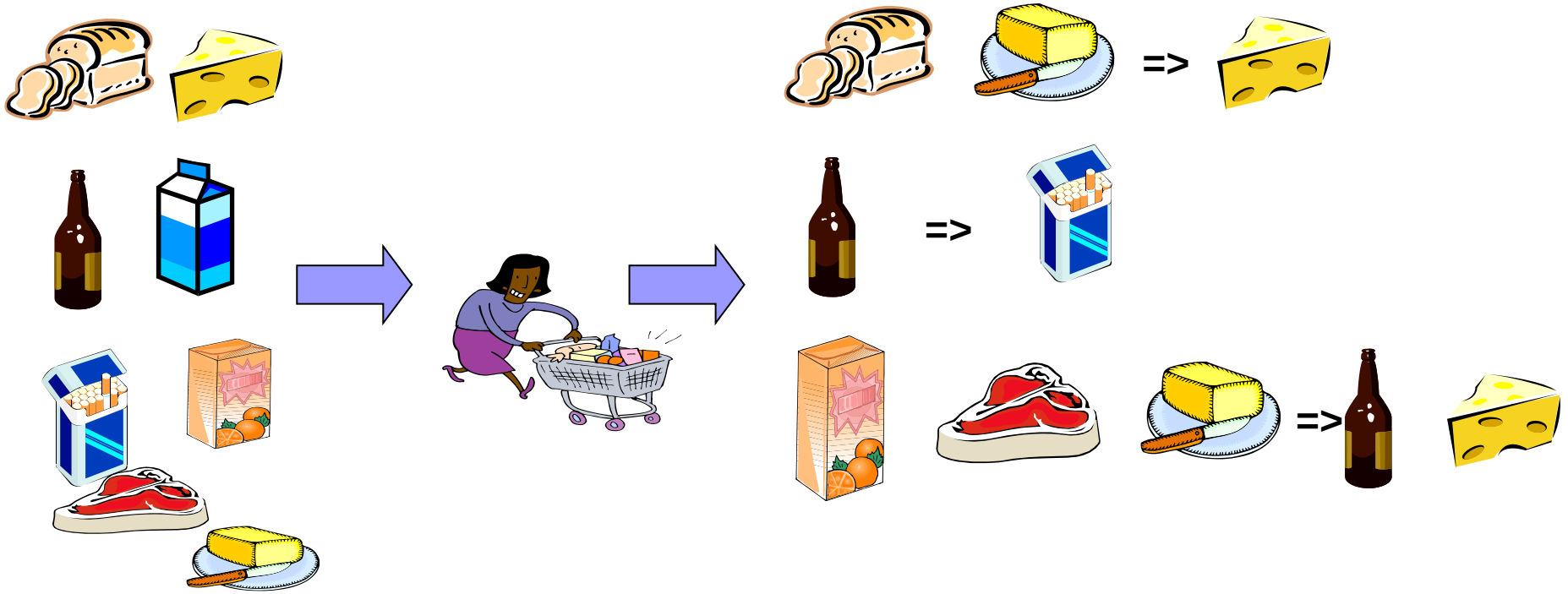
Лекция 2: Поиск ассоциативных правил

Типовая прикладная задача: анализ «корзины покупателя»

Ассортимент супермаркета

Интересные правила



Задача Определить интересные правила в предпочтениях покупателей при выборе товара

Ассоциативный анализ

- Правила с семантикой:
  - в s% случаях ЕСЛИ верно A_1, A_2 , …, A_k, ТО с достоверностью c будет верно B_1, B_2, …, B_m

$$\mathbf{A_1 \wedge A_2 \wedge \ldots \wedge A_k => B_1 \wedge B_2 \wedge \ldots \wedge B_m}$$

  - где A_1, A_2 , …, A_k, B_1, B_2, …, B_m – (различные!) предикаты,
  - s – поддержка (support), c – достоверность (confidence)
- Основная задача:
  - найти все интересные правила, с заданными ограничениями по s и c (возможно задание дополнительных ограничений на предикаты и сами правила)
- Основной математический аппарат:
  - дискретная математика, математическая логика, комбинаторная оптимизация (на основе метода «ветвей и границ» - вариации полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений).

Ассоциативный анализ

- Тип моделей:
  - Как правило, «описательный» (descriptive) Data mining => одна из задач - наглядное представление правил
- Тип обучения:
  - «без учителя» (unsupervised) => тренировочный набор не размечен
- Типы правил:
  - Булевы!!!
  - Числовые – нужна дискретизация, интервалы как булевы предикаты
  - Иерархические – если определена иерархия для значений атрибутов
  - Временные – как правило, семантика «в s случаях если произошло A и B, то потом случится C и D с вероятностью c»)
  - Пространственные – предикаты определяют пространственные связи между объектами, например «рядом», «далеко» и т.п.

Ассоциативный анализ

- Прикладные задачи:
  - «Экономические»: анализ корзины, маркетинг
  - «Безопасность» и Web usage mining: модели поведения пользователя
  - Text mining: поиск ключевых слов, характеристик и тематик
  - Биоинформатика, медицина
- Задачи анализа:
  - Поиск самих правил
  - Поиск исключений (из правил)
  - Выделение признаков (на основе правил)
  - Классификация и прогнозирование (на базе правил)

Булевы ассоциативные правила

$I = \{item_1, item_2, \ldots, item_n\}$ – множество атрибутов

D – множество транзакций $T \subseteq D$, каждая транзакция T – набор элементов из I,

Каждая транзакция T – бинарный вектор длины n: $(t_1, t_2, \ldots, t_n)$

$t_k=1$, если элемент $item_k$ присутствует в T, иначе $t_k=0$

Опр Транзакция T **содержит (поддерживает)** X – набор атрибутов из I, если $X \subseteq T$

Опр **Ассоциативным правилом** называется импликация $X \Rightarrow Y$, где $X, Y \subseteq I$ и $X \cap Y = \varnothing$

Опр Правило $X \Rightarrow Y$ имеет **поддержку (support) s**, если s% транзакций из D содержат X и Y

Опр Правило $X \Rightarrow Y$ имеет **достоверность (confidence) c**, если c% транзакций из D, содержащих X, также содержат Y.

Пример

D=

t	Хлеб	Кефир	Пиво	Чипсы
1	1	0	0	0
2	1	1	0	0
3	0	1	1	1
4	0	1	1	1
5	1	1	0	0
6	1	0	1	0
7	1	1	1	1
8	1	0	0	0
9	0	0	1	0
10	0	0	1	0

I={Хлеб, Кефир, Пиво, Чипсы}
supp(Хлеб) = 60%
supp(Кефир) = 50%
supp(Пиво) = 60%
supp(Чипсы) = 30%

Пример правила: Пиво=>Чипсы
supp(П=>Ч) = 30%
conf(П=>Ч) = 50%

Задача: Найти правила с параметрами:

minsupp = 30%,

minconf = 60%

Булевы ассоциативные правила

- Опр Найденные правила называются **интересными правилами**
- Опр Набор атрибутов X and Y называется **часто встречаемым** если supp(X and Y)>=minsupp

$I = \{i_1, i_2, ..., i_n\} - набор\ \ атрибутов$

Ассоциативное правило $X \Rightarrow Y$

$X \subseteq I, Y \subseteq I, X \cap Y = \{\}$

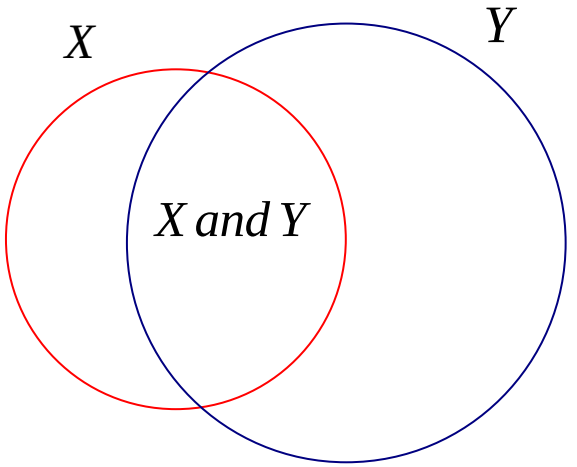
$support(X \Rightarrow Y) = p(X\ and\ Y)$

$$confidence(X \Rightarrow Y) = p(Y \mid X) = \frac{p(X\ and\ Y)}{p(X)}$$

$Задача: найти\ все\ ассоциативные\ правила\ \ с$

$support \geq MinSup\ и\ confidence >= MinConf$

Множество транзакций

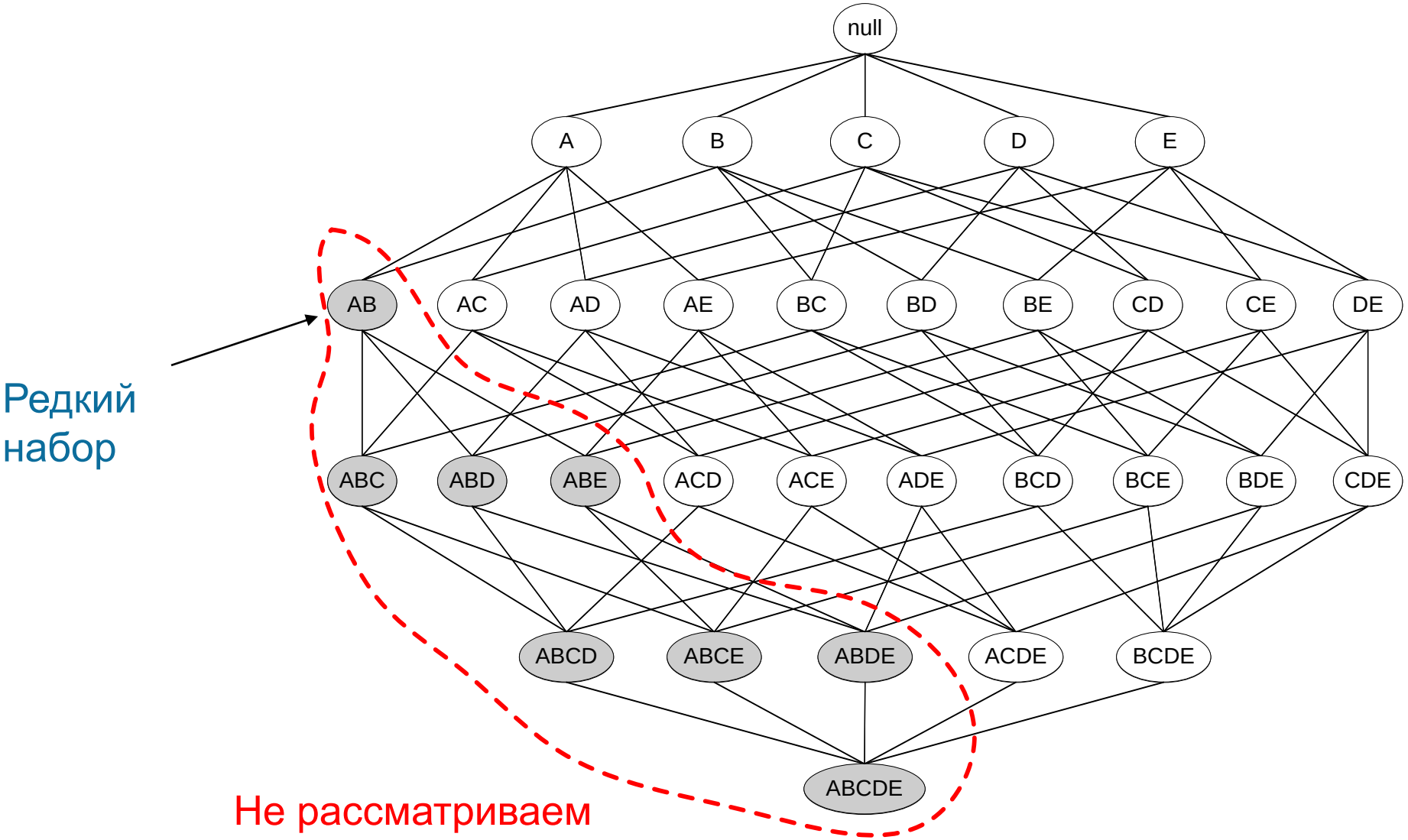


Популярные алгоритмы: Apriori, FP-tree

Алгоритм Apriori

- Основной принцип (анти-монотонность):
  - Любое подмножество часто встречаемого набора является часто встречаемым набором
- Формально:
  - Поддержка любого набора элементов не может превышать минимальной поддержки всех его подмножеств
  - Необходимое условие частой встречаемости k-элементного набора – частая встречаемость всех его (k-1)-элементных подмножеств
  - В примере: supp({Хлеб, Кефир, Чипсы}) <= supp({Хлеб, Кефир}), supp({Хлеб, Чипсы}),supp({Кефир, Чипсы}), supp({Кефир}), supp({Чипсы}),supp({Хлеб})
- Этапы алгоритма:
  - Генерация множества часто встречаемых наборов (supp >= minsupp): метод «ветвей и границ» - направленный перебор от простых (коротких) наборов к сложным (длинным) с отсечением
  - Генерация правил по найденным наборам (conf >= minconf)

Идея метода ветвей и границ для Apriori



Генерация множества часто встречаемых наборов атрибутов

L_1 = {часто встречаемые 1-элементные наборы}

for (k = 2; L_{k-1} != {}; k++) {

  C_k = apriori-gen(L_{k-1}); // Генерация k-элементных кандидатов

  forall transactions t $\in$ D {

    C_t = subset(C_k, t); // Кандидаты, которые содержатся в транзакции t

    forall candidates c $\in$ C_t

      c.count++;

  }

  L_k = {c $\in$ C_k | c.count >= minsupp }

}

Answer = $\bigcup_k L_k$

Генерация кандидатов apriori-gen

Ck = apriori-gen(Lk-1)

Шаг JOIN

INSERT INTO C_k

SELECT $p.item_1$, $p.item_2$, …, $p.item_{k-1}$, $q.item_{k-1}$

FROM L_{k-1} p, L_{k-1} q

WHERE $p.item_1 = q.item_1$, …, $p.item_{k-2} = q.item_{k-2}$,

$p.item_{k-1} < q.item_{k-1}$;

- **Шаг PRUNE**

  forall itemsets c $\in$ C_k

  forall (k-1)-subsets s of c

  if (s $\notin$ L_{k-1}) delete c from C_k;

Пример генерации кандидатов

- L_3={*abc, abd, acd, ace, bcd*}
- Join: L_3*L_3
  - *abcd* = *abc* + *abd*
  - *acde* = *acd* + *ace*
- Pruning:
  - *acde* удален, т.к. *ade* не в L_3
- C_4={*abcd*}

Пример

D=

t	Хлеб	Кефир	Пиво	Чипсы
1	1	0	0	0
2	1	1	0	0
3	0	1	1	1
4	0	1	1	1
5	1	1	0	0
6	1	0	1	0
7	1	1	1	1
8	1	0	0	0
9	0	0	1	0
10	0	0	1	0

Построение L1
supp(Хлеб) = 60%
supp(Кефир) = 50%
supp(Пиво) = 60%
supp(Чипсы) = 30%
L1 = {{Х}, {К}, {П}, {Ч}}

Построение L2
{Х, К}, {Х, П}, {Х, Ч}
{К, П}, {К, Ч}, {П, Ч}
supp({Х, К}) = 30%
supp({Х, П}) = 20%
supp({Х, Ч}) = 10%
supp({К, П}) = 30%
supp({К, Ч}) = 30%
supp({П, Ч}) = 30%
L2={{Х,К}, {К,П}, {К,Ч}, {П,Ч}}

Пример

D=

t	Хлеб	Кефир	Пиво	Чипсы
1	1	0	0	0
2	1	1	0	0
3	0	1	1	1
4	0	1	1	1
5	1	1	0	0
6	1	0	1	0
7	1	1	1	1
8	1	0	0	0
9	0	0	1	0
10	0	0	1	0

L2={{Х,К}, {К,П}, {К,Ч}, {П,Ч}}

Формируем L3

{К, П, Ч}

supp({К, П, Ч}) = 30%

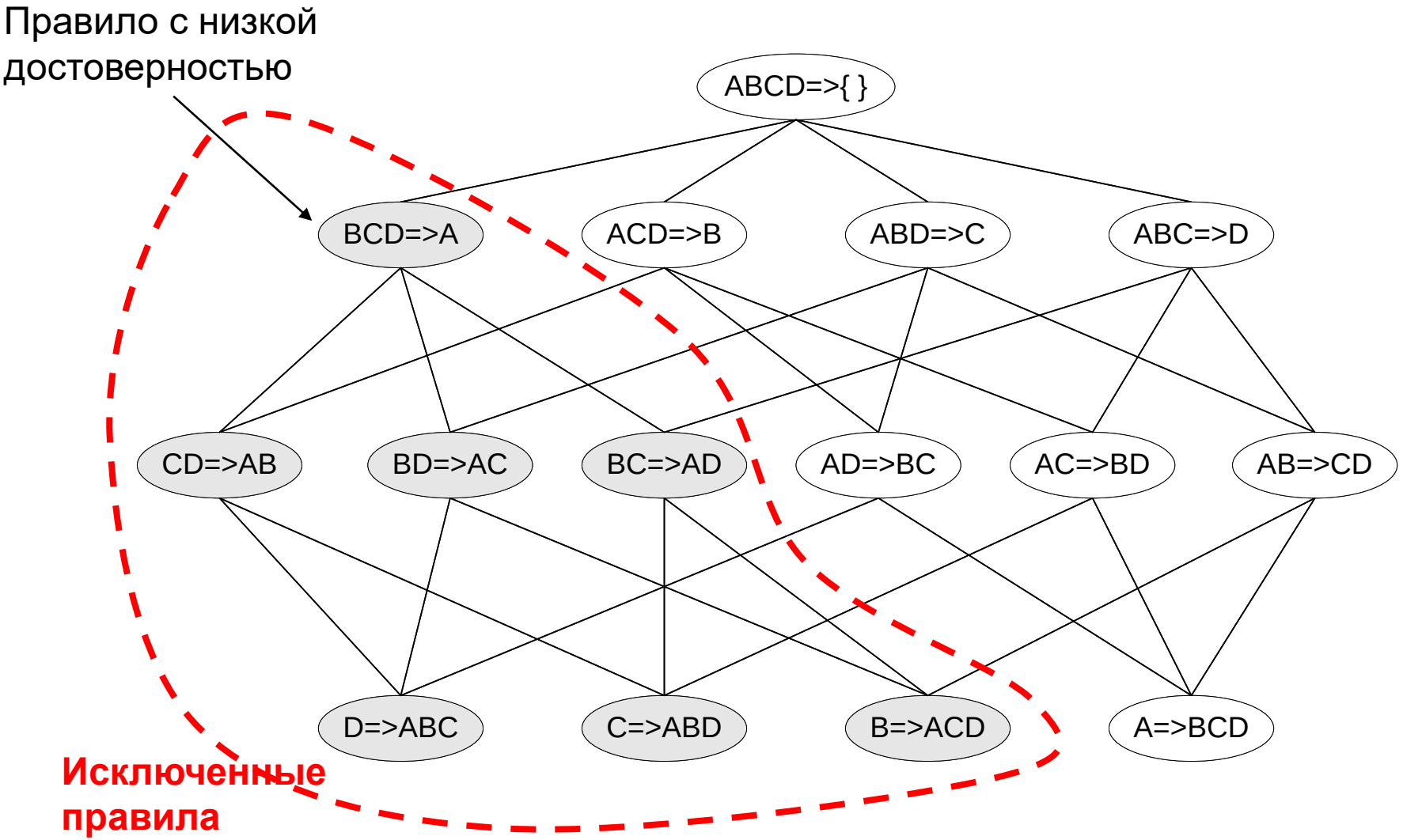
L3 = {{К, П, Ч}}

Результат=
{{Х}60%,{К}50%,{П}60%,{Ч}30%,
{Х,К}30%,{К,П}30%,{К,Ч}30%,
{П,Ч}30%,{К, П, Ч}30%}

Генерация правил

- Критерий:
  - conf(X=>Y) = P(Y|X) = support({X,Y}) / support(X)
  - conf(X=>Y)>=minconf
  - все support известны с 1-го этапа
- Принцип:
  - Если правило {A} => {B, C} интересно, то и {A, B} => {C} интересно
- Доказательство:
  - conf({A}=>{B, C}) = supp({A, B, C}) / support({A})>=minconf
  - conf({A, B}=>{C}) = supp({A, B, C}) / support({A, B})
  - support({A, B}) <= supp({A})
  - conf({A, B}=>{ C})>=minconf
- Алгоритм:
  - Для каждого часто встречаемого набора проверять правила на интересность, начиная со случая, когда в правой части правила находится один атрибут и постепенно добавлять/убавлять атрибуты в/из правую/левой часть(и).

Метод ветвей и границ для генерации правил



Пример

D=

t	Хлеб	Кефир	Пиво	Чипсы
1	1	0	0	0
2	1	1	0	0
3	0	1	1	1
4	0	1	1	1
5	1	1	0	0
6	1	0	1	0
7	1	1	1	1
8	1	0	0	0
9	0	0	1	0
10	0	0	1	0

Наборы:

- {Х}60%,{К}50%,{П}60%, {Ч}30%, **{Х,К}30%,{К,П}30%,{К,Ч}30%,{П,Ч}30%, {К, П, Ч}30%**

Правила:

conf({Х}=>{К})=50%

conf({К}=>{Х})=60%

conf({К}=>{П})=60%

conf({П}=>{К})=50%

conf({К}=>{Ч})=60%

conf({Ч}=>{К})=100%

conf({П}=>{Ч})=50%

conf({Ч}=>{П})=100%

conf({К, П}=>{Ч})=100%

conf({К}=>{П, Ч})=60%

conf({П}=>{К, Ч})=50%

conf({К, Ч}=>{П})=100%

conf({Ч}=>{К, П})=100%

conf({П, Ч}=>{К})=100%

Недостатки Apriori

- Суть алгоритма Apriori:
  - Использовать часто встречаемые наборы размера (k – 1) для генерации кандидатов часто встречаемых наборов размера k
  - Использовать dbscan и сравнения подмножеств атрибутов для расчета поддержки кандидатов
- Слабое место – генерация кандидатов
  - Огромное число кандидатов: 10^4 1-элементных наборов приводят к 10^7 2-элементным наборам, если надо найти наборы размера 100 $\{a_1, a_2, \ldots, a_{100}\}$, нужно сгенерировать $2^{100} \approx 10^{30}$ кандидатов.
  - Множественные dbscan: (n +1) сканирований, где n - длина наибольшего набора
- Пути решения:
  - Хэш-деревья для хранения наборов и счетчиков поддержки
  - Удаление неинформативных транзакций из базы
  - Разбиение базы и sampling - набор будет часто встречаемым, если он часто встречаемый на каком-то подмножестве транзакций, но: необходима оценка полноты и достоверности

Поиск частых наборов без кандидатов

- Основная задача, решаемая методом Frequent-Pattern tree (FP-tree):
  - «сжать» информацию о транзакциях и представить в «компактном» виде с быстрым поиском частых наборов (FP-tree)
  - уйти от частых сканирований БД, не генерировать кандидатов, а искать их динамически по структуре FP-tree
  - стратегия «разделяй и властвуй»: декомпозиция задачи поиска на более мелкие подзадачи – рекурсивное построение «пути» частых наборов в FP-tree дереве
- Свойства и требования к структуре:
  - «сжатая» информация для поиска наборов должна быть полной
  - размер вспомогательных структур не должен превосходить размер БД,
  - не должно быть несодержательной информации, например, о редких наборах
  - при поиске обратная упорядоченность по частоте наборов и атрибутов – более часто встречаемые атрибуты с большой вероятностью являются частью частых наборов

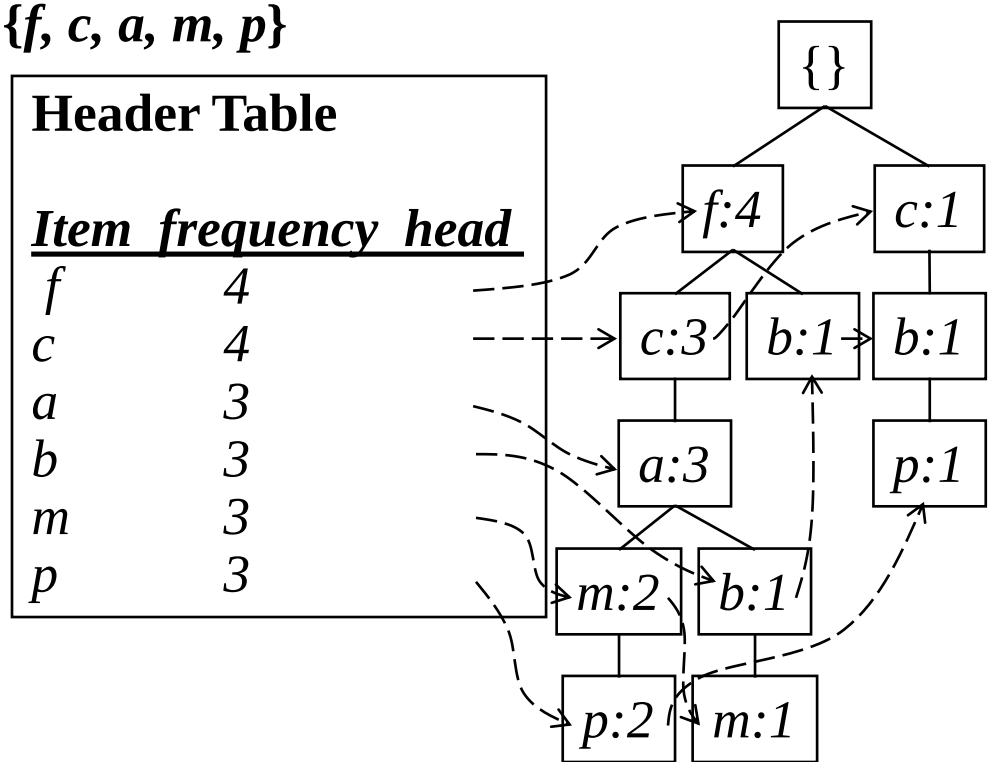
Построение FP-tree

TID	*Items*	*frequent items*
100	{*f, a, c, d, g, i, m, p*}	{*f, c, a, m, p*}
200	{*a, b, c, f, l, m, o*}	{*f, c, a, b, m*}
300	{*b, f, h, j, o*}	{*f, b*}
400	{*b, c, k, s, p*}	{*c, b, p*}
500	{*a, f, c, e, l, p, m, n*}	{*f, c, a, m, p*}

min_support = 0.5

Шаги:

1. Первое сканирование БД и построение частых 1-наборов
2. Обратная сортировка по частоте
3. Второе сканирование и построение FP-tree

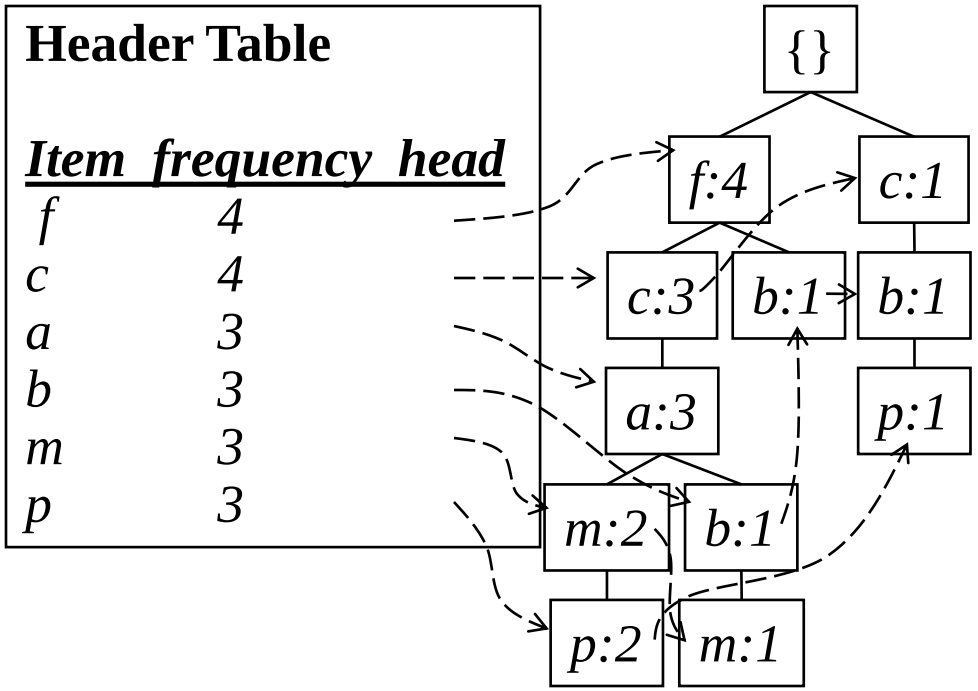


Поиск частых наборов с FP-tree

- Метод:
  - Для каждого элемента найти его условный базовый набор
  - На основе условного базового набора построить новое условное FP-tree поддерево для каждого элемента, рассматривая каждый путь как отдельную транзакцию
  - Повторить процесс для элементов каждого вновь созданного условного FP-tree поддерева
  - До тех пор пока результирующее FP-tree не будет пусто или не будет содержать единственный путь
  - Единственный путь генерирует все комбинации подпутей, каждый из которых есть частый набор

Шаг 1: От FP-tree к условному базовому набору

- Для каждого элемента проход FP-tree «вверх» по дугам с запоминанием «условного» пути и его поддержки
- В результате с каждым элементом связан условный базовый набор (набор возможных путей к вершине с поддержкой)



***Conditional* pattern bases**

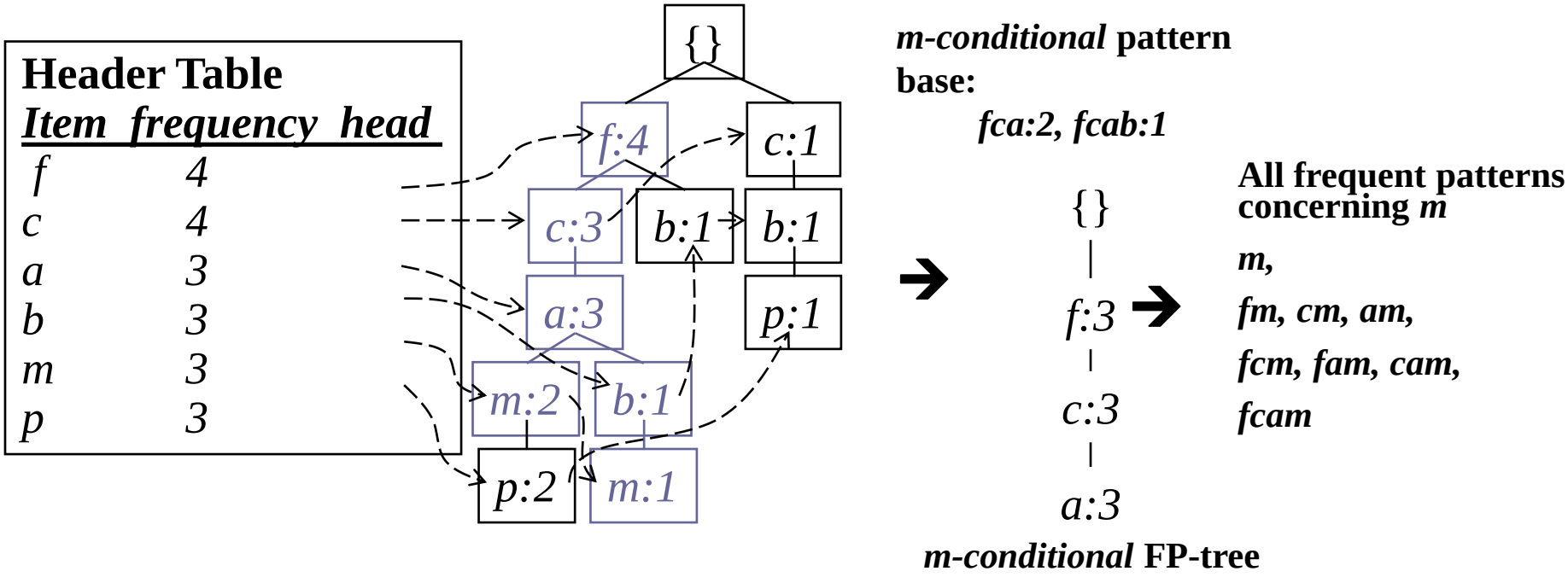
item	***cond. pattern base***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

Свойства условного FP-tree

- Свойство «узел-связь»:
  - Для каждого частого элемента a_i, все возможные частые наборы, содержащие a_i , могут быть получены обходом по пути «узел-связь» от a_i‘-го заголовочного элемента к корню FP-tree
- Свойство префикса:
  - Для поиска частых наборов для узла a_i в пути P, необходимо рассматривать только префикс под-пути от a_i в P, его поддержка должна быть равна поддержке узла a_i.

Шаг 2: Построение условного FP-tree

- Для каждого условного базового набора
  - Построить условное FP-tree, содержащее только пути из базового набора

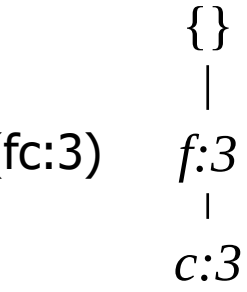


Поиск частых наборов по условным базовым наборам

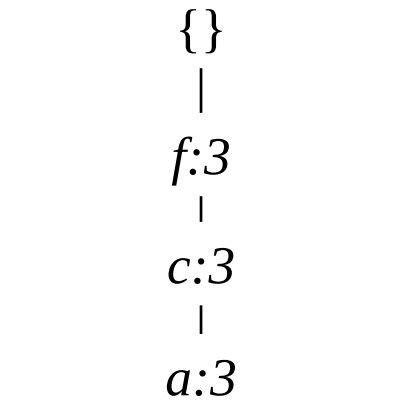
Item	Условный базовый набор	Условное FP-tree
p	{(fcam:2), (cb:1)}	{(c:3)}|p
m	{(fca:2), (fcab:1)}	{(f:3, c:3, a:3)}|m
b	{(fca:1), (f:1), (c:1)}	Empty
a	{(fc:3)}	{(f:3, c:3)}|a
c	{(f:3)}	{(f:3)}|c
f	Empty	Empty

Шаг 3: Рекурсивная обработка условного FP-tree

Условный базовый набор для “am”: (fc:3)

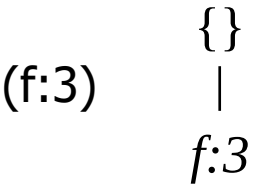


***am-conditional* FP-tree**



***m-conditional* FP-tree**

Условный базовый набор для“cm”: (f:3)



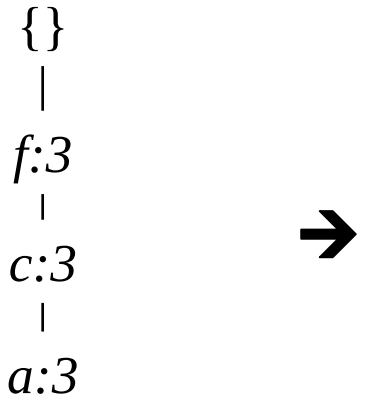
***cm-conditional* FP-tree**

Условный базовый набор для“cam”: (f:3)

***cam-conditional* FP-tree**

Единственный путь в FP-tree

- Предположим в FP-tree T есть единственный путь P
- Полное множество частых наборов из T могут быть получены перебором всех возможных комбинаций подпутей из P



→

All frequent patterns concerning *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

***m-conditional* FP-tree**

Принцип поиска частых наборов

- Свойство увеличения набора
  - Пусть α - частый набор в DB, B α's - условный базовый набор для α и β - поднабор в B. Тогда $\alpha \cup \beta$ есть частый набор в DB тогда и только тогда, когда β - частый набор в B.
- “*abcdef* ” - частый набор тогда и только тогда, когда
  - “*abcde* ” - частый набор
  - “*f* ” - частый элемент для множества транзакций, содержащих “*abcde* ”

Преимущества FP-tree перед Apriori

- Экспериментально:
  - FP-tree значительно (на некоторых задачах - на порядок) быстрее Apriori
- Причина
  - Нет генерации и проверки кандидатов
  - Используется компактная структура для поиска частых наборов и расчета поддержки
  - Нет повторяющихся сканирований БД
  - Основные операции – суммирование и построение дерева

Критика Support и Confidence

- Пример: (Aggarwal & Yu, PODS98)
  - Среди 5000 студентов
    - 3000 играют баскетбол
    - 3750 любят черный хлеб
    - 2000 и то и другое
  - *basketball* $\Rightarrow$ *bread* [40%, 66.7%] вводит в заблуждение, поскольку процент любителей хлеба 75% выше support 66.7%.
  - *basketball* $\Rightarrow$ *not bread* [20%, 33.3%] более полезное, хотя support and confidence ниже

	basketball	not basketball	sum(row)
bread	2000	1750	3750
not bread	1000	250	1250
sum(col.)	3000	2000	5000

Критика Support и Confidence

- Пример:
  - X и Y: положительно коррелированны,
  - X и Z, отрицательно коррелированны
  - support и confidence больше у X=>Z
- Нужна мера зависимости:

$$corr_{A,B} = \frac{P(A and\ B)}{P(A)P(B)}$$

- P(B|A)/P(B)
  - A => B

X	1	1	1	1	0	0	0	0
Y	1	1	0	0	0	0	0	0
Z	0	1	1	1	1	1	1	1

Rule	Support	Confidence
X=>Y	25%	50%
X=>Z	37,50%	75%

Itemset	Support	Interest
X,Y	**25%**	**2**
X,Z	**37,50%**	**0,9**
Y,Z	**12,50%**	**0,57**

Объективные меры интересности

1) $support(X \Rightarrow Y) = P(X \cap Y)$

2) $confidence(X \Rightarrow Y) = P(Y \mid X)$

3) $generality(X \Rightarrow Y) = P(Y)$

4) $lift(X \Rightarrow Y) = \frac{P(X \cap Y)}{P_{EXP}(X \cap Y)} = \frac{P(X \cap Y)}{P(X)P(Y)} = \frac{P(Y \mid X)}{P(Y)} = \frac{confidence(X \Rightarrow Y)}{P(Y)}$

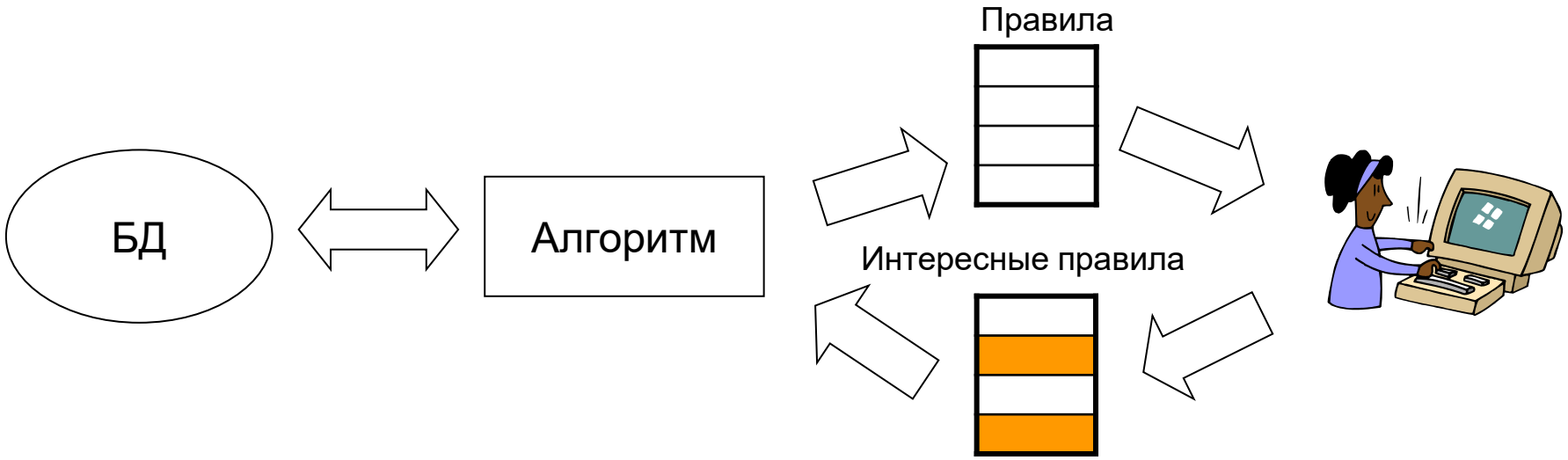
5) $RI(X \Rightarrow Y) = P(Y \mid X) - P(Y) = confidence(X \Rightarrow Y) - generality(X \Rightarrow Y)$

6) $J(X \Rightarrow Y) = P(Y)[P(Y \mid X)\log_2 \frac{P(Y \mid X)}{P(Y)} + (1 - P(Y \mid X))\log_2 \frac{1 - P(Y \mid X)}{1 - P(Y)}]$ (J-measure)

$D(p(x), q(x)) = \sum_{x \in X} p(x) \log \frac{p(x)}{p(y)}$ (Kullback, Leibler)

Интересность

- Объективная
- Субъективная (на основе информации, заданной экспертом)
  - «Полезная» (Actionable)
  - «Неожиданная» (Unexpected)



Использование ограничений

- Проблема итеративного анализа больших объемов данных:
  - невозможно без использования ограничений
- Типы ограничений:
  - стандартные: на support и confidence
  - на меры объективной интересности
  - на выборку «горизонтально» - подмножества транзакций
  - на выборку «вертикально» - подмножества атрибутов
  - на значения отдельных атрибутов (с точки зрения алгоритма аналогично «вертикальному»)
  - шаблоны правил для поиска – метаправила (задаются экспертом, учитываются методом «ветвей и границ» при генерации наборов и правил из них, сокращается перебор)
  - шаблоны «неинтересных» правил (поиск «неожиданных» правил, нарушающих шаблон) – для оценки субъективной интересности

Mlxtend (ML extensions)

- Обновляемая библиотека с широким спектром моделей ML
- Документация: http://rasbt.github.io/mlxtend/

- В курсе рассматривается:
  - Модуль предобработки **preprocessing**
  - Модуль анализа ассоциативных правил **frequent_patterns** (Apriori, Fpgrowth, …)

- Установка и импорт библиотеки
  - Установка через pip
    ```
    pip install mlxtend
    ```
  - Импорт через python
    ```
    from mlxtend.<module> import ...
    ```

Загрузка и обработка данных

- Набор транзакций:
  - CUSTOMER – Id покупателя
  - TIME – временная метка
  - PRODUCT – Target (элемент)
- Формирование списка транзакций:
  - Группировка по id: **`.groupby("CUSTOMER")`**
  - Организация списка: **`.aggregate({"PRODUCT":list})`**
  - Получение значений: **`.values[:, 0]`**

	CUSTOMER	TIME	PRODUCT
0	0.0	0.0	hering
1	0.0	1.0	corned_b
2	0.0	2.0	olives
3	0.0	3.0	ham
4	0.0	4.0	turkey

```
array([list(['hering', 'corned_b', 'olives', 'ham', 'turkey', 'bourbon', 'ice_crea']),
       list(['baguette', 'soda', 'hering', 'cracker', 'heineken', 'olives', 'corned_b']),
       list(['avocado', 'cracker', 'artichok', 'heineken', 'ham', 'turkey', 'sardines']),
       ...,
       list(['baguette', 'soda', 'hering', 'cracker', 'heineken', 'sardines', 'sardines']),
       list(['sardines', 'heineken', 'chicken', 'coke', 'ice_crea', 'corned_b', 'peppers']),
       list(['hering', 'corned_b', 'apples', 'olives', 'steak', 'bourbon', 'heineken'])],
      dtype=object)
```

Загрузка и обработка данных

- Кодирование транзакций:
  - **`preprocessing.TransactionEncoder`**
  - Матрица вхождения: **`fit_transform()`**

```
te = TransactionEncoder()
te_ary = te.fit_transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)
```

te_ary

```
[[False, False, False, ..., False, False,  True],
 [False, False, False, ...,  True, False, False],
 [False,  True,  True, ..., False, False,  True],
 ...,
 [False, False, False, ...,  True, False, False],
 [False, False, False, ..., False, False, False],
 [ True, False, False, ..., False,  True, False]]
```

df

	apples	artichok	avocado	baguette	bordeaux	bourbon	chicken	coke
0	False	False	False	False	False	True	False	False
1	False	False	False	True	False	False	False	False
2	False	True	True	False	False	False	False	False
3	False	False	False	False	False	True	False	True
4	True	False	True	False	False	False	False	False

Поиск ассоциативных правил

- Построение модели **`apriori:`**
  - Ограничение support: min_support
  - Макс. длина правил: max_len
- Формирование правил с помощью **`association_rules`**
  - metric ('support', 'confidence', 'lift', 'leverage', 'conviction')
  - Мин. значение метрики: min_threshold

`frequent_itemsets`

	support	itemsets
0	0.313686	(apples)
1	0.304695	(artichok)
2	0.362637	(avocado)
3	0.391608	(baguette)
4	0.402597	(bourbon)
5	0.314685	(chicken)

```
association_rules(frequent_itemsets,
                  metric="support",
                  min_threshold=0.2).head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(artichok)	(avocado)	0.304695	0.362637	0.210789	0.691803	1.907700	0.100295	2.068038
1	(avocado)	(artichok)	0.362637	0.304695	0.210789	0.581267	1.907700	0.100295	1.660497
2	(artichok)	(heineken)	0.304695	0.599401	0.251748	0.826230	1.378426	0.069114	2.305336

Поиск ассоциативных правил

```
# Чтение данных
path_to_file = "../assoc.sas7bdat"
df = pd.read_sas(path_to_file, encoding='latin-1')

# Формирование списков транзакций
transactions = df.groupby('CUSTOMER').aggregate({"PRODUCT":list}).values[:, 0]

# Кодирование транзакций
te = TransactionEncoder()
te_ary = te.fit_transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

# Построение модели
frequent_itemsets = apriori(df, min_support=0.1, use_colnames=True)

# Формирование правил
association_rules(frequent_itemsets, metric="support", min_threshold=0.2)
```

Поиск последовательностей

- Правила вида $\mathbf{A_1 => A_2 => \ldots => A_k}$
  - С семантикой «После события A_i следует A_j»
  - Требует признак с ролью sequence
  - Есть дополнительные параметры на временные окна
  - Нельзя корректно рассчитать lift
  - Поддержка задается отдельно
  - Можно использовать в рекомендательных системах
- Алгоритм поиска последовательностей:
  - Сначала поиск частых эпизодов (без учета времени)
  - Потом из частых эпизодов выделяются многоместные правила «следования» с учетом времени
  - Возможны правила вида A=>A

Поиск последовательностей и анализ графа связей

- Для узлов (событий) можно рассчитать разные полезные характеристики:
  - Число связей (входящих, исходящих, всего)
  - Коэфициент кластеризации (реальное число связей ближайших соседей деленное на максимально возможное число связей между ними) показывает находится ли узел внутри плотной группы
  - Хабы – узлы, из которых много ссылок на важные узлы (авторитеты)
  - Авторитеты – важные узлы, на которые ссылается много хабов
  - Близость – усредненное значение кратчайших путей до всех узлов в графе
  - Внутренность – показывает как часто этот узел встречается в кратчайших путях между другими узлами
  - Собственная центральность (Eigenvector Centrality) – уровень исходящих и входящих связей с учетом важности связанных узлов

networkx (Network Analysis)

- Обновляемая библиотека для работы с графами
- Документация: https://networkx.org/

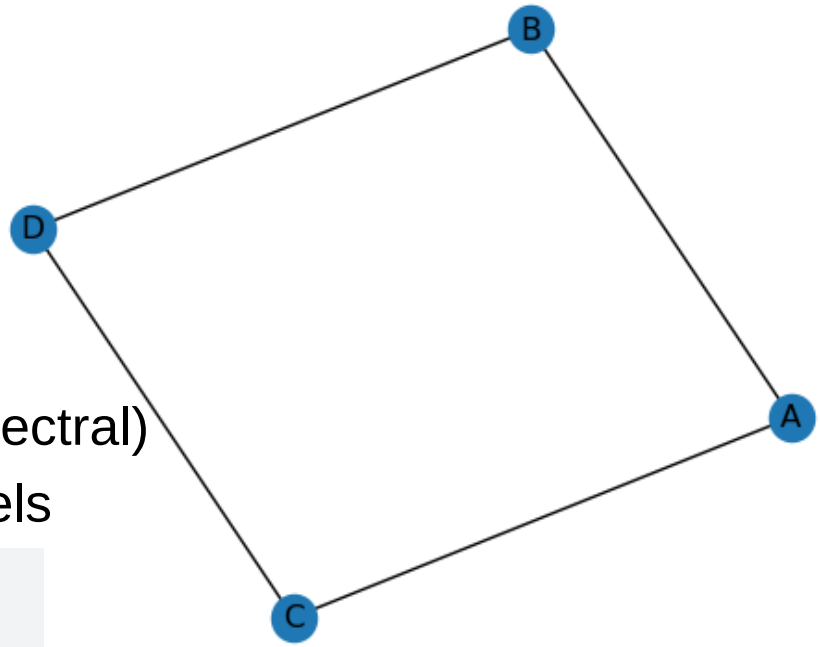
- В курсе рассматривается:
  - Построение и визуализация графов
  - Расчет мер центральности

- Установка и импорт библиотеки
  - Установка через pip
    `pip install mlxtend`
  - Импорт через python
    `import network as nx`

Графовые операции

- Типы графов:
  - Неориентированные: Graph, MultiGraph
  - Ориентированные: DiGraph, MultiDiGraph
- Построение графа:
  - Новая вершина: **`.add_node`**
  - Новое ребро: **`.add_edge`**
  - Можно добавлять вес
- Визуализация:
  - Выбор полотна (spring, spiral, spectral)
  - Визуализация nodes, edges, labels

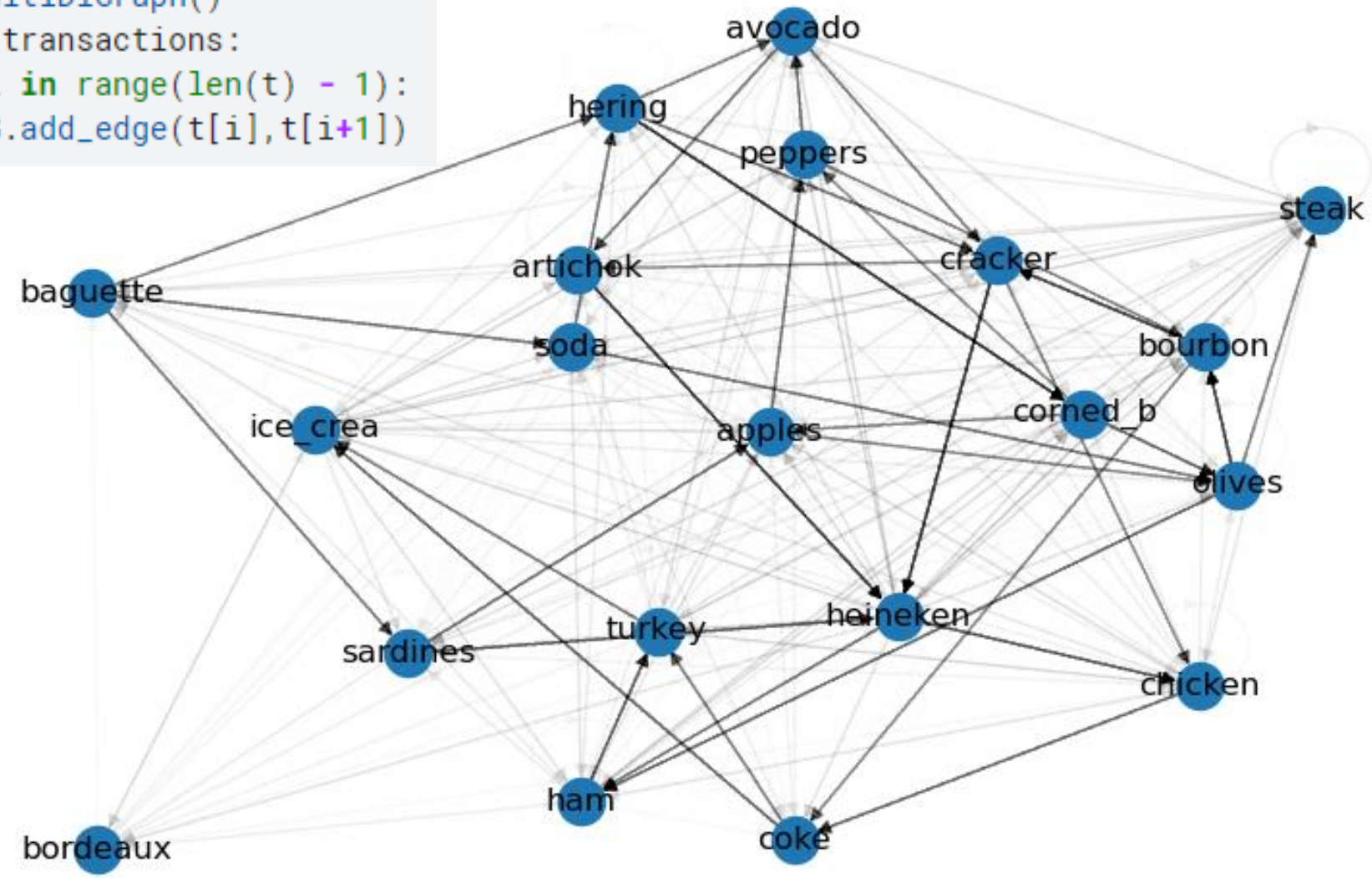
```
G = nx.Graph()
G.add_edge("A", "B")
G.add_edge("B", "D")
G.add_edge("A", "C")
G.add_edge("C", "D")
```




```
pos = nx.spring_layout(G)
nodes = nx.draw_networkx_nodes(G, pos)
edges = nx.draw_networkx_edges(G, pos)
labels = nx.draw_networkx_labels(G, pos)
```

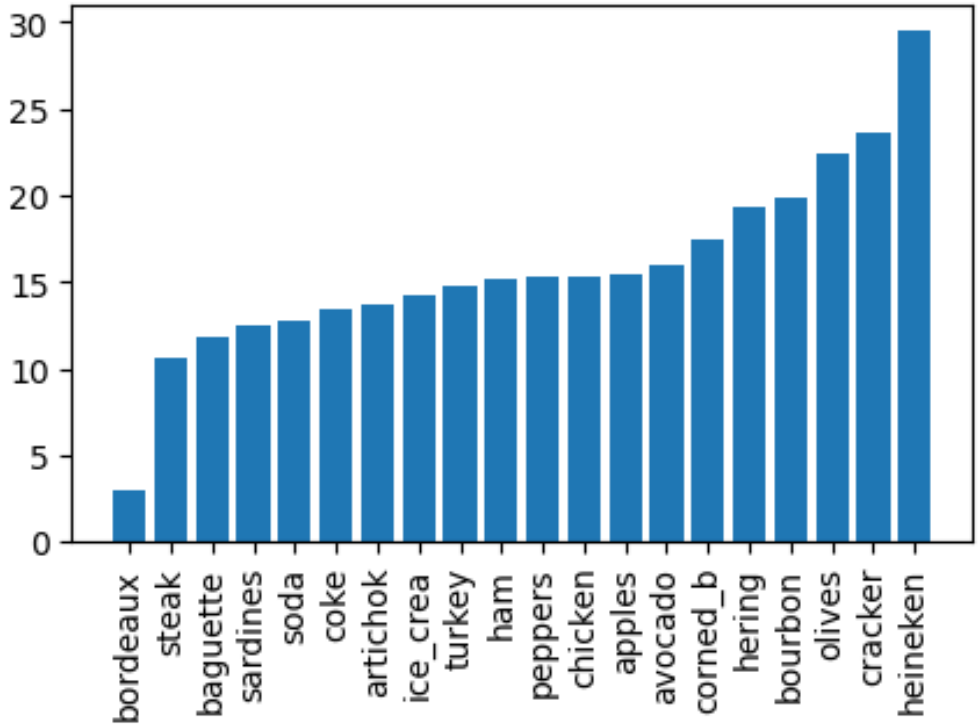
Пример графа транзакций

```
G = nx.MultiDiGraph()
for t in transactions:
    for i in range(len(t) - 1):
        G.add_edge(t[i],t[i+1])
```



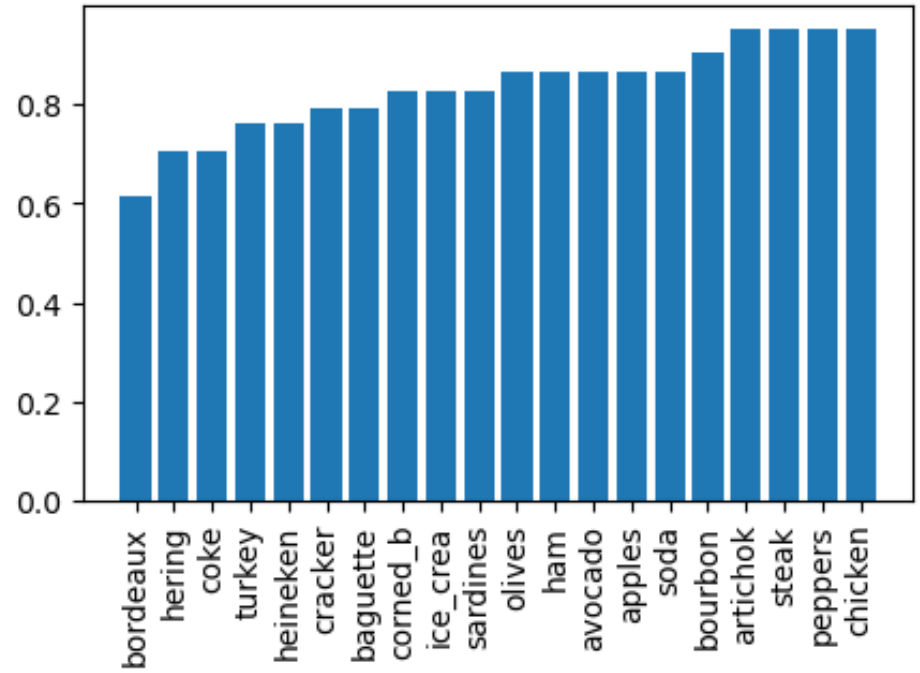
Меры центральности

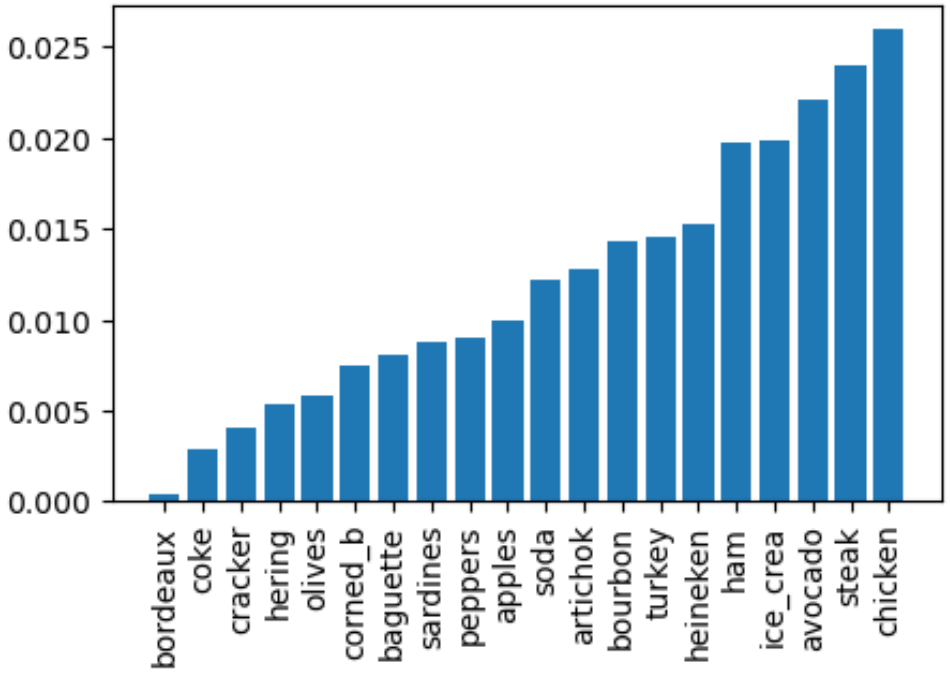
- Degree:
  - degree_centrality(G)
  - in_degree_centrality(G)
  - out_degree_centrality(G)
- Eigenvector:
  - eigenvector_centrality(G)
  - katz_centrality(G)
- closeness_centrality(G)
- Betweenness
- Group Centrality




```
nx.degree_centrality(G)
```

Меры центральности






```
nx.closeness_centrality(G)
```

```
nx.betweenness_centrality(G)
```

```
metr = dict(sorted(metr.items(), key=lambda x: x[1]))
plt.xticks(rotation='vertical')
plt.bar(metr.keys(), metr.values())
```

Визуализация

Поиск последовательностей и анализ графа связей

Алгоритм HITS:
Шаг 0. Инициализация. $\forall \boldsymbol{n}\ \mathbf{auth}(\boldsymbol{n}) = \mathbf{1}, \mathbf{hub}(\boldsymbol{n}) = \mathbf{1}$
Шаг 1. Авторитетности (обычно несколько итераций с sampling по n):
$\mathbf{auth}(\boldsymbol{n}) = \sum_{p \in Neib(n)} \mathrm{hub}(p)$
Шаг 2. Посредничество (обычно несколько итераций с sampling по n):
$\mathbf{hub}(\boldsymbol{n}) = \sum_{p \in Neib(n)} \mathrm{auth}(p)$
Шаг 3. Нормализация (разные схемы, иначе алг. может расходиться)
Если не выполнено условие остановки (по числу итераций, по времени, по отклонению нормы всех hub и auth) **то перейти на Шаг 1.**

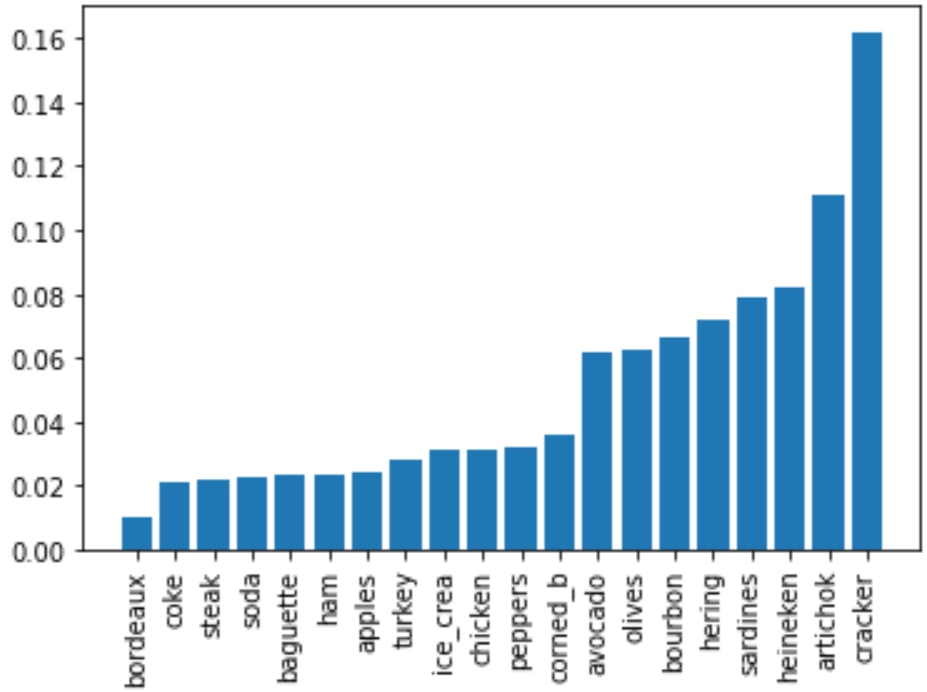
* Соседи определяются с учетом направления дуг

Алгоритм Eigenvector Centrality - поиск наибольшего с.зн. матрицы смежности (1 – есть связь, 0 – нет), причем у с.в. все координаты(они и есть оценка Eigenvector Centrality) неотрицательные.

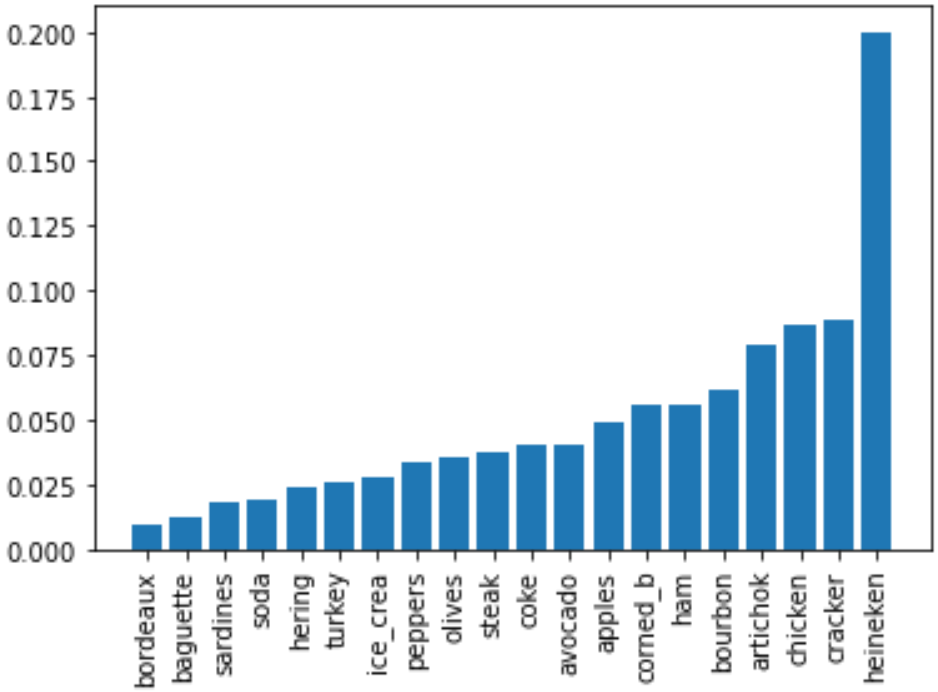
Пример расчета HITS

```
h, a = nx.hits(G)
```

hub



auth

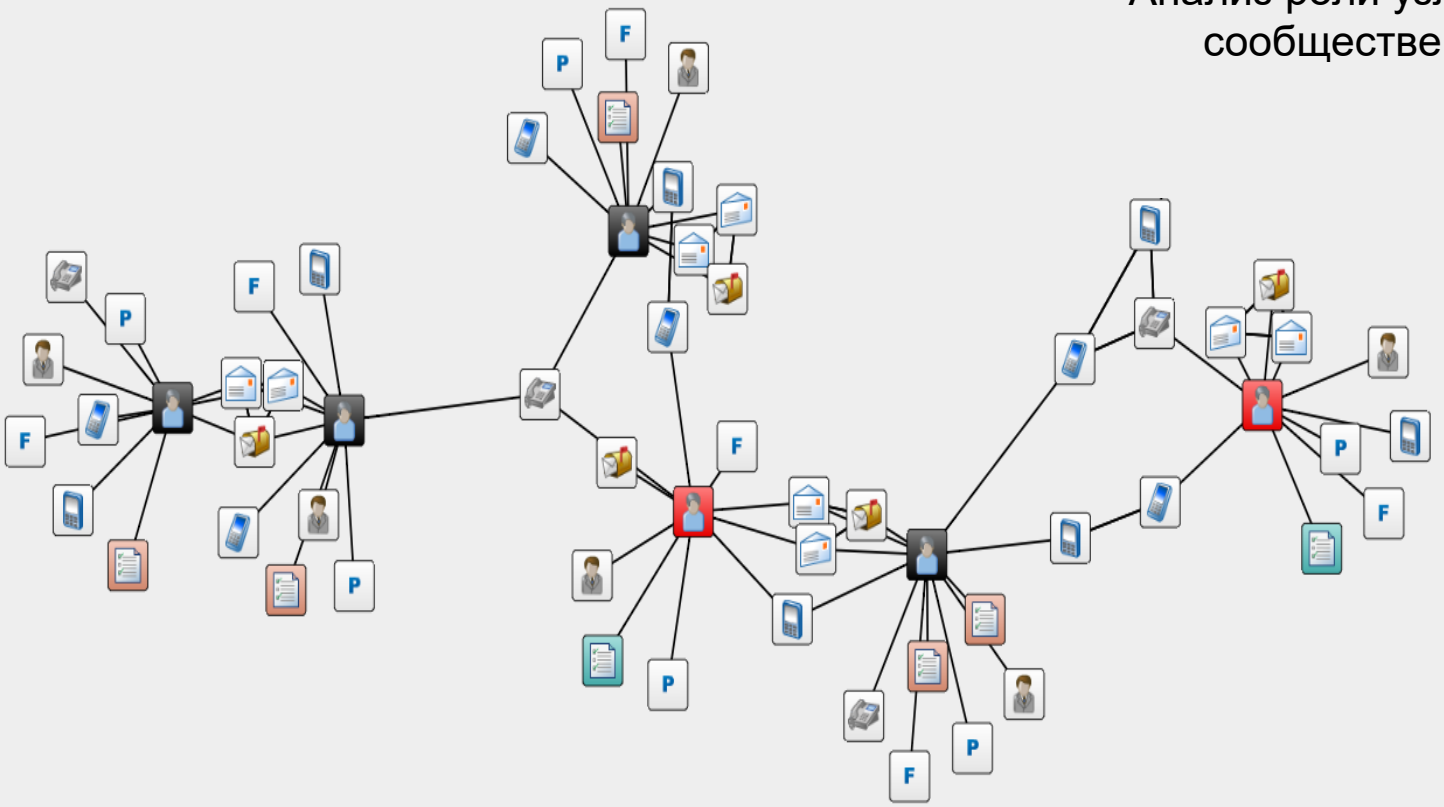


АНАЛИЗ ГРАФОВ

Построение графа на основе любых типов связей

Выявление тесно связанных сообществ

Анализ роли узла в сообществе



МЕТРИКИ ЦЕНТРАЛЬНОСТИ

Количественная характеристика узла, отражающая его роль в социальной сети

Типы метрик:

- Degree
- Influence
- Clustering Coefficient
- Closeness
- Betweenness
- Eigenvector Centrality
- Hub and Authority

Роль	Описание
Лидер	Высокое значение Closeness и Betweeness
Последователь	Высокое значение Closeness и Betweeness, но ниже чем у лидера
Трансфер	Высокое значение betweeness, но низкая centrality
"Крупная рыба"	Высокое значение Eigenvector
Распространитель	Высокое значение Authority
Случайный игрок	Низкое значение betweeness и низкое значение degree