

# Конспект по теме "Анализ алгоритмов"

## Вычислительная сложность

**Время работы алгоритма** не измеряют в секундах, оно определяется количеством выполняемых алгоритмом элементарных операций: сложений, умножений и сравнений. Когда говорят о времени работы на определённом компьютере, обычно его называют **реальным временем работы**. На время работы алгоритма также влияют его аргументы.

Время работы сложных алгоритмов вычислить нельзя. Поэтому рассчитывают их **вычислительную сложность**, или **асимптотическое время работы** — от слова **асимптота** — прямая, к которой кривая приближается, но не может её пересечь.

Обозначим длину входных данных  $n$ . Время работы — это функция от  $n$ , то есть  $T(n)$ . Асимптотическое время работы алгоритма показывает, как растёт  $T(n)$  при увеличении  $n$ .

Когда  $T(n)$  — это многочлен, то асимптотическое время работы равно одночлену наибольшей степени без коэффициента (например,  $n^2$  вместо  $5n^2$ ). При больших значениях  $n$  остальные одночлены не важны.

- Если  $T(n) = 4n + 3$ , асимптотическое время работы  $T(n) \sim n$ . У алгоритма **линейная сложность**.
- Если  $T(n) = 5n^2 + 3n - 1$ , асимптотическое время работы  $T(n) \sim n^2$ . У алгоритма **квадратичная сложность**.
- Если  $T(n) = 10n^3 - 2n^2 + 5$ , то  $T(n) \sim n^3$ . У алгоритма **кубическая сложность**.
- Если  $T(n) = 10$ , то  $T(n) \sim 1$ . У алгоритма **константная сложность**, то есть не зависит от  $n$ .

## Время обучения линейной регрессии

Задача обучения линейной регрессии такая:

$$w = \arg \min_w \text{MSE}(Xw, y)$$

Веса вычисляются по формуле:

$$w = (X^T X)^{-1} X^T y$$

Найдём вычислительную сложность расчёта весов, но прежде:

- Обозначим количество объектов в обучающей выборке как  $n$ , а количество признаков —  $p$ ;
- Размер матрицы  $X$  будет  $n \times p$ , размер вектора  $y$  —  $n$ ;
- Обозначим вычислительную сложность как  $T(n, p)$ , поскольку она зависит от двух параметров:  $n$  и  $p$ .

Тогда вычислительная сложность обучения линейной регрессии будет  $T(n, p) \sim np^2 + p^3 + np^2 + np$

Признаков обычно меньше, чем объектов, то есть  $p < n$ . Умножим обе части на  $p^2$ , получим  $p^3 < np^2$ . Оставим только одночлен наибольшей степени и получим:  $T(n, p) \sim np^2$ . Если признаков много, модель будет обучаться долго.

## Итеративные методы

Задача решается **прямым методом**, когда модель линейной регрессии обучается по такой формуле:

$$w = (X^T X)^{-1} X^T y$$

Прямые методы помогают найти точное решение по заданной формуле или алгоритму. Их вычислительная сложность не зависит от данных.

Другой подход к обучению модели линейной регрессии — **итеративный метод**, или **итеративный алгоритм**. Он не даёт точного решения, только приближённое. Алгоритм многократно выполняет похожие итерации, на каждом шаге решение становится всё точнее. Если слишком большая точность не нужна, хватит и небольшого числа итераций.

Вычислительная сложность итеративных методов зависит от числа шагов, на которое может влиять и содержание данных.

## Метод бисекции

Итеративным методом найдём решение уравнения  $f(x) = 0$ . Пусть  $f(x)$  — **непрерывная функция**, то есть график такой функции можно нарисовать, не отрывая карандаша от бумаги.

Решить уравнение поможет **метод бисекции**. На вход он принимает непрерывную функцию и отрезок  $[a, b]$ . У значений  $f(a)$  и  $f(b)$  разные знаки.

Когда эти два условия выполняются:

1. функция непрерывна,
  2. у значений на концах отрезка знаки разные,
- то корень уравнения где-то на отрезке.

На каждой итерации метод бисекции:

- Проверяет, равно ли нулю любое значение  $f(a)$  или  $f(b)$ . Если равно, то решение найдено;
- Находит середину отрезка  $c = (a + b) / 2$ ;
- Сравнивает знак  $f(c)$  со знаками  $f(a)$  и  $f(b)$ :
  - Если у  $f(c)$  и  $f(a)$  разные знаки, то корень обязательно есть на отрезке  $[a, c]$ . К этому отрезку и переходит алгоритм на следующей итерации;
  - Если у  $f(c)$  и  $f(b)$  разные знаки, то корень обязательно есть на отрезке  $[b, c]$ . К нему переходит алгоритм на следующей итерации;

- Знаки  $f(a)$  и  $f(b)$  разные, поэтому других вариантов быть не может.

Обычно заранее выбирают точность решения, например,  $\epsilon = 0.000001$ . На каждой итерации отрезок с корнем уменьшается в два раза. Если получаем длину отрезка меньше  $\epsilon$ , работу алгоритма можно остановить. Такое условие называется **условием остановки**.

## Сравнение методов

Большая часть этого курса посвящена ключевому для машинного обучения итеративному методу — **градиентному спуску** (англ. *gradient descent*). На нём построено много алгоритмов обучения.

Обозначим его преимущества перед прямыми методами:

```
Choose randomly  $x_0$ 
While  $\|f'(x_{n-1}) - f'(x_n)\| > \epsilon$  do
  Choose a decreasing  $\gamma_n$  (generally  $1/n$ )
  Compute  $x_{n+1} = x_n - \gamma_n \nabla f(x_n)$ 
End while
Do some random restarts
Return the lowest couple  $x_n, f(x_n)$  found.
```

- Для линейной регрессии с функцией потерь  $MSE$  он работает быстрее на больших наборах данных.
- Градиентный спуск годится для линейной регрессии и с другими функциями потерь (не для всех есть формулы).
- Он применим в обучении нейронных сетей, для которых тоже нет прямой формулы.