Тестовое задание для стажёра команды продуктовой аналитики, часть А

Выполнил: Шишков Алексей

ФКН ВШЭ, прикладная математика и информатика, 2 курс

Часть А.

Задача 1.

Найдём MSE для текущей модели. Она считается по формуле

$$\frac{(y_1 - y_1')^2 + ... + (y_n - y_n')^2}{n},$$

где y – верные значения, y' – значения, полученные с помощью модели. Нам дано, что

$$y_1 - y_1' = \ldots = y_{80} - y_{80}' = 0.5,$$

$$y_{81} - y_{81}' = \ldots = y_{100} - y_{100}' = -0.3.$$

MSE в этом случае равно

$$\frac{0.5^2 + \ldots + 0.5^2 + (-0.3)^2 + \ldots + (-0.3)^2}{100} = \frac{0.25 \cdot 80 + 0.09 \cdot 20}{100} = 0.218.$$

Если мы прибавим C ко всем предсказаниям, то мы получим новые предсказания $y_i'' = y_i' + C$. MSE тогда будет равняться

$$\frac{(0.5 - C)^2 + ... + (0.5 - C)^2 + (-0.3 - C)^2 + ... + (-0.3 - C)^2}{100} =$$

$$= \frac{20 - 80C + 80C^2 + 20C^2 + 12C + 1.8}{100} = C^2 - 0.68C + 0.218.$$

Нам хочется минимизировать эту функцию по C. Это парабола ветвями вверх, минимум находится в вершине, то есть в $C = \frac{0.68}{2} = \frac{68}{200} = \frac{17}{50}$, и MSE тогда равно $\frac{64}{625}$.
Также можно посчитать, для каких C ответ получается меньше, чем 0.218, для этого надо решить неравенство $C^2 - 0.68C + 0.218 < 0.218$. Решением является $0 < C < \frac{17}{25}$.

Ответ: да, константа существует, можно взять любое C из $\left(0, \frac{17}{25}\right)$. Для достижения минимального MSE стоит взять $C = \frac{17}{50}$.

Задача 2.

Отрицательные значения может возвращать алгоритм градиентного бустинга. Для того, чтобы понять, почему так происходит, обратимся к способу построения обеих моделей.
Для случайного леса несколько деревьев решения строятся с помощью бэггинга, для предсказания используется усреднение ответов всех деревьев. А так как каждое дерево может возвращать только усреднение нескольких (возможно — одного) ответа обучающей выборки (оно усредняет значения в листах), то можно сказать, что в случайном лесе ответом может быть только сумма ответов обучающей выборки с неотрицательными коэффициентами. Так как все ответы обучающей выборки неотрицательны, то и результат предсказания не может быть отрицательным.
В градиентом бустинге тоже используются решающие деревья. Однако они строятся последовательно, и на каждом шаге для следующего дерева подбирается коэффициент, с которым оно берётся. И этот коэффициент уже может быть каким угодно, так что в итоге ответы могут быть отрицательными.
Ответ: в градиентном бустинге.

Задача 3.

Такое могло произойти по нескольким причинам.
Во-первых, R^2 может не так хорошо показывать то, от чего мы пытались избавиться. В одной части выборки мы могли улучшить отличие дисперсии от реальных значений, в другой — ухудшить, и в среднем результат был бы таким же.
Во-вторых, наша модель может упереться в свой предсказательный максимум. Возможно, мы выбрали неправильную модель, и даже если мы будем пытаться улучшить построение модели, ничего не выйдет.
В-третьих, возможно в исходных данных могли быть много выбросов и плохих объектов, что не позволяет модели хорошо работать.
В-четвёртых, мы могли ошибиться при построении модели. Стоит ещё раз перепроверить код, библиотеки, перечитать документацию и проверить расчёты.

Задача 4.

Стандартное отклонение n величин считается по формуле

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - M)^2}{n}},$$

где x_i — значение i-й величины, M — среднее арифметическое n величин: $\frac{\sum_{i=1}^{n} x_i}{n}$. Раскроем скобки в формуле подсчёта σ:

$$\sqrt{\frac{\sum_{i=1}^{n}(x_i - M)^2}{n-1}} = \sqrt{\frac{\sum_{i=1}^{n}(x_i^2 - 2x_iM + M^2)}{n}} = \sqrt{\frac{\sum_{i=1}^{n}x_i^2 - 2M\sum_{i=1}^{n}x_i + nM^2}{n}} =$$

$$= \sqrt{\frac{\sum_{i=1}^{n}x_i^2 - 2nM + nM^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n}x_i^2 - nM^2}{n}} = \sqrt{\frac{\sum_{i=1}^{n}x_i^2 - \frac{\left(\sum_{i=1}^{n}x_i\right)^2}{n}}{n}}$$

Таким образом, храня и эффективно пересчитывая в одной переменной сумму значений, во второй — сумму квадратов, в третьей — общее число обработанных чисел, можно использовать эту формулу для подсчёта стандартного отклонения. Памяти для этого нужно всего 24 бита на каждый float, что укладывается в половину от ста мегабайт.
Полный код можно увидеть в файле `./A/4-stream-std.py`.

Задача 5.

Число задач в запросах я обозначу как `{X}`.
Из условия следует, что задачи выполняются подряд, и процесс прекращается, если i-я задача достигла максимального числа перезапусков.
Первым запросом выберем задачи, которые выполнились, но не с первого раза. Для этого рассмотрим номера задач в таблице, которые больше 100, и возьмём их по модулю 100.

```
SELECT MOD(task_id, {X}) AS task_base_id FROM tasks WHERE task_id >= {X};
```

Далее возможно есть задача (одна), которая так и не выполнилась. Для того, чтобы вычислить её id, можно взять максимум по id среди всех выполненных задач (то есть максимум модулей) и прибавить 1. Если полученное число будет меньше 100, то надо объединить результат прошлого запроса с этим числом.

```
SELECT task_base_id FROM (
  SELECT MAX (done.task_base_id) + 1 as task_base_id from (
    SELECT MOD(task_id, {X}) AS task_base_id FROM tasks
  ) AS done
) AS first_not_done WHERE task_base_id < {X}
```

Таким образом, объединённый запрос будет выглядеть как

```
SELECT MOD(task_id, {X}) AS task_base_id FROM tasks WHERE task_id >= {X}
UNION
SELECT task_base_id FROM (
  SELECT MAX (done.task_base_id) + 1 as task_base_id from (
    SELECT MOD(task_id, {X}) AS task_base_id FROM tasks
  ) AS done
) AS first_not_done WHERE task_base_id < {X};
```

Код запроса также можно найти в файле `./A/5.sql`.

Часть B.

Главная проблема в коде, который был предоставлен, состоит в том, что функция обновления `Counter`-а не является атомарной, но вызывалась параллельно. Это приводило к тому, что некоторые обновления не учитывались и забывались: если в `Counter` с текущим значением 0 придёт два обновления одновременно, то нули запишется в переменные local, оба нуля как-то увеличатся, а потом функция, которая записывает значение второй по счёту затрёт изменения, которые были сделаны первой функцией (так как `Counter.value` уже будет изменено, а `local` во второй функции не будет знать об этих изменениях).
Эту можно исправить таким образом: убрать сохранение данных в локальную переменную функции, а изменять значение `self.value` с помощью `+=`. Также стоит убрать `time.sleep(0.1)` из функции обновления значения.
Теперь код хотя бы выдаёт правильный результат. Но он всё ещё работает неоптимально: вызов функции `Task.execute` должен отнимать больше времени, чем `Counter.update`, поэтому логичнее сделать вызов его параллельным, а не наоборот. Сделав параллельным выполнение `Task` и ожидание выполнения с помощью `concurrent.futures.as_completed`, мы сэкономим время, а не запуская параллельно обновление значения (`Counter.update`) мы точно не получим проблему потери каких-то значений (и вряд ли потеряем много времени, потому что прибавление одного числа не должно быть дорого).
Для упрощения дебага я добавил отладочную информацию, а также в класс задачи ввёл новое поле — идентификатор задачи, по которому её можно опознавать.
Полный код можно найти в папке `./B`.