Конькова Нина Александровна, https://vk.com/id285308879
27 задание ЕГЭ

Сложная задача

Условие:

У вас есть большой набор данных для задачи бинарной классификации, сохранённый в файле **data_hard.txt**. Файл содержит до миллиона записей. В этом файле каждая строка содержит два числа: первое число **x** — это целочисленное значение признака (например, масса объекта), второе число **y** — это метка класса (0 или 1). Найдите оптимальный порог **t**, который позволяет разделить объекты двух классов, минимизируя количество ошибок классификации, и посчитайте, сколько объектов было ошибочно классифицировано при использовании оптимального порога **t**. Выведите в ответ два числа: найденный порог t (если таких несколько, выведите порог с минимальным индексом) и количество ошибок классификации.

Объект считается ошибочно классифицированным, если:

- $x<t$, а $y=1$, или
- $x\geq t$, а $y=0$.

Пример содержимого файла data_hard.txt:
0.5 0
1.0 1
1.5 0
2.0 1
2.5 1
3.0 0
3.5 0
4.0 1
Пример вывода:
2.5
3

Решение:

1. Сначала мы считываем файл построчно и обрабатываем данные частями (пакетами). Это сделано для того, чтобы не перегружать оперативную память при работе с большими файлами. Каждая строка файла разбивается на два числа: значение признака x и метку класса y, которые добавляются в текущий пакет.
2. Сортируем текущий пакет данных по значению признака x. Для каждой точки в пакете подсчитываем количество ошибок для текущего порога t, обновляем минимальное количество ошибок в пределах пакета и запоминаем лучший порог. Также ведём учёт ошибок за пределами текущего пакета.
3. После обработки всех пакетов результаты объединяются: уточняем глобальные счётчики ошибок для классов y=0 и y=1. Таким образом, оптимальный порог и минимальное количество ошибок пересчитываются с учётом всех пакетов.
4. В конце выводим оптимальный порог t и минимальное количество ошибок в формате, который требуется в условии задачи.

```python
def find_best_threshold_hard_exclusive(file_path, chunk_size=10000):
    # Шаг 1: Инициализация переменных
    min_errors = float('inf')
    best_t = None
    count_0_total, count_1_total = 0, 0

    # Шаг 2: Считаем общее количество классов (0 и 1)
    with open(file_path, 'r') as file:
        for line in file:
            _, y = map(int, line.split())
            if y == 0:
                count_0_total += 1
            else:
                count_1_total += 1

    # Шаг 3: Пакетная обработка
    left_0, left_1 = 0, 0
    with open(file_path, 'r') as file:
        data_chunk = []
        for line in file:
            x, y = map(int, line.split())
            data_chunk.append((x, y))

            # Когда накопили chunk_size строк, обрабатываем пакет
            if len(data_chunk) >= chunk_size:
                data_chunk.sort()  # Сортируем пакет по x
                for i in range(len(data_chunk)):
                    x, y = data_chunk[i]

                    # Условие: пропускаем текущий порог t (не включаем в подсчёт)
                    if i < len(data_chunk) - 1 and data_chunk[i + 1][0] == x:
                        continue

                    # Подсчёт ошибок для текущего порога
                    errors_left = left_1  # Ошибки слева
                    errors_right = count_0_total - left_0  # Ошибки справа
                    total_errors = errors_left + errors_right

                    # Условие выбора минимального порога
                    if total_errors < min_errors or (total_errors == min_errors and (best_t is None or x < best_t)):
                        min_errors = total_errors
                        best_t = x

                    # Обновляем счётчики
                    if y == 0:
                        left_0 += 1
                    else:
                        left_1 += 1

                # Обнуляем текущий пакет
                data_chunk = []

        # Обработка оставшихся данных
        if data_chunk:
            data_chunk.sort()
            for i in range(len(data_chunk)):
                x, y = data_chunk[i]
```

```
                # Пропускаем текущий порог t
                if i < len(data_chunk) - 1 and data_chunk[i + 1][0] == x:
                    continue

                errors_left = left_1
                errors_right = count_0_total - left_0
                total_errors = errors_left + errors_right

                if total_errors < min_errors or (total_errors == min_errors and (best_t is None or x < best_t)):
                    min_errors = total_errors
                    best_t = x

                if y == 0:
                    left_0 += 1
                else:
                    left_1 += 1

    return best_t, min_errors

# Пример вызова
file_path = 'data_easy.txt'
best_t, min_errors = find_best_threshold_hard_exclusive(file_path)
print(best_t, min_errors)
```

Листинг кода:

```
#Сложный уровень
def find_best_threshold_hard_exclusive(file_path, chunk_size=10000):
    # Шаг 1: Инициализация переменных
    min_errors = float('inf')
    best_t = None
    count_0_total, count_1_total = 0, 0

    # Шаг 2: Считаем общее количество классов (0 и 1)
    with open(file_path, 'r') as file:
        for line in file:
            _, y = map(int, line.split())
            if y == 0:
                count_0_total += 1
            else:
                count_1_total += 1

    # Шаг 3: Пакетная обработка
    left_0, left_1 = 0, 0
    with open(file_path, 'r') as file:
        data_chunk = []
        for line in file:
            x, y = map(int, line.split())
            data_chunk.append((x, y))

            # Когда накопили chunk_size строк, обрабатываем пакет
            if len(data_chunk) >= chunk_size:
                data_chunk.sort()  # Сортируем пакет по x
                for i in range(len(data_chunk)):
                    x, y = data_chunk[i]
```

```
                # Условие: пропускаем текущий порог t (не включаем в подсчёт)
                if i < len(data_chunk) - 1 and data_chunk[i + 1][0] == x:
                    continue

                # Подсчёт ошибок для текущего порога
                errors_left = left_1  # Ошибки слева
                errors_right = count_0_total - left_0  # Ошибки справа
                total_errors = errors_left + errors_right

                # Условие выбора минимального порога
                if total_errors < min_errors or (total_errors == min_errors and (best_t is None or x <
best_t)):
                    min_errors = total_errors
                    best_t = x

                # Обновляем счётчики
                if y == 0:
                    left_0 += 1
                else:
                    left_1 += 1

            # Обнуляем текущий пакет
            data_chunk = []

        # Обработка оставшихся данных
        if data_chunk:
            data_chunk.sort()
            for i in range(len(data_chunk)):
                x, y = data_chunk[i]

                # Пропускаем текущий порог t
                if i < len(data_chunk) - 1 and data_chunk[i + 1][0] == x:
                    continue

                errors_left = left_1
                errors_right = count_0_total - left_0
                total_errors = errors_left + errors_right

                if total_errors < min_errors or (total_errors == min_errors and (best_t is None or x <
best_t)):
                    min_errors = total_errors
                    best_t = x

                if y == 0:
                    left_0 += 1
                else:
```

```
                left_1 += 1

    return best_t, min_errors

# Пример вызова
file_path = 'data_hard.txt'
best_t, min_errors = find_best_threshold_hard_exclusive(file_path)

print(best_t, min_errors)
```

Ответ для файла: 3 449837