

Задание 5. Линейные модели, часть 2: Классификация

Курс по методам машинного обучения, 2023-2024, Масляков Глеб

1 Характеристики задания

- **Длительность:** 2 недели
- **Кросс-проверка:** 28 баллов + 2 бонус; в течение 1 недели после дедлайна; нельзя сдавать после жесткого дедлайна
- **Юнит-тестирование:** 12 баллов; Можно сдавать после дедлайна со штрафом в 40%; Публичная часть; PEP8
- **Почта:** ml.cmc@mail.ru
- **Темы для писем на почту:** ВМК.ML[Задание 5][peer-review], ВМК.ML[Задание 5][unit-tests]

Кросс-проверка: После окончания срока сдачи, у вас будет еще неделя на проверку решений как минимум **3х других студентов** — это **необходимое** условие для получения оценки за вашу работу. Если вы считаете, что вас оценили неправильно или есть какие-то вопросы, можете писать на почту с соответствующей темой письма

2 Описание задания

В данном задании требуется реализовать несколько классов для предварительной обработки категориальных признаков с целью повышения качества линейной модели. Далее, необходимо воспользоваться свежереализованными классами для предобработки и ответить на вопросы в прилагаемом ноутбуке. Старайтесь отвечать развёрнуто и обосновано. Там где необходимо, напишите код, подтверждающий ваши размышления, и затежайте формулы.

Подробное описание того, что нужно реализовать приведено в следующей главе. Также, вы можете загрузить пример решения из системы и посмотреть на пример использования в публичных тестах.

3 Кросс-проверка

- **Ссылка на задание:** [ссылка тут](#)

Внимание! Отправлять задание нужно в систему во вкладку **Linear models: classification (notebook)**.

Замечание: После отправки ноутбука убедитесь, что все графики сохранены корректно и правильно отображаются в системе.

Замечание: Перед сдачей проверьте, пожалуйста, что не оставили в ноутбуке где-либо свои ФИО, группу и так далее — кросс-рецензирование проводится анонимно.

4 Юнит-тестирование

Реализованные классы с предобработкой необходимо сохранить в шаблонном файле Task.py. После реализации ваш код можно протестировать локально, а затем его необходимо сдать в проверяющую систему (вкладка **Linear models (unit-tests)**). Более подробное **описание заданий** приведено в секции **7**.

Замечание: Запрещается пользоваться библиотеками, импорт которых не объявлен в файле с шаблонами функций.

Замечание: Задания, в которых есть решения, содержащие в каком-либо виде взлом тестов, дополнительные импорты и прочие нечестные приемы, будут автоматически оценены в 0 баллов без права пересдачи задания.

5 Стил программирования

При выполнении задач типа unit-tests, ML-задания вам необходимо будет соблюдать определенный стиль программирования (codestyle). В данном случае мы выбирали PEP8 как один из популярных стилей для языка Python. Зачем мы это вводим? Хорошая читаемость кода – не менее важный параметр, чем работоспособность кода :) Единый стиль позволяет быстрее понимать код сокомандников (в командных проектах, например), упрощает понимание кода (как другим, так и вам). Также, привыкнув к какому-либо стилю программирования, вам будет проще переориентироваться на другой.

Полезные при изучении PEP8 ссылки, если что-то непонятно, дополнительный материал можно найти самостоятельно в интернете:

- [Официальный сайт PEP8, на английском](#)
- [Небольшое руководство по основам на русском](#)

Требования к PEP8 мы вводим только для заданий с авто-тестами, требований к такому же оформлению ноутбуков нет. Но улучшение качества кода в соответствии с PEP8 в них приветствуется!

Внимание!!! В проверяющей системе, при несоответствии прикрепляемого кода PEP8, будет высвечиваться вердикт Preprocessing failed. Более подробно посмотреть на ошибки можно, нажав на них:

12.10.2022 cross_val.py
19:22 scalers.py

Preprocessing failed

Результат

Время
работы в
секундах

Preprocessing failed: Runtime error

```
Traceback (most recent call last):
  File "pre.py", line 39, in <module>
    raise RuntimeError(err_message)
RuntimeError: Found 6 errors or warnings in submission.
Detailed info:
scalers.py:6:65: W291 trailing whitespace
scalers.py:17:73: W291 trailing whitespace
scalers.py:31:13: E128 continuation line under-indented for visual indent
scalers.py:38:56: W291 trailing whitespace
scalers.py:44:43: W291 trailing whitespace
scalers.py:80:33: E131 continuation line unaligned for hanging indent
```

Также посылки, упавшие по code style, считаются за попытку сдачи и идут в счет общего количества посылок за день.

Проверить стиль программирования локально можно при помощи утилиты [pycodestyle](#) (в окружении, которое вы ставили, эта утилита уже есть) с параметром максимальной длины строки (мы используем 160 вместе дефолтных 79):

```
pycodestyle --max-line-length=160 your_file_with_functions.py
```

6 Тестирование

В cv-gml можно скачать все файлы, необходимые для тестирования, одним архивом. Для этого просто скачайте zip-архив во вкладке **шаблон решения** соответствующего задания и разархивируйте его. Далее следуйте инструкциям по запуску тестирования.

Запуск тестов производится одной из четырёх команд

```
python run.py onehot
python run.py weights
python run.py counters
python run.py foldcounters
```

7 Что нужно реализовать

7.1 One-hot преобразование

Класс `MyOneHotEncoder` реализует one-hot кодирование признаков. Пусть i -ый признак принимает значения $\{a_1, a_2, \dots, a_n\}$. Тогда значение a_j должно быть преобразовано в вектор длины n , у которого все компоненты кроме j -ой нулевые, а j -ая компонента равна единице. При этом, неоднозначность с номерами значений признаков разрешается следующим образом: меньший номер достаётся меньшему значению (для строк имеется ввиду лексикографический порядок).

В классе `MyOneHotEncoder` требуется реализовать методы `fit` и `transform`. Метод `fit` принимает на вход `pandas.DataFrame` размера $n_{\text{object}} \times n_{\text{features}}$ — обучающая выборка с категориальными признаками. После вызова метода `fit` объект класса `OneHotEncoder` должен запомнить всю необходимую информацию для one-hot преобразования признаков.

Метод `transform` (который и осуществляет напрямую one-hot кодирование) также принимает на вход `pandas.DataFrame` размера $n_{\text{object}} \times n_{\text{features}}$, где n_{features} совпадает с таковым у метода `fit`. Возвращает метод `transform` `numpy.array` размера $n_{\text{object}} \times (|f_1| + \dots + |f_{n_{\text{features}}}|)$, где $|f_i|$ — количество уникальных значений i -го признака. Для устранения неопределённости будем считать, что при $i < j$, соответствующие i -ому признаку $|f_i|$ бинарных признаков идут раньше, чем соответствующие j -ому признаку $|f_j|$ бинарных признаков. Отдельно обрабатывать ситуацию, когда данные, попавшие в метод `transform`, содержат значения признаков, не встречавшиеся в обучающей выборке, **НЕ ТРЕБУЕТСЯ**.

7.2 Счётчики

Класс `Counters` реализует другой способ кодирования категориальных признаков. Пусть i -ый признак принимает значения $\{a_1, a_2, \dots, a_n\}$. Тогда признак a_j должен быть преобразован в вектор длины 3. Первая компонента данного вектора представляет из себя среднее значение целевой переменной для объектов, у которых i -ый признак принимает значение a_j :

$$\text{successes} = \frac{\sum_{k=1}^{n_{\text{objects}}} y_k \mathbb{I}[x_k^i = a_j]}{\sum_{k=1}^{n_{\text{objects}}} \mathbb{I}[x_k^i = a_j]}$$

Здесь \mathbb{I} означает индикаторную функцию: $\mathbb{I}[\zeta] = 1$, если выражение ζ истинно, и $\mathbb{I}[\zeta] = 0$, если выражение ζ ложно.

Вторая компонента — это доля объектов, у которых i -ый признак принимает значение a_j :

$$\text{counters} = \frac{\sum_{k=1}^{n_{\text{objects}}} \mathbb{I}[x_k^i = a_j]}{n_{\text{objects}}}$$

Третья компонента — сглаженное отношение между этими величинами:

$$\text{relation} = \frac{\text{successes} + a}{\text{counters} + b}, \quad a \geq 0, \quad b \geq 0$$

В классе `SimpleCounterEncoder` необходимо реализовать метод `fit`, аналогичный одноимённому методу классу `OneHotEncoder`. Отличие в том, что помимо обучающих объектов метод `fit` принимает на вход соответствующий вектор со значениями целевой переменной в формате `pandas.Series` (при правильной реализации форматы `numpy.array` и `pandas.DataFrame` также подойдут).

Метод `transform` в классе `counters` получает на вход `pandas.DataFrame` размера $n_{\text{object}} \times n_{\text{features}}$, где n_{features} совпадает с таковым у метода `fit`. Возвращает метод `transform` `numpy.array` размера $n_{\text{object}} \times 3 \cdot n_{\text{features}}$. Также у данного метода будут параметры a и b по умолчанию равные 10^{-5} .

Одним из недостатков кодирования при помощи счётчиков является риск утечки значения целевой переменной для данного объекта. На практике это может приводить к переобучению. Для того, чтобы это предотвратить обучающая выборка разбивается на k подмножеств, и величины `successes`, `counters` и `relation` считаются по оставшимся $k - 1$ подмножествам (как в кросс-валидации).

Данную стратегию реализует класс `FoldCounters`, в котором опять нужно реализовать методы `fit` и `transform`. Также обратите внимание, что в метод `__init__` подаётся на вход параметр `n_folds` — число подмножеств, которое необходимо использовать в методе `__init__`. Метод `fit` устроен также, как и в `Counters`, за исключением параметра `seed`. Для разбиения обучающей выборки на фолды необходимо использовать вспомогательную функцию `group_k_fold`. Данное разбиение необходимо запомнить в методе `fit`! Функция `group_k_fold` генерирует случайное разбиение, поэтому не забываем прокидывать параметр `seed`.

Метод `transform` в классе `FoldCounters` получает на вход `pandas.DataFrame` размера $n_{\text{object}} \times n_{\text{features}}$, где n_{object} и n_{features} совпадают с таковыми у метода `fit` (`fit` и `transform` применяются к одной выборке). Возвращает метод `transform` `numpy.array` размера $n_{\text{object}} \times 3 \cdot n_{\text{features}}$. Также у данного метода будут параметры `a` и `b` по умолчанию равные 10^{-5} .

Отдельно рассматривать ситуацию, когда в одно или несколько разбиений не попали все значения какого-то признака **НЕ ТРЕБУЕТСЯ**.

7.3 Теоретический вопрос

Рассмотрим следующую задачу. Величина x принимает значения из конечного множества X . Целевая переменная y принимает значения $\{0, 1\}$. То есть можно сказать, что у нас бинарная классификация с одним категориальным признаком. К величине x применяется one-hot преобразование, в результате которого возникает величина x' (это уже вектор из нулей и единиц). Прогноз $p(x)$ целевой переменной y равен: $p(x) = \langle w, x' \rangle$, где w — обучаемые веса. Обратите внимание, что здесь нет свободного члена, и нет сигмоиды. Обучение весов w осуществляется минимизацией логистической функции потерь (logloss) по весам w :
$$L = - \sum_{i=1}^{n_{\text{objects}}} y_i \cdot \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i)).$$
 Как будут выглядеть оптимальные веса?

В качестве ответа на вопрос, требуется написать функцию `weights` (не используя специализированных библиотек для линейной классификации). Данная функция принимает на вход параметры x и y : оба `numpy.array` размера n_{objects} — это обучающая выборка. Возвращает функция список с оптимальными значениями весов. Порядок весов в списке определяется аналогично one-hot кодированию.