

FP-GROWTH И ЭФФЕКТИВНЫЕ АЛГОРИТМЫ АССОЦИАТИВНОГО АНАЛИЗА

ПРИМЕНЕНИЕ АЛГОРИТМА FP-GROWTH

1. ЦЕЛЬ ЛЕКЦИИ



Понять принцип работы FP-Growth.



Разобрать FP-tree, Header Table и node-link.



Понять Conditional Pattern Base и Conditional FP-tree.



Применять FP-Growth в Python.



Оценивать влияние min_support.

2. МЕСТО FP-GROWTH В DATA MINING

FP-Growth относится к Frequent Pattern Mining.

Основная задача — поиск часто встречающихся наборов.

Процесс: транзакции → поиск частых наборов → генерация правил → интерпретация.

Применения: e-commerce, web mining, медицина, промышленность, кибербезопасность.

3. ПОЧЕМУ ПОЯВИЛСЯ FP-GROWTH?

Apriori генерирует наборы-кандидаты и проверяет их.

При большом числе элементов количество комбинаций быстро растёт.

Потенциальное число непустых подмножеств: $2^n - 1$.

FP-Growth использует другую стратегию: сжатие данных и pattern growth.

4. ГЛАВНАЯ ИДЕЯ FP-GROWTH

Apriori: генерировать кандидатов → проверять.

FP-Growth: сжать транзакции → исследовать структуру дерева.

Общие префиксы транзакций объединяются.

Поиск выполняется через условные базы и условные FP-деревья.

5. ЧТО ТАКОЕ FP-TREE?

Компактное префиксное дерево.

Узел хранит элемент и счётчик.

Общие части транзакций используют общие ветви.

Node-link связывает узлы одного элемента.

Цель — уменьшить избыточность представления.

6. ПРИМЕР СЖАТИЯ ТРАНЗАКЦИЙ

Рассмотрим транзакции:

№	Товары
T1	Bread, Milk, Butter
T2	Bread, Milk
T3	Bread, Milk, Butter
T4	Bread, Eggs
T5	Milk, Butter

После сортировки элементов по частоте одинаковые начала транзакций объединяются.

Например:

T1: Bread → Milk → Butter

T2: Bread → Milk

T3: Bread → Milk → Butter

Общий префикс Bread → Milk хранится совместно.

Счётчики показывают число проходов по ветви.

7. ЭТАП 1 — ПОДСЧЁТ ЧАСТОТ

Сначала алгоритм проходит по базе транзакций и подсчитывает частоту каждого элемента.

Например:

Item	Frequency
Bread	4
Milk	4
Butter	3
Eggs	1

Подсчитывается частота каждого элемента.

Задаётся min_support.

Редкие элементы удаляются.

При minsup=2

Eggs исключается.

8. ЭТАП 2 — СОРТИРОВКА

В каждой транзакции оставляются частые элементы.

Они сортируются по убыванию глобальной частоты.

Это увеличивает общие префиксы.

Пример: Butter, Bread, Milk → Bread, Milk, Butter.

9. ЭТАП 3 — ПОСТРОЕНИЕ FP-TREE

Рассмотрим набор транзакций:

Транзакция	Элементы
T1	Bread, Milk, Butter
T2	Bread, Milk
T3	Bread, Milk, Butter
T4	Bread, Eggs
T5	Milk, Butter

После удаления Eggs:

T1: Bread Milk Butter
T2: Bread Milk
T3: Bread Milk Butter
T4: Bread
T5: Milk Butter

FP-tree может иметь структуру:

```
      Root
     /    \
Bread(4)  Milk(1)
   |        |
Milk(3)   Butter(1)
   |
Butter(2)
```

10. СТРУКТУРА FP-TREE



Root — корневой узел.



Item node — элемент и счётчик.



Parent/child links — структура дерева.



Node-link — связь узлов одного элемента.

11. HEADER TABLE

Одного дерева недостаточно.
Для быстрого доступа к узлам используется **Header Table**.

Item	Support	Node-link
Bread	4	→ узлы Bread
Milk	4	→ узлы Milk
Butter	3	→ узлы Butter

- Обеспечивает быстрый доступ к узлам.
- Хранит элементы и их поддержку.
- Node-link связывает соответствующие узлы.
- Позволяет быстро собрать пути выбранного элемента.

12. КАК FP-GROWTH ИЩЕТ ЗАКОНОМЕРНОСТИ

1. выбирает элемент;
2. находит все пути, связанные с этим элементом;
3. формирует **conditional pattern base**;
4. строит для неё **conditional FP-tree**;
5. рекурсивно ищет частые наборы.

13. CONDITIONAL PATTERN BASE

Conditional Pattern Base — набор префиксных путей FP-tree, связанных с выбранным элементом.

Для Butter:

Bread→Milk→Butter;

Bread→Milk→Butter;

Milk→Butter.

Conditional Pattern Base:

Bread, Milk : 2;

Milk : 1.

14. CONDITIONAL FP-TREE

Строится из Conditional Pattern Base.

Редкие элементы снова исключаются.

Затем выполняется рекурсивный поиск.

Из условного дерева извлекаются наборы, связанные с выбранным элементом.

15. ПОЛНАЯ СХЕМА FP-GROWTH

Транзакции → подсчёт частот → удаление редких элементов

↓ сортировка → FP-tree → Header Table

↓ Conditional Pattern Base

↓ Conditional FP-tree

↓ Frequent Itemsets

16. НЕБОЛЬШОЙ ПРИМЕР

Рассмотрим небольшую базу:

ID	Транзакция
T1	A, B, C
T2	A, B
T3	A, C
T4	A, B, C
T5	B, C

Пусть:
minsup=2

Частоты:

Item	Support
A	4
B	4
C	4

Все три элемента являются частыми.

ПОСТРОЕНИЕ ДЕРЕВА

Сортировка:

```
A → B → C
```

Транзакции:

```
T1: A B C
T2: A B
T3: A C
T4: A B C
T5: B C
```

После вставки транзакций получаем примерно:

```
          Root
         /    \
      A(4)    B(1)
     /   \       \
  B(3)  C(1)     C(1)
    |
  C(2)
```

Обратите внимание:

- `A → B → C` используется несколькими транзакциями;
- дерево сжимает общие префиксы;
- `C` встречается в нескольких ветвях.

17. FP-GROWTH VS APRIORI

Характеристика	Apriori	FP-Growth
Основной принцип	Генерация кандидатов	Рост частых шаблонов
Представление данных	Транзакции	FP-tree
Генерация кандидатов	Да	Нет в классическом виде
Количество проходов по БД	Может быть много	Обычно два основных прохода
Использование дерева	Нет	Да
Сжатие данных	Ограниченное	Да
Работа с большими данными	Может быть затратной	Часто эффективнее
Основная структура	Candidate itemsets	FP-tree

18. FP-GROWTH VS ECLAT

Apriori — горизонтальное представление и кандидаты.

FP-Growth — FP-tree.

Eclat — вертикальное TID-представление.

Eclat использует пересечения множеств идентификаторов транзакций.

КОГДА FP-GROWTH ОСОБЕННО ПОЛЕЗЕН?

FP-Growth подходит для задач, где:

- большое количество транзакций;
- много повторяющихся комбинаций;
- требуется поиск частых наборов;
- Apriori создаёт слишком много кандидатов;
- необходимо анализировать корзины покупок.

20. ОГРАНИЧЕНИЯ FP-GROWTH

Много уникальных элементов снижает эффект сжатия.

Очень низкий min_support даёт много наборов.

Разреженные данные могут давать менее компактное дерево.

FP-tree требует памяти.

Большое число результатов усложняет интерпретацию.

21. ПРАКТИЧЕСКИЙ КЕЙС

Интернет-магазин хочет найти товары, которые покупают вместе.

1: Laptop, Mouse, Bag

2: Laptop, Mouse

3: Laptop, Keyboard

4: Laptop, Mouse, Bag

5: Mouse, Bag

6: Laptop, Mouse

22. ПОДГОТОВКА ДАННЫХ В PYTHON

Для практической работы удобно использовать библиотеку **mlxtend**.

Python

```
pip install mlxtend pandas
```

Python

```
import pandas as pd

from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth
from mlxtend.frequent_patterns import association_rules
```

Создадим небольшой набор данных:

```
transactions = [
    ['Laptop', 'Mouse', 'Bag'],
    ['Laptop', 'Mouse'],
    ['Laptop', 'Keyboard'],
    ['Laptop', 'Mouse', 'Bag'],
    ['Mouse', 'Bag'],
    ['Laptop', 'Mouse']
]
```

Каждая строка представляет одну транзакцию.

ПРЕОБРАЗОВАНИЕ ДАННЫХ

Алгоритм работает с бинарным представлением.

Python

```
te = TransactionEncoder()

te_array = te.fit(transactions).transform(transactions)

df = pd.DataFrame(
    te_array,
    columns=te.columns_
)

print(df)
```

Получаем таблицу вида:

	Bag	Keyboard	Laptop	Mouse
0	1	0	1	1
1	0	0	1	1
2	0	1	1	0
3	1	0	1	1
4	1	0	0	1
5	0	0	1	1

1 означает наличие товара в транзакции.

РЕЗУЛЬТАТ FP-GROWTH

Результат может иметь структуру:

support	itemsets
0.83	{Laptop}
0.67	{Mouse}
0.50	{Bag}
0.67	{Laptop, Mouse}
0.33	{Laptop, Bag}
0.33	{Mouse, Bag}
0.33	{Laptop, Mouse, Bag}

Здесь support — доля транзакций, содержащих данный набор.

23. ЗАПУСК FP-GROWTH

```
from mlxtend.frequent_patterns import fpgrowth

frequent_itemsets = fpgrowth(
    df,
    min_support=0.3,
    use_colnames=True
)

print(frequent_itemsets)
```

Здесь:

min_support=0.3

означает, что набор должен встречаться минимум в **30% транзакций**.
У нас 6 транзакций.
Следовательно:

$$6 \times 0.3 = 1.8$$

То есть фактически набор должен встретиться минимум в 2 транзакциях.

24. ГЕНЕРАЦИЯ АССОЦИАТИВНЫХ ПРАВИЛ

После получения frequent itemsets можно сформировать правила:

```
from mlxtend.frequent_patterns import association_rules

rules = association_rules(
    frequent_itemsets,
    metric='confidence',
    min_threshold=0.6
)

print(rules[['antecedents',
             'consequents',
             'support',
             'confidence',
             'lift']])
```

Важно разделять два этапа:

```
FP-Growth
     ↓
Frequent Itemsets
     ↓
Association Rules
```

FP-Growth непосредственно ищет частые наборы, а не является самим этапом интерпретации правил.

26. ВЛИЯНИЕ MIN_SUPPORT

Высокий min_support → меньше наборов.

Низкий min_support → больше наборов.

Слишком высокий порог может скрыть закономерности.

Слишком низкий порог создаёт много малозначимых результатов.

Порог зависит от размера данных и предметной области.

ПРАКТИЧЕСКИЙ КЕЙС

Ситуация

Интернет-магазин хочет определить товары,
которые клиенты часто покупают вместе.

Дано:

```
order_id
product_id
product_name
quantity
date
```

Необходимо

1. Сформировать транзакции по order_id.
2. Удалить некорректные записи.
3. Преобразовать данные в transaction format.
4. Применить FP-Growth.
5. Использовать несколько значений min_support.
6. Сравнить количество найденных наборов.
7. Сформировать ассоциативные правила.
8. Проанализировать confidence и lift.
9. Выбрать наиболее интересные закономерности.
10. Предложить практические рекомендации.

27. ЭКСПЕРИМЕНТ С MIN_SUPPORT

supports = [0.05, 0.10, 0.20, 0.30]

Для каждого значения выполнить FP-Growth и определить количество найденных наборов.

```
for s in supports:
    result = fpgrowth(
        df, min_support=s,
        use_colnames=True
    )
    print(f"min_support={s}: "
          f"{len(result)} itemsets")
```

28. ПРАКТИЧЕСКИЙ PIPELINE

Исходные данные → очистка → транзакции

↓ бинарное представление → FP-Growth

↓ Frequent Itemsets → Association Rules

↓ фильтрация → интерпретация → рекомендации

29. ТИПИЧНЫЕ ОШИБКИ ПРИ ПРИМЕНЕНИИ FP-GROWTH

Не преобразовать данные в транзакционный формат.

Не удалить некорректные элементы.

Выбрать min_support без анализа.

Считать высокий support доказательством полезности.

Смешивать itemsets и rules.

Игнорировать практическую интерпретацию.

КЛЮЧЕВЫЕ ПРЕИМУЩЕСТВА FP-GROWTH

1. Не требует классической генерации кандидатов
Это одно из главных отличий от Apriori.
2. Использует компактное представление данных
FP-tree объединяет общие префиксы.
3. Хорошо работает с повторяющимися структурами
Чем больше общих префиксов, тем выше потенциал сжатия.
4. Позволяет рекурсивно находить частые шаблоны
Используются conditional pattern bases и conditional FP-trees.

ГЛАВНОЕ, ЧТО НЕОБХОДИМО ЗАПОМНИТЬ

FP-Growth:

$$Transactions \rightarrow FP\text{-}Tree \rightarrow Conditional\ FP\text{-}Tree \rightarrow Frequent\ Itemsets$$

Основные понятия:

- **FP-tree**
- **Header Table**
- **Node-link**
- **Conditional Pattern Base**
- **Conditional FP-tree**
- **Frequent Itemsets**
- **min_support**

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

Тема «Применение FP-Growth для выявления часто покупаемых товаров»
Исходные данные
Использовать открытый транзакционный набор данных либо сформировать собственный набор не менее чем из **100 транзакций**.
Выполнить:
1. Подготовку данных

- очистить данные;
- сформировать транзакции;
- преобразовать их в бинарную форму.

2. Применение FP-Growth
Проверить несколько значений:

```
min_support = 0.05
min_support = 0.10
min_support = 0.20
```

3. Сравнить результаты
Определить:

- количество frequent
- наиболее часто встр
  комбинации;
- изменение результатов при разных значениях min_support.

Python

```
association_rules()
```

4. Построить правила
Использовать:

5. Проанализировать

Для выбранных правил рассмотреть:

- support;
- confidence;
- lift.

6. Сделать вывод

Сформулировать минимум **5 практических рекомендаций** для выбранной предметной области.

31. КОНТРОЛЬНЫЕ ВОПРОСЫ

Что такое FP-tree?

Для чего нужна Header Table?

Что такое node-link?

Что такое Conditional Pattern Base?

Чем FP-Growth отличается от Apriori?

Как min_support влияет на результат?

В чём отличие FP-Growth от Eclat?

32. ГЛАВНОЕ

FP-Growth — алгоритм поиска частых наборов на основе pattern growth.

FP-tree сжимает транзакции за счёт общих префиксов.

Header Table и node-links ускоряют доступ.

Conditional Pattern Base → Conditional FP-tree → frequent itemsets.

После FP-Growth можно генерировать ассоциативные правила.