Поиск ассоциативных правил. Алгоритм Apriori, FP-tree growth

Сергей Лисицын

19 апреля 2011 г.

Задача поиска ассоциативных правил

- Имеется некоторое множество транзакций, наборов событий, предметов или других сущностей
- Необходимо найти статистически устойчивые «связи» между подмножествами таких транзакций
- Если считать «появление» некоторого подмножества в транзакции случайным событием – то мы будем искать подмножества, для которых наблюдается корреляция (в интуитивном смысле этого слова)

Формальная задача поиска ассоциативных правил

Пусть имеется конечное множество признаков M, некоторое множество $T = \{t_1, t_2, \ldots, t_n\}$, где элементы $t_i \subset M$ называются транзакциями. Ассоциативным правилом называется случайное событие

$$s \Rightarrow d \quad \sim \quad (d \subset t | s \subset t)$$

имеющая смысл события «если s принадлежит t, то и d принадлежит t», для которой определена функция поддержки

$$support(s \Rightarrow d) = \frac{|\{t : s \cup d \subseteq t \in T\}|}{|T|}$$

Достоверностью ассоциативного правила называется функция

$$confidence(s, d) = \frac{support(s \cup d)}{support(s)},$$

Алгоритмы поиска ассоциативных правил

- Прецедентные по своей природе – работают с конкретными данными, не рассматривая вероятностные пространства
- Основным алгоритмом для решения этой задачи является Apriori
- Нет смысла говорить об обобщающей способности или других особенностях обучения

Анализ потребительских корзин

market basket analysis

- Область, в которой впервые решалась задача поиска ассоциативных правил
- Любая транзакция – некоторое множество покупок одним клиентом
- Полученные правила позволяют принимать решения, положительно сказывающиеся на продажах: начиная от расположения «связанных товаров» заканчивая различными акциями

Другие схожие задачи

- Предсказание поведения клиентов компаний
- Поиск связей в текстах
- Другие задачи поиска связей между объектами и событиями
- ...

Алгоритм Apriori

- Один из базовых алгоритмов поиска ассоциативных правил
- Впервые предложен в работе Ракеша Аграваля (R. Agrawal), представителя IBM
- Основан на правиле антимонотонности (downward closure lemma), позволяющем априорно отсечь наборы, не удовлетворяющие минимальной поддержке
- Высокие требования к памяти и вычислительным мощностям
- В контексте алгоритма используют следующие понятия:
  - Набор – некоторое подмножество множества признаков M
  - Расширение набора – результат объединения наборов

Свойство антимонотонности

downward closure lemma

Если для некоторых наборов $A, B \subseteq M$: $A \subseteq B$, то

$$support(A) \geq support(B)$$

- Поддержка набора при его расширении никогда не возрастает
- Свойство существенно уменьшает вычислительную сложность поиска правил при обычном комбинаторном подсчёте

Описание алгоритма

первый этап, нахождение частых наборов

Вход: база транзакций T, минимальный уровень поддержки $support_{min}$
Выход: подмножества элементов транзакционной базы, удовлетворяющие $support_{min}$

1. найти все пары (одноэлементный набор – поддержка) в транзакционной базе T
2. отсечь из полученных пар все элементы, не удовлетворяющие минимальной поддержке $support_{min}$
3. пока множество $L \neq \emptyset$ не пусто
   - 3.1 добавить L к результату
   - 3.2 $L = L \times L$ (декартово произведение, cross join, исключая повторы как внутри наборов, так и самих наборов)
   - 3.3 отсечь элементы L, не удовлетворяющие поддержке

Описание алгоритма

второй этап, нахождение ассоциативных правил

Вход: частовстречающиеся наборы транзакционной базы, минимальный уровень достоверности $confidence_{min}$
Выход: кортежи $(s, d, confidence(s \Rightarrow d))$, удовлетворяющие минимальному уровню достоверности $confidence_{min}$

1. для всех непересекающихся подмножеств полученных наборов по формуле

$$confidence(s, d) = \frac{support(s \cup d)}{support(s)}$$

   вычислить достоверность и добавить полученный кортеж (s,d, достоверность $s \Rightarrow d$) в результат

2. отсечь из результата элементы, не удовлетворяющие минимальной достоверности $confidence_{min}$

Пример работы алгоритма

Поиск частых наборов

Пусть транзакционная база $T = \{(\mathbf{A}), (\mathbf{A}, \mathbf{B}), (\mathbf{A}, \mathbf{C}), (\mathbf{A}, \mathbf{B}), (\mathbf{A}), (\mathbf{A}, \mathbf{B}, \mathbf{C}), (\mathbf{C}, \mathbf{D})\}$, а уровень поддержки $support_{min} = 2$:

Набор	Частота
(A)	6
(B)	3
(C)	3
(D)	1

Пример работы алгоритма

Поиск частых наборов

Пусть транзакционная база $T = \{(\mathbf{A}), (\mathbf{A}, \mathbf{B}), (\mathbf{A}, \mathbf{C}), (\mathbf{A}, \mathbf{B}), (\mathbf{A}), (\mathbf{A}, \mathbf{B}, \mathbf{C}), (\mathbf{C}, \mathbf{D})\}$, а уровень поддержки $support_{min} = 2$:

Набор	Частота
(A)	6
(B)	3
(C)	3
(D)	1

$\Rightarrow$

Набор	Частота
(A,B)	3
(A,C)	2
(B,C)	1

Пример работы алгоритма

Поиск частых наборов

Пусть транзакционная база $T = \{(\mathbf{A}), (\mathbf{A}, \mathbf{B}), (\mathbf{A}, \mathbf{C}), (\mathbf{A}, \mathbf{B}), (\mathbf{A}), (\mathbf{A}, \mathbf{B}, \mathbf{C}), (\mathbf{C}, \mathbf{D})\}$, а уровень поддержки $support_{min} = 2$:

Набор	Частота
(A)	6
(B)	3
(C)	3
(D)	1

$\Rightarrow$

Набор	Частота
(A,B)	3
(A,C)	2
(B,C)	1

$\Rightarrow$

Набор	Частота
(A,B,C)	1

Пример работы алгоритма

Нахождение правил по частым наборам

Набор	Частота
(A,B)	3
(A,C)	2
(A)	6
(B)	4
(C)	3

Правило	Достоверность
$(A) \Rightarrow (B)$	0.5
$(A) \Rightarrow (C)$	0.33

```
def frequentItemset(transactions):
    freq_itemset = {}
    for itemset in transactions:
        for item in itemset:
            key = tuple([item])
            if freq_itemset.has_key(key): freq_itemset[key]+=1
            else: freq_itemset[key]=1
    return freq_itemset

def pruneBySupport(itemset,support):
    return filter(lambda item: itemset[item]>support,itemset)

def unifyItemset(itemset):
    return list(set([tuple(set(tuple(x)+tuple(y)))
                     for x in itemset
                     for y in itemset
                     if not x==y]))
```

```
def countFrequencies(transactions, itemset):
    freq_itemset = dict([(x,0) for x in itemset])
    for element in transactions:
        for item in freq_itemset.keys():
          if set(item).issubset(set(element)): freq_itemset[item]+=1
    return freq_itemset

def apriori(transactions,support):
    L1 = frequentItemset(transactions)
    L = pruneBySupport(L1,support)
    result = []
    while len(L)>0:
        result.append(L)
        L = unifyItemset(L)
        L = countFrequencies(transactions,L)
        L = pruneBySupport(L,support)
    return result
```

Эффективность алгоритма

- Действительно огромное количество требуемой памяти (особенно при низких уровнях поддержки)[1]
- Кроме требований по памяти, высокие требования по времени вычисления
- Алгоритм отлично работает для небольших баз, но не для очень больших
- Все существующие модификации этого алгоритма (AprioriTID, AprioriHybrid, ...) всё равно схожи с перебором

[1] нулевая поддержка в базе размером N на k-ом шаге вызовет создание и подсчёт частот N^k элементов $\sim O(N^{k+1})$

Недостатки

- Требуется постоянная генерация возможных наборов, сканирование множества наборов, отсечение, отсюда огромное количество необходимой памяти и времени вычисления
- По полученным частым наборам необходимо производить поиск правил (не менее трудоёмкий)
- Наиболее эффективным решением проблем Apriori является замена его алгоритмом построения FP-деревьев

Метод создания FP-деревьев

FP (frequent pattern) tree growth

- Эффективное решение, преодолившее недостатки производительности алгоритма Apriori
- Хранит всю необходимую информацию в виде дерева
- Построение дерева производится за два полных обхода всего множества транзакций
- Для нахождения ассоциативных правил используется построенное дерево

Описание алгоритма
Построение дерева

Вход: множество транзакций D, минимальный уровень поддержки $support_{min}$
Выход: FP-дерево

1. Найти поддержку для каждого одиночного элемента, отбросить элементы с недостаточной поддержкой
2. Отсортировать одиночные элементы по убыванию поддержки в список *FList*
3. Создать корень дерева T и пометить его пустым
4. Для каждой транзакции d из D
   - Отсортировать элементы d в порядке, соответствующем порядку в *FList*
   - Представить d в виде $[p|P]$, где p – первый элемент, P – остальные и выполнить Insert-tree$([p|P], T)$

Описание алгоритма
Вставка

Insert-tree $([p|P], T)$

1. Если у T имеется потомок N соответствующий p – увеличить его поддержку
2. Иначе создать потомок узла T – N, соответствующий p, проинициализировать его поддержку 1 и связать узел N со всеми узлами с такими же именами
3. Если P не пусто, запустить алгоритм рекурсивно Insert-tree (P, N)

Эффективность

- FP-дерево существенно быстрее алгоритма Apriori
- Для большинства баз данных FP дерево достаточно компактно и «умещается» в памяти
- В случае большого размера построенного дерева можно использовать B+ деревья

Источники

- Wasilewska A., "Apriori algorithm"
- Воронцов К.В., «Лекции по логическим алгоритма классификации»
- Kohonen A., "FP-Tree"