数据挖掘实验实验报告

实验二　Apriori 算法

姓 名: 柴 博 文
学 号: 04194012
班 号: 大数据 1901

数据挖掘与机器学习
(秋季, 2021)

西安邮电大学
计算机学院
数据科学与大数据专业

2021 年 10 月 27 日

摘 要

本次实验代码均可以在github 仓库下找到.

目 录

1 概述

- **算法简述** Apriori 算法将会从你所给的数据集当中生成频繁项集, 频繁项集为由数据集当中原子数据组成的不重复集合, 并计算这些项集在数据集当中出现的频率 (即当前项集的支持度), 随后在将这些项集中出现频率大于最小支持度的去掉, 在将剩下的进行组合, 将组合的项集再进行计算, 知道在也不能组合生成新的项集. 在频繁项集生成之后, 再使用频繁项集生成关联规则 (对于频繁项集 X, Y(两者), 同时出现 (X,Y) 的概率, 相对与只出现 Y 的频率).
- **算法流程**首先需要从数据集当中生成频繁 1 项集, 计算频繁 n 项集的支持度, 再通过当前频繁 n 项集计算频繁 n 项集 (合并当前频繁 n-1 项集, 不合并多项不同的), 然后再遍历所有的频繁项集, 求频繁 n 项集和频繁 n-k 项集的差集, 在算出支持度和提升度

2　频繁项集生成

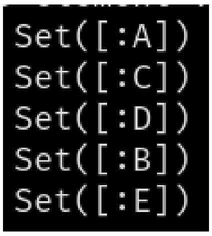
首先是从数据集生成频繁 1 项集

```
# 合并数据集中的每一项，并进行散列去重
C1::Set{Set{T}} = (∪(data...) .|> v -> Set([v])) |> Set
dict = scan(data, C1, ε) # 生成的频繁1项集的支持度散列表
```

对于数据形如

index	Vector
hline 1	:A, :C, :D
2	:B, :C, :E
3	:A, :B, :C, :E
4	:B, :E

生成的频繁一项集机器支持度 (去除了支持度小于 0.5 的项): 图2.



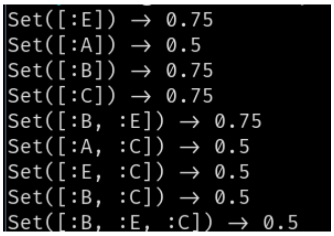
```
Set([:A])
Set([:C])
Set([:D])
Set([:B])
Set([:E])
```

图 1: 频繁 1 项集的生成

接下来, 就是根据频繁 n-1 项集生成频繁 n 项集

```
for p ∈ L[n-1], q ∈ L[n-1]
    if p != q
        push!(C, p ∪ q) # union sets
    end
end
```

然后对频繁 n 项集进行支持度计算



```
Set([:E]) → 0.75
Set([:A]) → 0.5
Set([:B]) → 0.75
Set([:C]) → 0.75
Set([:B, :E]) → 0.75
Set([:A, :C]) → 0.5
Set([:E, :C]) → 0.5
Set([:B, :C]) → 0.5
Set([:B, :E, :C]) → 0.5
```

图 2: 频繁项集的支持度

scan 函数参数: data: 即数据集 C: 频繁 n 项集 ε: 最小支持度 scan 函数返回值: 返回频繁 n 项集的 KV 表, V -> 支持度

```julia
function scan(
    data::Vector{Vector{T}},
    C::Set{Set{T}},
    ε::Float64,
)::Dict{Set{T},Float64} where {T}
    dict = Dict{Set{T},Float64}()
    data .|> row -> C .|> set -> if set ⊆ row # check each set is subset of each row in data which in C
        v = get!(dict, set, 0) + 1 # Int, yes, set default if is missing, and get value
        dict[set] = v
    end
    len = data |> length # Int, get data set size
    for (key, value) ∈ dict # Vector{T} => Float64, cal each sets support%
        support = value / len
        dict[key] = support
    end
    dict |> keys .|> key -> if dict[key] < ε # remove set which support less than ε
        delete!(dict, key)
    end
    dict
end
```

3 关联规则生成

关联规则的生成需要所有频繁 $[1-n]$ 项集的支持度, 以及 $[1-n]$ 级频繁项集对应的支持度表

进行便利操作, 对于频繁 n 项集的表中的每一个元素 n-set, 与所有频繁项集的每一个 set 分别取差集,

n-set $\Longrightarrow$ set 即为一个关联关系

rules 函数的入参: dicts: 频繁 n 项集与其支持度的表的集合 support: 所有频繁项集与其支持度的集合 ς: 最小置信度

rules 函数的返回值所有关联规则与其支持度和提升度组成的表

```
Set([:E]) ⇒ Set([:B, :C]) → (0.6666666666666666, 1.3333333333333333)
Set([:C]) ⇒ Set([:E]) → (0.6666666666666666, 0.8888888888888888)
Set([:E, :C]) ⇒ Set([:B]) → (1.0, 1.3333333333333333)
Set([:E]) ⇒ Set([:B]) → (1.0, 1.3333333333333333)
Set([:A]) ⇒ Set([:C]) → (1.0, 1.3333333333333333)
Set([:B, :E]) ⇒ Set([:C]) → (0.6666666666666666, 0.8888888888888888)
Set([:B]) ⇒ Set([:E]) → (1.0, 1.3333333333333333)
Set([:C]) ⇒ Set([:B, :E]) → (0.6666666666666666, 0.8888888888888888)
Set([:C]) ⇒ Set([:A]) → (0.6666666666666666, 1.3333333333333333)
Set([:B, :C]) ⇒ Set([:E]) → (1.0, 1.3333333333333333)
Set([:B]) ⇒ Set([:E, :C]) → (0.6666666666666666, 1.3333333333333333)
Set([:C]) ⇒ Set([:B]) → (0.6666666666666666, 0.8888888888888888)
Set([:B]) ⇒ Set([:C]) → (0.6666666666666666, 0.8888888888888888)
Set([:E]) ⇒ Set([:C]) → (0.6666666666666666, 0.8888888888888888)
```

图 3: 关联规则结果图 (数据同上)

```julia
function rules(
    dicts::Vector{Dict{Set{T},Float64}},
    support::Dict{Set{T}, Float64},
    ς::Float64
   )::Dict{Pair{Set{T},Set{T}}, Tuple{Float64, Float64}} where {T}
    result = Dict{Pair{Set{T}, Set{T}},Tuple{Float64, Float64}}()
    for i ∈ 2:(dicts |> length)
        for x ∈ 1:i-1
            for set ∈ dicts[x] |> keys
                for n_set ∈ dicts[i] |> keys
                    if set ⊆ n_set
                        diff = setdiff(n_set, set)
                        # set ∪ diff = n_set
                        # confidence(set => diff) = count(n_set) / count(set)
                        # (n_set, set => diff, support[n_set] / support[set]) |> println
                        confidence = support[n_set] / support[set]
                        if confidence >= ς
                            result[set => diff] = confidence, confidence / support[diff]
                        end
                    end
```

```
                    end
                end
            end
        end
        result
    end
```

4 代码

```julia
#=
main:
- Julia version: 1.6.3
- Author: lqxc
- Date: 2021-10-19
=#

# D means the whole dataset, which should be Vector{Vector{T}}
# ! count(X) = filter(D, X ⊆ _).count()
# support(X => Y) = count(X ∪ Y) / |D|
# confidence(X => Y) = count(X ∪ Y) / count(X)

function rules(
    dicts::Vector{Dict{Set{T},Float64}},
    support::Dict{Set{T}, Float64},
    ς::Float64
   )::Dict{Pair{Set{T},Set{T}}, Tuple{Float64, Float64}} where {T}
    result = Dict{Pair{Set{T}, Set{T}},Tuple{Float64, Float64}}()
    for i ∈ 2:(dicts |> length)
        for x ∈ 1:i-1
            for set ∈ dicts[x] |> keys
                for n_set ∈ dicts[i] |> keys
                    if set ⊆ n_set
                        diff = setdiff(n_set, set)
                        # set ∪ diff = n_set
                        # confidence(set => diff) = count(n_set) / count(set)
                        # (n_set, set => diff, support[n_set] / support[set]) |> println
                        confidence = support[n_set] / support[set]
                        if confidence >= ς
                            result[set => diff] = confidence, confidence / support[diff]
                        end
                    end
                end
            end
        end
    end
    result
end

#=
# scan the data with each item set in item set set,
# and get each support
=#

function scan(
    data::Vector{Vector{T}},
    C::Set{Set{T}},
    ε::Float64,
```

```julia
)::Dict{Set{T},Float64} where {T}
    dict = Dict{Set{T},Float64}()
    data .|> row -> C .|> set -> if set ⊆ row # check each set is subset of each row in data which in C
        v = get!(dict, set, 0) + 1 # Int, yes, set default if is missing, and get value
        dict[set] = v
    end
    len = data |> length # Int, get data set size
    for (key, value) ∈ dict # Vector{T} => Float64, cal each sets support%
        support = value / len
        dict[key] = support
    end
    dict |> keys .|> key -> if dict[key] < ε # remove set which support less than ε
        delete!(dict, key)
    end
    dict
end

function apriori(
    data::Vector{Vector{T}};
    ε::Float64 = 0.5,
    ς::Float64 = ε,
)::Tuple{
    Vector{Dict{Set{T}, Float64}},
    Dict{Pair{Set{T},Set{T}}, Tuple{Float64, Float64}}
} where {T}
    n = 2
    C1::Set{Set{T}} = (∪(data...) .|> v -> Set([v])) |> Set
    support = Dict{Set{T}, Float64}()
    dict = scan(data, C1, ε)
    merge!(+, support, dict)
    L::Vector{Set{Set{T}}} = [[(dict |> keys)...] |> Set]
    D::Vector{Dict{Set{T},Float64}} = [dict]
    while L[n-1] |> length != 0
        C = Set{Set{T}}()
        # len = L[n-1] |> length
        for p ∈ L[n-1], q ∈ L[n-1]
            if p != q
                push!(C, p ∪ q)
            end
        end
        dict = scan(data, C, ε)
        merge!(+, support, dict)
        push!(D, dict)
        push!(L, [keys(dict)...] |> Set)
        n += 1
    end
    D, rules(D, support, ς)
end

# data = readlines("files/groceries.csv") .|> s -> split(s, ",")
```

```
# apriori(data; ε = 0.03, ς = 0.2) .|>
# [
#    dicts -> dicts .|> dict -> dict |> keys .|> key -> println(key, " -> ", dict[key]),
#     dict -> dict |> keys .|> key -> println(key, " -> ", dict[key])
# ]
apriori([[:A, :C, :D], [:B, :C, :E], [:A, :B, :C, :E], [:B, :E]]; ε = 0.5) .|>
[
    dict -> dict .|> dict -> dict |> keys .|> key -> println(key, " -> ", dict[key]),
    dict -> dict |> keys .|> key -> println(key, " -> ", dict[key])
]
```