```
from nltk.stem.snowball import SnowballStemmer

stemmer = SnowballStemmer(language='english')
sentence = 'George admitted the talks happened'.split()
" ".join([stemmer.stem(word) for word in sentence])
```

Лемматизация

Приведение всех слов к начальной форме:

- George admitted the talks happened -> George admit the talk happen
- write wrote written -> write

Лемматизация

```
from nltk import WordNetLemmatizer

def simple_lemmatizer(sentence):
    lemmatizer = WordNetLemmatizer()
    tokenized_sent = sentence.split()
    pos_taged = [
        (word, get_wordnet_pos(tag))
        for word, tag in nltk.pos_tag(tokenized_sent)
    ]
    return " ".join([
        lemmatizer.lemmatize(word, tag)
        for word, tag in pos_taged
    ])
```

Лемматизация

```
import pymorphy2
def simple_lemmatizer(sentence):
    lemmatizer = pymorphy2.MorphAnalyzer()
    tokenized_sent = sentence.split()
    return " ".join([
        lemmatizer.parse(word)[0].normal_form
        for word in tokenized_sent
    ])
```

Стемминг vs лемматизация

- стемминг обрезает слова ($\Rightarrow$ быстро)
- лемматизация ищет в базе знаний к данному слову начальную форму ($\Rightarrow$ медленно)

Проблемы

- Нет учёта контекста и порядка слов
- ~~Учет слов, которые не несут дискриминативной информации~~
- ~~Огромное признаковое пространство~~

Майнинг словосочетаний

> Метод опорных векторов — метод машинного обучения.

- Коллокации — устойчивые словосочетания
  - `метод опорных векторов, метод машиного обучения, опорных векторов, машинного обучения`
- n-граммы — последовательности из n слов
  - 2-граммы: `метод опорных, опорных векторов, векторов метод, метод машинного, машинного обучения`
- s-скип-n-граммы — последовательности из n слов с s пропусками
  - 1-скип-2-граммы: `метод векторов, опорных метод, векторов машинного, метод обучения`

Регулярные выражения

- теория: https://www.regular-expressions.info/quickstart.html
- Документация библиотеки `re`
- Сервис онлайн проверки регулярных выражений
- Упражнения на регулярные выражения
- "Регулярный" кроссворд
- Ещё хорошая справка

Итог: обработка естественного языка

- токенизация
- bag of words, tf-idf
- стоп-слова
- стемминг, лемматизация
- регулярные выражения