ВМК МГУ

Практическое задание 1. Неточный метод Ньютона для ℓ_2-регуляризованной логистической регрессии.

Курс: Методы оптимизации в машинном обучении, осень 2015

Начало выполнения задания: 19 октября 2015 г.
Срок сдачи: 4 ноября (среда), 23:59.
Среда для выполнения задания: Python.

1 Модель логистической регрессии

Рассматривается задача классификации на два класса. Имеется обучающая выборка $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, где $\mathbf{x}_i \in \mathbb{R}^D$ — вектор признаков i-го объекта, а $y_i \in \{-1, +1\}$ — его метка класса. Задача заключается в предсказании метки класса y_{new} для нового объекта, представленного своим вектором признаков $\mathbf{x}_{\text{new}}$.

В модели логистической регрессии предсказание метки класса выполняется по знаку линейной функции:

$$y(\mathbf{x}) := \text{sign}(\mathbf{w}^\top \mathbf{x}),$$

где $\mathbf{w} \in \mathbb{R}^D$ — параметры модели, настраиваемые в процессе обучения.

Обучение модели осуществляется с помощью минимизации следующей ℓ_2-регуляризованной функции потерь:

$$F(\mathbf{w}) := \sum_{i=1}^N \ln(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i)) + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 . \tag{1}$$

Здесь $\lambda > 0$ — задаваемый пользователем коэффициент регуляризации. Использование регуляризации позволяет снизить вероятность переобучения алгоритма.

2 Неточный метод Ньютона

Рассматривается задача безусловной оптимизации

$$\min_{\mathbf{x} \in \mathbb{R}^n} f(\mathbf{x}), \tag{2}$$

где функция f является дважды непрерывно дифференцируемой.

Метод Ньютона строит последовательность точек $\mathbf{x}_k$, сходящуюся к решению задачи (2). Каждая итерация метода Ньютона имеет вид

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k.$$

Направление оптимизации $\mathbf{d}_k$ находится из следующей системы линейных уравнений:

$$\mathbf{H}_k \mathbf{d}_k = -\mathbf{g}_k, \tag{3}$$

где $\mathbf{g}_k$ и $\mathbf{H}_k$ суть градиент и гессиан функции f в точке $\mathbf{x}_k$. Длина шага α_k выбирается с помощью поиска по прямой.

В неточном методе Ньютона система (3) решается с помощью метода сопряженных градиентов. В этом случае матрицу $\mathbf{H}_k$ можно в явном виде в памяти не формировать, т. к. сама $\mathbf{H}_k$ методу не нужна; нужна только процедура умножения $\mathbf{H}_k$ на произвольный вектор $\mathbf{d}_k$. Кроме того, не имеет большого смысла решать систему (3) сильно точно, если текущая точка $\mathbf{x}_k$ находится далеко от оптимума. Поэтому обычно метод сопряженных градиентов останавливают, как только невязка $\mathbf{r}_k := \mathbf{H}_k \mathbf{d}_k + \mathbf{g}_k$ удовлетворяет условию

$$\|\mathbf{r}_k\| \le \eta_k \|\mathbf{g}_k\| .$$

Последовательность $\{\eta_k\}$, $\eta_k \in (0, 1)$, называется *форсирующей последовательностью* и обычно выбирается следующим образом:

$$\eta_k := \min \left\{ 0.5, \sqrt{\|\mathbf{g}_k\|} \right\} .$$

3 Формулировка задания

1. Реализовать оракул для функции (1). Реализация должна корректно обрабатывать следующие типы исходных данных:
   (a) плотные данные, заданные в обычном формате numpy-массива;
   (b) разреженные данные, заданные в формате разреженной scipy-CSR-матрицы.
2. Реализовать обычный метод Ньютона. Для выбора длины шага использовать сильные условия Вольфа.
3. Проверить работу обычного метода Ньютона на небольшой задаче логистичестической регрессии (1). Построить график сходимости от времени работы. Убедиться, что метод корректно работает как с плотными, так и с разреженными данными.
4. Реализовать метод сопряженных градиентов для квадратичной функции. Проверить правильность работы метода на небольшой системе линейных уравнений.
5. Экспериментально проверить утверждение о том, что число итераций метода сопряженных градиентов, необходимое для решения системы, примерно равно числу кластеров собственных значений матрицы системы. Построить графики сходимости метода от числа итераций.
6. Реализовать функцию умножения гессиана логистической регрессии на произвольный вектор $\mathbf{d}$. Реализованная функция должна корректно обрабатывать случай разреженных данных и **не формировать внутри себя саму матрицу гессиана**. Проверить правильность реализации с помощью разностной аппроксимации через градиент:
$$\nabla^2 \mathbf{F}(\mathbf{w})\mathbf{d} \approx \frac{\nabla \mathbf{F}(\mathbf{w} + \epsilon \mathbf{d}) - \nabla \mathbf{F}(\mathbf{w})}{\epsilon}.$$
7. Реализовать неточный метод Ньютона. Сравнить с точным методом на задаче из п. 3. Построить графики сходимости от времени работы.
8. Применить реализованный неточный метод Ньютона для обучения ℓ_2-регуляризованной логистической регрессии на реальных данных с числом признаком $D \geq 5000$. Сравнить со следующими методами из модуля `scipy.optimize`:
   (a) нелинейный метод сопряженных градиентов (функция `minimize` с параметром `method='CG'`);
   (b) метод L-BFGS (функция `minimize` с параметром `method='L-BFGS-B'`).

   Построить графики сходимости от времени работы. Сравнение необходимо провести **как минимум на трех различных наборах данных**.
9. Написать отчет в формате PDF с описанием всех проведенных исследований.

4 Рекомендации по выполнению задания

1. Чтобы каждый раз при вызове оракула функции (1) не вычислять заново произведения $y_i \mathbf{x}_i$, рекомендуется заранее предподсчитать матрицу $\mathbf{Z} := (z_{ij})_{i,j=1}^{N,D}$, элементами которой являются числа
$$z_{ij} := -y_i x_{ij}, \qquad i = 1, \ldots, N, \quad j = 1, \ldots, D. \tag{4}$$
   После этого исходную матрицу $\mathbf{X}$ из памяти можно удалить.

   Если матрица $\mathbf{X}$ является разреженной, то для быстрого вычисления $\mathbf{Z}$ можно умножить $\mathbf{X}$ слева на матрицу $\mathbf{Y} := \operatorname{diag}(\mathbf{y})$. Для формирования разреженной диагональной матрицы удобно использовать функцию `dia_matrix` из модуля `scipy.sparse`.
2. При поиске по прямой для нахождения точки α_k, удовлетворяющей сильным условиям Вольфа, можно использовать функцию `line_search` из модуля `scipy.optimize`.
3. Графики сходимости методов рекомендуется строить в логарифмической шкале по оси y.

4. Случайную симметричную матрицу $\mathbf{A}$, имеющую заданный спектр $\mathbf{s}$, можно сгенерировать через спектральное разложение:

   (a) Сгенерировать случайную ортогональную матрицу $\mathbf{Q}$:

   ```
   from numpy import random as npr
   from scipy.linalg import orth
   Q = npr.randn(n, n)
   Q = orth(Q)
   ```

   (b) Взять $\mathbf{A} := \mathbf{Q}\mathbf{S}\mathbf{Q}^\top$, где $\mathbf{S} := \mathrm{diag}(\mathbf{s})$.

5. Реальные данные можно скачать с сайта LIBSVM [1]. Для выполнения задания подойдут любые из следующих наборов: news20.binary, rcv1.binary, gisette, real-sim, kdd2010, url, webspam, splice-site.

   Загрузить данные можно с помощью функции `load_svmlight_file` из модуля `sklearn.datasets`. Функция всегда выдает матрицу $\mathbf{X}$ в формате разреженной scipy-CSR-матрицы. Если данные на самом деле разреженными не являются, рекомендуется перевести $\mathbf{X}$ в формат numpy-массива. Это можно сделать с помощью метода `toarray`.

6. Замерить время работы методов из модуля `scipy.optimize` по итерациям можно с помощью передачи параметра `callback` в функцию `minimize`. При этом удобно использовать следующую обертку для функции `minimize`:

```
from scipy.optimize import minimize
from time import time
from numpy.linalg import norm

def write_hist(hist, elaps, f, norm_g):
    hist['elaps'].append(elaps)
    hist['f'].append(f)
    hist['norm_g'].append(norm_g)

def minimize_wrapper(func, x0, mydisp=False, **kwargs):
    hist = {'elaps': [], 'f': [], 'norm_g': []}
    if mydisp: print('%9s %15s %15s' % ('elaps', 'f', 'norm_g'))

    aux = {'tstart': time(), 'elaps': 0}
    def callback(x):
        aux['elaps'] += time() - aux['tstart']
        f, g = func(x)
        norm_g = norm(g, np.inf)
        write_hist(hist, aux['elaps'], f, norm_g)
        if mydisp: print('%9d %15.6e %15.6e' % (aux['elaps'], f, norm_g))
        aux['tstart'] = time()

    callback(x0) # scipy optimizers don't use the 'callback' for the initial point
    out = minimize(func, x0, jac=True, callback=callback, **kwargs)

    return out, hist
```

   Пример запуска метода L-BFGS с помощью функции-обертки:

```
hist = minimize_wrapper(func, x0, mydisp=True, method='L-BFGS-B', options={'ftol': 0})[1]
```

5 Оформление задания

Выполненное задание следует отправить письмом по адресу bayesml@gmail.com с заголовком письма

«[MOMO15] Задание 1, Фамилия Имя».

Убедительная просьба присылать выполненное задание только один раз с окончательным вариантом. Также убедительная просьба строго придерживаться заданных ниже прототипов реализуемых функций.

1. Оракул функции логистической регрессии (1):

Модуль:	`lossfuncs`
Функция:	`logreg(w, Z, regcoef, hess=False)`

Параметры:	`w: numpy.ndarray`
	Точка вычисления, D-мерный вектор.
	`Z: numpy.ndarray` или `scipy.sparse.csr_matrix`
	Матрица $\mathbf{Z}$, заданная (4), $(N \times D)$-матрица.
	`regcoef: float`
	Коэффициент регуляризации $\lambda > 0$.
	`hess: bool`, опционально
	Вычислять дополнительно гессиан или нет.

Возврат:	`f: float`
	Значение функции (1) в точке `w`.
	`g: numpy.ndarray`
	Градиент функции (1) в точке `w`, D-мерный вектор.
	`H: numpy.ndarray`, только при `hess=True`
	Гессиан функции (1) в точке `w`, $(D \times D)$-матрица.

2. Умножение гессиана функции логистической регрессии (1) на произвольный вектор:

Модуль:	`lossfuncs`
Функция:	`logreg_hessvec(w, d, Z, regcoef)`

Параметры:	`w: numpy.ndarray`
	Точка вычисления, D-мерный вектор.
	`d: numpy.ndarray`
	Произвольный D-мерный вектор.
	`Z: numpy.ndarray` или `scipy.sparse.csr_matrix`
	Матрица $\mathbf{Z}$, заданная (4), $(N \times D)$-матрица.
	`regcoef: float`
	Коэффициент регуляризации $\lambda > 0$.

Возврат:	`Hd: numpy.ndarray` Произведение гессиана функции (1) в точке `w` на вектор `d`.

3. Обычный метод Ньютона:

Модуль:	`optim`
Функция:	`newton(func, x0, disp=False, maxiter=500, tol=1e-5, c1=1e-4, c2=0.9)`

Параметры:	`func: callable func(x)`
	Оракул минимизируемой функции.
	Принимает:
	`x: numpy.ndarray`
	Точка вычисления, n-мерный вектор.
	Возвращает:
	`f: float`
	Значение функции в точке `x`.
	`g: numpy.ndarray`
	Градиент функции в точке `x`, n-мерный вектор.
	`H: numpy.ndarray`
	Гессиан функции в точке `x`, $(n \times n)$-матрица.
	`x0: numpy.ndarray`
	Начальная точка, n-мерный вектор.
	`disp: bool`, опционально
	Отображать прогресс метода по итерациям или нет.
	`maxiter: int`, опционально
	Максимальное число итераций метода.
	`tol: float`, опционально
	Точность оптимизации по ℓ_∞-норме градиента.
	`c1: float`, опционально

	Константа c_1 в первом условии Вольфа. `c2: float`, опционально Константа c_2 во втором условии Вольфа.

Возврат:	`x: numpy.ndarray`
	Найденная оценка минимума, n-мерный вектор.
	`hist: dict`
	История процесса оптимизации по итерациям. Словарь со следующими полями:
	`elaps: list of floats`
	Время, пройденное с начала оптимизации.
	`f: list of floats`
	Значение функции.
	`norm_g: list of floats`
	ℓ_∞-норма градиента.

4. Метод сопряженных градиентов:

Модуль:	`optim`
Функция:	`cg(matvec, b, x0, disp=False, tol=1e-5, maxiter=None)`

Параметры:	`matvec: callable matvec(d)`
	Функция умножения матрицы системы на произвольный вектор `d`.
	Принимает:
	`d: numpy.ndarray`
	Произвольный n-мерный вектор.
	Возвращает:
	`Ad: numpy.ndarray`
	Произведение матрицы системы на вектор `d`, n-мерный вектор.
	`b: numpy.ndarray`
	Правая часть системы, n-мерный вектор.
	`x0: numpy.ndarray`
	Начальная точка, n-мерный вектор.
	`disp: bool`, опционально
	Отображать прогресс метода по итерациям или нет.
	`tol: float`, опционально
	Точность оптимизации по ℓ_∞-норме невязки.
	`maxiter: int` или `None`, опционально
	Максимальное число итераций метода. Если `None`, то выбрать равным n.

Возврат:	`x: numpy.ndarray`
	Найденная оценка решения системы, n-мерный вектор.
	`hist: dict`
	История процесса оптимизации по итерациям. Словарь со следующими полями:
	`norm_r: list of floats`
	ℓ_∞-норма невязки.

5. Неточный метод Ньютона:

Модуль:	`optim`
Функция:	`hfn(func, x0, hessvec=None, disp=False, maxiter=500, tol=1e-5, c1=1e-4, c2=0.9)`

Параметры:	`func: callable func(x)`
	Оракул минимизируемой функции.
	Принимает:
	`x: numpy.ndarray`
	Точка вычисления, n-мерный вектор.
	Возвращает:
	`f: float`
	Значение функции в точке `x`.
	`g: numpy.ndarray`

	Градиент функции в точке `x`, n-мерный вектор.
	`x0: numpy.ndarray`
	Начальная точка, n-мерный вектор.
	`hessvec: callable hessvec(x, d)` или `None`, опционально
	Функция умножения гессиана в точке `x` на произвольный вектор `d`.
	Принимает:
	`x: numpy.ndarray`
	Точка вычисления, n-мерный вектор.
	`d: numpy.ndarray`
	Произвольный n-мерный вектор.
	Возвращает:
	`Hd: numpy.ndarray`
	Произведение гессиана в точке `x` на вектор `d`, n-мерный вектор.
	Если `None`, то использовать разностную аппроксимацию через градиент.
	`disp: bool`, опционально
	Отображать прогресс метода по итерациям или нет.
	`maxiter: int`, опционально
	Максимальное число итераций метода.
	`tol: float`, опционально
	Точность оптимизации по ℓ_∞-норме градиента.
	`c1: float`, опционально
	Константа c_1 в первом условии Вольфа.
	`c2: float`, опционально
	Константа c_2 во втором условии Вольфа.

Возврат:	`x: numpy.ndarray`
	Найденная оценка минимума, n-мерный вектор.
	`hist: dict`
	История процесса оптимизации по итерациям. Словарь со следующими полями:
	`elaps: list of floats`
	Время, пройденное с начала оптимизации.
	`f: list of floats`
	Значение функции.
	`norm_g: list of floats`
	ℓ_∞-норма градиента.

Список литературы

[1] LIBSVM Data: Classification, Regression, and Multi-label. — http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.