Домашнее задание №6. Метод Ньютона. Квазиньютоновские методы (практическая часть)

Это практическое домашнее задание. Для его выполнения нужно будет реализовать нужные методы, а затем поставить эксперименты и сделать выводы. Отчет об экспериментах принимается в формате `Jupyter notebook`. Также нужно прислать код реализованных вами методов (файл `methods.py`). За это задание можно получить до 10 баллов + 2 бонусных.

Требования к отчету:

- На графиках должны быть подписаны оси, сами графики – проименованы. Если на одном графике несколько линий, используйте легенду, чтобы их различать. Проверяющему, который смотрит ваш отчет, должно быть понятно, что изображено на графике, даже если он не видел кода.
- Пояснения к экспериментам и выводы оформляйте с помощью `markdown`, а не в виде комментариев к коду.

В этом задании код построен так же, как и в ДЗ №3:

- `oracles.py` – оракул для логистической регрессии с регуляризатором. В этом файле ничего менять не нужно.
- `methods.py` – методы оптимизации. Инструмент для линейного поиска `LineSearchTool` уже реализован. В этом модуле нужно будет реализовать методы Ньютона, BFGS и L-BFGS.
- Правило останова и методы линейного поиска такие же, как и в ДЗ №3.

Важно: перед отправкой задания убедитесь, что ваш код проходит все тесты. Для этого запустите файл с тестами: `python3 tests.py`. **В случае, если тесты не пройдены успешно, домашнее задание не проверяется**.

1 Методы

1.1 Общая схема

В этом задании, как и в ДЗ №3, будем придерживаться следующей схемы.

Алгоритм 1 Общая схема итеративного метода оптимизации

Вход: Начальное приближение x_0, максимальное число итераций N.

1: **for** $k = 0, \ldots, N$ **do**
2: *Вызов оракула*: вычислить $f(x), \nabla f(x), \nabla^2 f(x)$ и т.д.
3: *Критерий остановки*: если выполнен критерий остановки, то выйти.
4: *Вычисление направления*: вычислить направление поиска d_k.
5: *Линейный поиск*: найти подходящую длину шага α_k.
6: *Шаг*: $x_{k+1} = x_k + \alpha_k d_k$.
7: **end for**

Выход: Точка x_k.

- *Критерий остановки*: как и в ДЗ №3, используйте критерий

$$\frac{\|\nabla f(x_k)\|^2}{\|\nabla f(x_0)\|^2} \leq \varepsilon.$$

- *Линейный поиск*: постоянный шаг, правило Армихо или правило Вульфа. В этом задании класс `LineSearchTool` уже реализован в модуле `methods.py`, но вы можете воспользоваться своим кодом из ДЗ №3.

1.2 Метод Ньютона

Рассмотрим задачу безусловной выпуклой оптимизации

$$f(x) \to \min_{x \in \mathbb{R}^n}, \tag{1}$$

где $f(x)$ – дважды непрерывно дифференцируемая функция, причем выполнено условие

$$\forall\ x, s \in \mathbb{R}^n :\ m \|s\|^2 \leq \left\langle s, \nabla^2 f(x) s \right\rangle \leq M \|s\|^2$$

Метод Ньютона является методом *второго порядка*, т.к. он использует вторую производную: гессиан $\nabla^2 f(x)$. Его шаг записывается в виде

$$\boxed{x_{k+1} = x_k - \alpha_k [\nabla^2 f(x_k)]^{-1} \nabla f(x_k)}$$

где размер шага α_k можно задать постоянным, а можно использовать линейный поиск (условия Армихо или Вульфа, например – см. ДЗ №3). **Важно**: доказано, что шаг $\alpha_k = 1$ гарантирует локальную сходимость (сверхлинейную или квадратичную, если гессиан липшицев), поэтому *начинать линейный поиск нужно с шага размера* 1.

Для вычисления $d_k = -[\nabla^2 f(x_k)]^{-1} \nabla f(x_k)$ не обязательно пользоваться явным обращением матрицы. Достаточно решить систему

$$\nabla^2 f(x_k) d = \nabla f(x_k)$$

относительно d. Т.к. $\nabla^2 f(x_k)$ – положительно определенная симметричная матрица, это дает преимущество при решении системы, а именно, возможность применить **разложение Холецкого**.

Для вычисления d_k используйте `scipy.linalg.cho_factor` (находит разложение) и `scipy.linalg.cho_solve` (решает систему, зная разложение Холецкого). Это будет более вычислительно устойчиво, чем непосредственно обратить матрицу $\nabla^2 f(x_k)$ с помощью `np.linalg.inv`, например.

1.3 BFGS

Подсчет и обращение $\nabla^2 f(x_k)$ может оказаться трудоемким при большой размерности задачи. Вместо этого, можно использовать более простую в вычислении матрицу $H_k \approx [\nabla^2 f(x_k)]^{-1}$ и обновлять её на каждой итерации. Существует много различных схем пересчета H_k (см. лекцию 11, например), но мы остановимся на методе **BFGS** (*Бройдена-Флетчера-Голдфарба-Шанно*), который считается наиболее устойчивым и показывает хорошие результаты на практике.

Схема обновления матрицы H_k в методе BFGS такова:

$$s_k = x_{k+1} - x_k,\ y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$$
$$H_{k+1} = \left(I_n - \frac{s_k y_k^\top}{\langle y_k, s_k\rangle}\right) H_k \left(I_n - \frac{y_k s_k^\top}{\langle y_k, s_k\rangle}\right) + \frac{s_k s_k^\top}{\langle y_k, s_k\rangle}$$

В качестве начального приближения выберем $H_0 = I_n$.

Направление поиска задается как $d_k = -H_k \nabla f(x_k)$, и получается алгоритм вида

$$\boxed{x_{k+1} = x_k - \alpha_k H_k \nabla f(x_k)}$$

где α_k выбирается постоянным или с помощью линейного поиска.

1.4 L-BFGS

Дополнительная модификация BFGS – ограниченная память, или limited memory. Будем хранить в памяти последние ℓ значений (s_k, y_k): $\mathcal{H}_k = \{(s_{k-i}, y_{k-i})\}_{i=1}^{\ell}$. В качестве матрицы H_k возьмём матрицу, полученную в результате ℓ-кратного формулы обновления BFGS. Начальную матрицу примем $H_{k-\ell} = \gamma_0^{(k)} I_n$, где $\gamma_0^{(k)} = \frac{\langle y_{k-1}, s_{k-1}\rangle}{\langle y_{k-1}, y_{k-1}\rangle}$ (такой выбор $\gamma_0^{(k)}$ соответствует правилу *Барзилая-Борвейна*). Это приведет к рекурсивному алгоритму умножения матрицы H_k на вектор.

Алгоритм 2 LBFGS-mul

Вход: $v \in \mathbb{R}^n$, $\gamma_0 \in \mathbb{R}$, последовательность $\mathcal{H}$, состоящая из пар (s, y).

if $\mathcal{H} = \emptyset$ **then**
  return $\gamma_0 v$
end if
(s, y) = последняя пара из $\mathcal{H}$
$\mathcal{H}' = \mathcal{H}$ без последней пары
$v' = v - \frac{\langle s, v\rangle}{\langle y, s\rangle} y$
$z = \text{LBFGS-mul}(v', \mathcal{H}', \gamma_0)$
return $z - \frac{\langle y, z\rangle}{\langle y, s\rangle} s + \frac{\langle v, s\rangle}{\langle y, s\rangle} s$

Заметим, что для алгоритма LBFGS-mul в памяти нужно хранить только ℓ пар (s, y), т.е. использование памяти составляет $O(nl)$, что лучше, чем $O(n^2)$ для метода Ньютона и BFGS. Пользуясь этой процедурой, построим метод L-BFGS (*limited memory BFGS*).

Алгоритм 3 L-BFGS

Вход: Начальная точка $x_0 \in \mathbb{R}^n$, размер истории ℓ.

1: **for** $k = 1, \ldots, N$ **do**
2: Задать историю: $\ell_0 = \min\{\ell, k\},\ \mathcal{H}_k = \{(s_{k-i}, y_{k-i})\}_{i=1}^{\ell_0}$.
3: Вычислить $\gamma_0 = \frac{\langle y_{k-1}, s_{k-1}\rangle}{\langle y_{k-1}, y_{k-1}\rangle}$
4: Найти направление поиска $d_k = \text{LBFGS-mul}(-\nabla f(x_k), \mathcal{H}_k, \gamma_0)$.
5: Сделать шаг $x_{k+1} = x_k + \alpha_k d_k$.
6: **end for**

Опять же, размер шага α_k может быть постоянным или выбираться с помощью одномерного поиска.

2 Целевая функция

В этом задании нужно ставить эксперименты на задаче *логистической регрессии с l2-регуляризатором*. Пусть $A \in \mathbb{R}^{m \times n}$ – матрица признаков, $y \in \{-1, 1\}^n$ – вектор меток. Наша функция потерь определяется как

$$f(x) = \frac{1}{m} \sum_{i=1}^{m} \log\left(1 + \exp(-y_i \cdot (Ax)_i)\right) + \lambda \|x\|^2.$$

- Возьмите коэффициент регуляризации равным $\lambda = \frac{1}{m}$.
- Оракул для логистической регрессии с методами `.func`, `.grad`, `.hess` реализован в модуле `oracles.py`. Он создается с помощью функции `create_log_reg_oracle`. **Важно**: оракул оптимизирован под использование разреженных матриц. Если матрица A имеет большую разреженность, передавайте её в `create_log_reg_oracle` в соответствующем формате – `scipy.sparse.csr_matrix`, например. Если передать разреженную матрицу в формате `np.ndarray`, длительность вычисления градиента и гессиана сильно увеличится.

3 Задание

1. Скачайте код из репозитория `https://github.com/alexrogozin12/made_opt/tree/master/homework_6`. В нём содержатся заготовки методов, которые вам предстоит реализовать.
2. Скачайте датасеты `a9a`, `gisette` и `real-sim` с сайта LIBSVM. Загрузить датасеты в этом формате можно с помощью `sklearn.datasets.load_svmlight_file`.
3. Если метод Ньютона и BFGS не смогут работать на `real-sim`, укажите это в отчете и объясните, почему так произошло.
4. Перед отправкой задания на забудьте проверить, что ваш код проходит все тесты. Для этого запустите файл с тестами: `python3 tests.py`. **В случае, если тесты не пройдены успешно, домашнее задание не проверяется**.

3.1 Методы (4 балла)

Реализуйте методы в модуле `methods.py`:

1. Ньютона;
2. BFGS;
3. L-BFGS.

Как и в ДЗ №3, методы должны сохранять значения функции, нормы градиента, времени в поле `.hist`. Также нужно сохранять значения x_k в случае, если размерность задачи не больше 2. Это обязательно для прохождения тестов!

3.2 (Бонус) Точные решения и "правильные" графики (2 балла)

Вообще говоря, для построения хороших графиков лучше по оси Y откладывать не значение функции $f(x_k)$, а невязку по функции $f(x_k) - f(x^*)$ в логарифмической шкале. Но для реальной задачи значение x^* заранее неизвестно.

Для каждого из датасетов найдите решение задачи логистической регрессии x^* и оптимальное значение f^*. Сделайте это, запустив L-BFGS (размер истории можно взять порядка 10) с точностью 10^{-16}. Обратите внимание, что используется правило останова

$$\frac{\|\nabla f(x_k)\|^2}{\|\nabla f(x_0)\|^2} \leq \varepsilon,$$

поэтому, если просто указать `tolerance=1e-16`, то после останова будет $\|\nabla f(x_k)\|^2 \leq \|\nabla f(x_0)\|^2 \cdot 10^{-16}$, а значение $\|\nabla f(x_0)\|^2$ может быть большим. Не забудьте сделать на это поправку, так чтобы в конце получилось $\|\nabla f(x_k)\|^2 \leq 10^{-16}$.

После этого во всех пунктах стройте графики, откладывая по оси Y значения $f(x_k) - f^*$ в логарифмической шкале.

3.3 Метод Ньютона: постоянный шаг vs линейный поиск (2 балла)

Запустите метод Ньютона с постоянным шагом 1 и с линейным поиском по Армихо или Вульфу (можно выбрать любой) на задаче логистической регрессии.

Постройте кривые зависимости функции от времени для разных стратегий выбора шага на одном графике. То же самое для зависимости функции от номера итерации. Сравните результаты и сделайте выводы.

3.4 L-BFGS: зависимость от размера истории (2 балла)

Протестируйте работу L-BFGS с разными размерами истории ℓ. Определите, какой размер истории является оптимальным для каждого из 3 датасетов, постройте соответствующие графики. Из графиков должно быть видно, что с ростом ℓ LBFGS сначала работает лучше, потом начинает работать хуже, и переломный момент наступает при оптимальном размере истории. Стратегию выбора длины шага выберите самостоятельно.

3.5 Сравнение методов (2 балла)

На каждом из датасетов запустите имеющиеся методы:

- Градиентный спуск (из ДЗ №3);
- Метод Ньютона;
- BFGS;
- L-BFGS.

Постройте кривые зависимости функции от времени для всех методов на одном графике.

- Какой метод является наилучшим на каждом из датасетов?
- На каком датасете метод Ньютона показывает себя лучше и почему?