ВМК

Задание 3. Градиентные методы обучения линейных моделей

Практикум 317 группы, 2017

Начало выполнения задания: 9 ноября 2017 года.
Мягкий дедлайн: **23 ноября 2017 года, 23:59.**
Жёсткий дедлайн: **7 декабря 2017 года, 23:59.**

Формулировка задания

Данное задание направлено на ознакомление с линейными моделями и градиентными методами обучения. В задании необходимо:

1. Написать на языке Python собственную реализацию линейного классификатора с произвольной функцией потерь, реализации многоклассовых классификаторов (one-vs-all и all-vs-all). Прототипы функций должны строго соответствовать прототипам, описанным в спецификации и проходить все выданные тесты. Задание, не проходящее все выданные тесты, приравнивается к невыполненному. При написании необходимо пользоваться стандартными средствами языка Python, библиотеками numpy, scipy и matplotlib. Библиотекой scikit-learn пользоваться запрещается, если это не обговорено отдельно в пункте задания.
2. Вывести все необходимые формулы, привести выкладки в отчёте.
3. Провести описанные ниже эксперименты с модельными данными и датасетом 20newsgroups.
4. Написать отчёт о проделанной работе (формат PDF). Отчёт должен быть подготовлен в системе LaTeX.

Линейная модель бинарной классификации

Рассмотрим задачу бинарной классификации. Пусть дана обучающая выборка $X = (x_i, y_i)_{i=1}^l$, где $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{Y} = \{1, -1\}$. Линейная модель классификации определяется следующим образом:

$$a(x) = \text{sign}\left(\langle w, x\rangle + w_0\right), \quad \text{где } w \in \mathbb{R}^d \text{ — вектор весов, } w_0 \text{ — сдвиг.}$$

Далее будем считать, что среди признаков есть константа. Тогда классификатор задаётся как:

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

Процесс обучения заключается в настройке вектора w. Величина $M_i(w) = y_i\langle w, x\rangle$ называется отступом (margin) объекта x_i относительно алгоритма $a(x)$. Если $M_i(w) < 0$, алгоритм допускает ошибку на объекте x_i. Чем больше отступ $M_i(w)$, тем более надёжно и правильно алгоритм классифицирует объект x_i. Пусть $\mathcal{L}(M(w))$ — монотонно невозрастающая функция, такая что $\mathbb{I}[M(w) < 0] \leqslant \mathcal{L}(M(w))$. Будем настраивать вектор весов w, оптимизируя функционал $Q(X, w)$:

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

Метод обучения, использующий логистическую функцию потерь $\log_2(1 + \exp(-M))$, называется логистической регрессией. Одно из основных свойств логистической регрессии — возможность корректного оценивания вероятности принадлежности объекта к каждому из классов:

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

Нетрудно заметить, что максимизация логарифма правдоподобия выборки при такой параметризации вероятностей эквивалентна минимизации суммы логистических функций потерь.

Обычно к оптимизируемому функционалу добавляют второе слагаемое — регуляризатор, не зависящий от данных. Второе слагаемое ограничивает вектор параметров модели тем самым уменьшая переобучение. Один из примеров регуляризации — L_2 регуляризатор:

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

Многоклассовая классификация

Пусть теперь множество $\mathbb{Y} = \{1, \dots K\}$. Задачу многоклассовой классификации можно свести к набору бинарных задач.

- Один против всех (one-vs-all)

  Обучается K классификаторов $a_1(x), \dots, a_K(x)$. Алгоритм $a_j(x)$ обучается по выборке X_j:

  $$X_j = (x_i, 2\mathbb{I}[y_i = j] - 1)_{i=1}^{l}$$

  Таким образом, каждому классификатору $a_j(x)$ соответствует набор весов w_j:

  $$a_j(x) = \text{sign}\left(\langle w_j, x\rangle\right)$$

  Итоговый классификатор будет выдавать класс, соответствующий самому уверенному алгоритму:

  $$a(x) = \arg\max_{j\in\{1,\dots,k\}}\langle w_j, x\rangle$$

- Каждый против каждого (all-vs-all)

  Обучается C_k^2 классификаторов $a_{sj}(x), s, j \in \{1, \dots, k\}, s < j$. Алгоритм $a_{sj}(x)$ обучается по выборке X_{sj}:

  $$X_{sj} = \{(x_i, y_i) \in X \mid y_i = s \text{ или } y_i = j\}$$

  Таким образом, каждому классификатору $a_{sj}(x)$ соответствует набор весов w_{sj}:

  $$a_{sj}(x) = \text{sign}\left(\langle w_{sj}, x\rangle\right)$$

  Итоговый классификатор будет выдавать класс, который наберёт больше всего голосов построенных алгоритмов:

  $$a(x) = \arg\max_{k\in\{1,\dots,K\}}\sum_{s=1}^{K}\sum_{j\neq s}\mathbb{I}[a_{sj} = k]$$

Существует прямое обобщение логистической регрессии на случай многих классов — мультиномиальная регрессия. Пусть построено K линейных моделей $a_1(x), \dots, a_K(x)$, $a_j(x) = \text{sign}\left(\langle w_j, x\rangle\right)$. Каждая модель даёт оценку принадлежности объекта к определённому классу. Эти оценки можно перевести в вероятности с помощью функции $\text{softmax}(z_1, \dots, z_K)$, которая переводит произвольный вещественный вектор в дискретное вероятностное распределение:

$$\text{softmax}(z_1, \dots, z_k) = \left(\frac{\exp(z_1)}{\sum_{k=1}^{K}\exp(z_k)}, \dots, \frac{\exp(z_K)}{\sum_{k=1}^{K}\exp(z_k)}\right)$$

Вероятность k-ого класса можно выразить так:

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

Обучение производится с помощью метода максимального правдоподобия:

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

Стохастический градиентный спуск

Для минимизации функционала $Q(X, w)$ можно использовать метод градиентного спуска. В этом методе выбирается начальное приближение для вектора весов w, затем запускается итерационный процесс, на каждом шаге которого вектор w изменяется в направлении антиградиента функционала $Q(X, w)$:

$$w^{(k+1)} = w^{(k)} - \eta_k \nabla_w Q(X, w) = w^{(k)} - \frac{1}{l}\eta_k \sum_{i=1}^{l} \nabla_w \mathcal{L}(M_i(w))$$

Параметр $\eta_k > 0$ — темп обучения (learning rate), который может равняться константе, а может, например, монотонно уменьшаться с течением итераций. В задании вам предлагается использовать формулу:

$$\eta_k = \frac{\alpha}{k^\beta}, \quad \text{где } \alpha, \beta \text{ — заданные константы}$$

Остановка алгоритма может происходить при слишком малом изменении функционала или нормы вектора весов или после заданного числа итераций.

Функционал $Q(X, w)$ представляет собой сумму функций потерь на каждому объекте. Вычислять градиент на каждой итерации при большом числе объектов может быть очень трудоёмко. Можно оценить градиент суммы градиентом одного слагаемого, на каждой итерации слагаемое выбирается случайно:

$$i \sim unif\{l\}$$

$$w^{(k+1)} = w^{(k)} - \eta_k \nabla_w \mathcal{L}(M_i(w))$$

Такой метод называется методом стохастического градиентного спуска. На каждой итерации можно выбирать не один объект, а небольшое подмножество (mini-batch), что ускорит сходимость алгоритма. Преимуществом алгоритма помимо маленькой вычислительной сложности является то, что на каждом шаге необходимо держать в память лишь один объект из обучающей выборки.

Разностная проверка градиента

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 — базисный вектор, $e_i = [0, 0, \ldots, 0, 1, 0, \ldots, 0]$, ε — небольшое положительное число.

Теоретическая часть

1. Записать формулы для задачи логистической регрессии и градиента этой же задачи в матрично-векторном виде.

   В матрично-векторном виде допускается использование матричных операций, умножения на скаляр, стандартного скалярного произведения, поэлементного произведения ($\odot$), а также применения ко всем элементам вектора некоторой скалярной функции.

2. Записать формулы для задачи многоклассовой логистической регрессии и градиента этой же задачи.

3. Покажите, что при количестве классов = 2, задача многоклассовой логистической регрессии сводится к бинарной логистической регрессии.

Список экспериментов

Во всех экспериментах ниже не забывайте добавлять константный признак в данные!

1. Загрузите с сайта LIBSVM датасет `real-sim`. Загрузить датасет можно с помощью функции `load_svmlight_file` из модуля `sklearn.datasets`.

   Сравните работу алгоритмов градиентного спуска и стохастического градиентного спуска для задачи логистической регрессии на этом датасете. Для алгоритма стохастического спуска сравните несколько начальных приближений между собой. Используйте следующие параметры для методов:

`max_iter=10000, l2_coef=1e-5, tolerance=1e-5, step_alpha=1, step_beta=0, batch_size=1`

Приведите следующие графики:

- Зависимость значения функции от реального времени работы метода
- Зависимость значения функции от итерации метода (эпохи в случае стохастического варианта)
- Зависимость точности (accuracy) от реального времени работы метода
- Зависимость точности (accuracy) итерации метода (эпохи в случае стохастического варианта)

Сделайте выводы по проделанному эксперименту.

2. Исследуйте как влияют параметры размера шага `step_alpha` и `step_beta` на качество работы обоих методов (здесь и далее под качеством подрузамеваются значение оптимизируемой функции и точность решения). Обязательно проверьте значения `step_alpha {0.1, 1, 100, 1000}` и значения `step_beta {0, 0.1, 0.01, 0.5, 1, 2}`. Остальные параметры оставьте такими же, как и в предыдущем эксперименте.

3. Исследуйте как влияет случайность выбора объектов (`random_seed`) в алгоритме стохастического градиентного спуска на качество метода при одном и том же начальном приближении.

4. Исследуйте как влияет размер подвыборки `batch_size` на качество метода стохастического градиентного спуска. Рассмотрите размеры `{1, 5, 10, 20, 50, 100}`. Дополнительно исследуйте как изменились зависимости метода от других параметров алгоритма при размерах подвыборки отличных от 1.

5. Сгенерируйте несколько выборок точек с 2 признаками и 3 классами (по 100 объектов каждого класса) на которых будете проводить эксперименты. Для этого можно воспользоваться функцией `make_blobs` из пакета `sklearn.datasets`.

   Рассмотрите три способа решения многоклассовых задач классификации линейными моделями: один против всех, каждый против каждого и многоклассовая логистическая регрессия. Сравните три способа между собой:

   - Укажите какие особенности, преимущества и недостатки с точки зрения построения разделяющих плоскостей, качества разделения классов и вычислительной эффективности характерны для каждого метода.
   - Для каждой из стратегий подумайте над примерами ситуаций, когда стоит выбирать ее для решения задачи многоклассовой классификации.

6. Загрузить обучающую выборку датасета 20newsgroups при помощи метода `sklearn.datasets.fetch_20newsgroups`. Убрать все заголовки, подписи и цитаты, используя аргумент `remove`. Перевести во всех документах все буквы в нижний регистр. Заменить во всех документах символы, не являющиеся буквами и цифрами, на пробелы.

   Замечание 1. Метод `sklearn.datasets.fetch_20newsgroups` может работать некорректно, если датасет уже содержится на диске.

   Замечание 2. Полезные функции: `str.lower`, `str.split`, `str.isalnum`, `re.sub`, `re.split`.

7. Преобразовать датасет в разреженную матрицу `scipy.sparse.csr_matrix`, где значение x в позиции (i, j) означает, что в документе i слово j встретилось x раз. Разрешается воспользоваться конструктором `sklearn.feature_extraction.text.CountVectorizer`.

   Замечание. Не забудьте потом учесть то, что число используемых токенов в тестовой выборке может отличаться от числа токенов в обучающей.

8. Произвести tf-idf преобразование датасета при помощи `sklearn.feature_extraction.text.TfidfTransformer`, обученного по обучающей выборке. Используйте параметры по умолчанию.

   Подберите параметры линейной модели при помощи кросс-валидации или отложенной выборки. Исследуйте как влияет использование преобразования tf-idf на точность модели на кросс-валидации.

   Замечание. Подбор параметров следует начинать с выбора `step_alpha`, который не приводит к переполнению и осцилированию значения функции, при небольшом `batch_size` и небольшом числе итераций. Затем необходимо подобрать `batch_size`, который может позволить оперативная память вашего компьютера. Далее можно подбирать оптимальное количество итераций, требуемую точность и `step_beta`. Коэффициент регуляризации можно настраивать в самом конце. Возможен и другой порядок подбора параметров алгоритма, возможно, даже более эффективный.

   Замечание. Не стесняйтесь делать много итераций алгоритма!

9. Загрузите тестовую выборку (параметр `subset` метода `fetch_20newsgroups`). Примените лучший алгоритм к тестовой выборке. Сравнить точность с полученной по кросс-валидации. Вывести несколько документов из тестовой выборки, на которых были допущены ошибки. Проанализировать их. Построить и проанализировать матрицу ошибок (confusion matrix). Можно воспользоваться функцией `sklearn.metrics.confusion_matrix`.

10. Примените алгоритмы лемматизации и стемминга на датасете 20newsgroups (например, алгоритмы `SnowballStemmer` и `WordNetLemmatizer` из библиотеки `nltk`). Исследуйте как предобработка корпуса повлияла на точность классификации, время работы алгоритма и размерность признакового пространства.

11. Попробуйте ускорить работу алгоритма за счёт сокращения словаря. Удалите самые частотные слова и самые редкие. Исследуйте как зависит качество и время работы алгоритма от:

    - Сколько было выброшено частотных слов
    - Сколько было выброшено редких слов

    Исследуйте, как измениться качество алгоритма и время его работы, если вместо частотных слов выбрасывать стоп-слова (воспользуйтесь функцией `stopwords.words` из модуля `nltk`).

12. Добавьте в признаковое пространство различные n-граммы (измените параметр `ngramm_range` у `TfidfTransformer`). Исследуйте как влияет размер максимальных добавленных n-грамм на качество и скорость работы алгоритма.

Требования к реализации

Замечание. Для всех функций можно задать аргументы по умолчанию, которые будут удобны вам в вашем эксперименте. Ко всем функция можно добавлять необязательные аргументы, а в словарь `history` разрешается сохранять необходимую в ваших экспериментах информацию.

Прототипы всех функций описаны в файлах, прилагающихся к заданию.

Среди предоставленных файлов должны быть следующие модули и функции в них:

1. Модуль `oracles.py` с реализациями функций потерь и их градиентов.

   Обратите внимание на то, что все функции должны быть полностью векторизованы (т.е. в них должны отсутствовать циклы).

   Замечание. В промежуточных вычислениях стоит избегать вычисления значение $\exp(-b_i\langle x_i, w\rangle)$, иначе может произойти переполнение. Вместо этого следует напрямую вычислять необходимые величины с помощью специализированных для этого функций: `np.logaddexp`, `scipy.special.logsumexp` и `scipy.special.expit`. В ситуации, когда вычисления экспоненты обойти не удаётся, можно воспользоваться процедурой «клипинга» (функция `numpy.clip`).

   Замечание. При вычислении нормировки $\frac{\exp(\alpha_i)}{\sum_k \exp(\alpha_k)}$ может произойти деление на очень маленькое число, близкое к нулю. Необходимо воспользоваться следующим трюком:

   $$\frac{\exp(\alpha_i)}{\sum_k \exp(\alpha_k)} = \frac{\exp(\alpha_i - \max \alpha_j)}{\sum_k \exp(\alpha_k - \max \alpha_j)}$$

2. Модуль `utils.py` с реализацией функции численного подсчёта градиента произвольного функционала.

3. Модуль `sgd_classifier` с реализацией метода стохастического градиентного спуска.

4. Модуль `multiclass` с реализацией многоклассовых методов one-vs-all и all-vs-all.

   Замечание. При реализации all-vs-all может пригодиться модуль `itertools`

Бонусная часть

1. **(до 12 баллов)** Реализуйте режим работы алгоритма `SGDClassifier`, при котором всё обучающая выборка не будет храниться в оперативной памяти. Сохраните датасет 20newsgroups в формате `vowpal wabbit` на диск. В формате `vowpal wabbit` в каждой строчке файла записан один документ коллекции, допускаются записи `someword:3`, что означает, что слово `someword` встретилось в соответствующем документе 3 раза. Перемешайте документы в файле, чтобы большое количество документов одного класса не шли подряд друг за другом. Дополнительно сохраните список всех слов, которые встречались в коллекции.

Реализуйте итератор, который будет считывать следующие `batch_size` строк из созданного файла и преобразовывать их в `numpy.array` или `sparse.csr_matrix` соответствующего размера. При считывании последнего документа, итератор начинает считывание документов заново.

Реализуйте специальный режим обучения для алгоритма стохастического градиентного спуска, который будет принимать на вход вместо матрицы объекты-признаки рассмотренный выше итератор. Теперь алгоритм на каждой итерации не будет выбирать произвольные документы, а будет брать следующую пачку документов, пришедшую из итератора. Таким образом, алгоритм станет «менее случаен». Исследуйте, повлияет ли это на качество работы алгоритма. Рассмотрите несколько различных значений параметра `batch_size`. Попробуйте оценить сколько памяти удаётся сэкономить, используя такой подход, и насколько медленнее такой подход по времени.

Заметьте, что критерий остановки оптимизации алгоритма связанный с разностью значений функции на соседних эпохах для такого режима обучения неэффективен, необходимо использовать другие критерии, не зависящие от объектов все выборки, либо производит константное число итераций метода.

2. **(до 3 баллов)** Напишите unit-тесты к предложенным функциям

3. **(до 5 баллов)** Улучшить качество работы линейных алгоритмов на датасете 20newsgroups с помощью средств, не использующихся в задании. Например, можно реализовать ансамбль линейных алгоритмов, использовать продвинутые методы выделения коллокаций, выделять специальные термины или сущности в тексте. Размер бонуса зависит от величины улучшения и от изобретательности подхода.

4. **(до 5 баллов)** Качественное проведение дополнительного (не пересекающегося с основным заданием и с заданием по ММРО) исследования по теме линейных алгоритмов, обучаемых с помощью градиентных методов: формулируется изучаемый вопрос, ставятся эксперименты, позволяющие на него ответить, делаются выводы. Перед исследованием необходимо обсудить тему с преподавателем.