データマイニング

第3回授業

相関分析

担当：菊池

相関分析とはどんな分析手法か？

データベースに潜むルール・パターンを見出す手法.

＜わかり易い例＞

スーパーマーケット，コンビニエンスストア等で1人1人の客が1回に行った買い物の内容を記録しておき，長期間に渡り記録された大量のデータを分析する
→ 1回の買い物でどのような商品を同時に買う傾向にあるのかを掴む.

例えば，分析の結果,

「商品Aと商品Bを購入した顧客は，同時に商品Cも購入する傾向にある」

ということがわかれば,

- 商品A，BやCをどのように陳列すべきか
- （これらの3品セットをより多く買ってもらう為に近づけて陳列する / 店内を長く歩き回ってもらうために3品を離れた場所に陳列する）
- セット商品をどのような組み合わせとして設定すべきか
- 商品A，Bの特売を行う際には商品Cの在庫をどれぐらいにすれば良いか

などの売り上げ向上に繋がる戦略上のヒントが得られる.

他の例

客が同時に複数の商品・サービスを購買したり，複数の行動を連続してとったりするような場合の分析に有効.

- 電話のオプションの申し込み（キャッチホン，転送電話，短縮ダイヤル，ナンバーディスプレイ等）は，どういうサービスをセットにすることで収益を増やせるかについての知見を提供する.
- 病院の治療履歴より，同じ患者が受けた特定の治療の組み合わせについて分析することで，合併症についての知見が得られる.

  （糖尿病を患った患者は網膜症を発症しやすい，など）

定式化

前述の例の

商品Aと商品Bを購入した顧客は同時に商品Cも購入する傾向にある

という事実は,

　　{商品A，商品B}　⇒　{商品C}

のように定式化できる.

ここで一般化し，商品1つ１つを**アイテム**とみなす.

これらアイテムの集合が**アイテム集合**であり,

　　$I = \{ i_1, i_2, \cdots, i_m \}$

のように表す.

アイテム集合はお店全体の品揃えに対応する.

各要素（アイテム）がお店の陳列商品に対応する.

相関ルール

一般に，X，Yをアイテムの集合として（例えばX,Yはそれぞれ商品群で，X＝｛商品A，商品B｝，Y＝｛商品C｝のように対応）

$$X \Rightarrow Y \qquad (1)$$

と記述される事実を**相関ルール**（association rule）という．

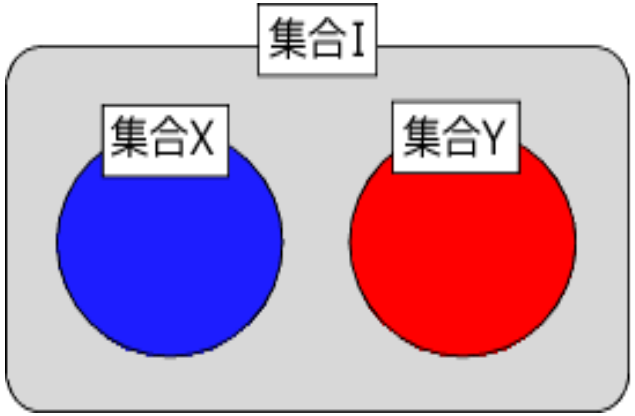
(1)の左辺（矢印の左側）を**前提部**，右辺を**帰結部**という．

なお，(1)において

$X \cap Y = \phi$ (空集合)　　※空集合：中身のない空っぽの集合

を満たしているとする．

すなわち，集合Xと集合Yとで共通する要素が無いものとする．



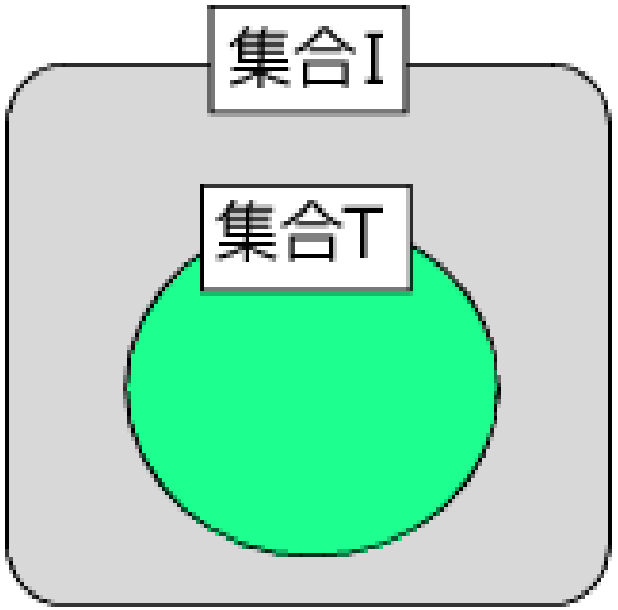
トランザクション

1回の取引（買い物）のことを**トランザクション**（第2回授業で既出）と呼び，ここでは**T**と表す．

トランザクションＴはアイテム集合の中のいくつかの要素から構成されるので，Ｔはアイテムの集合と言え，

$$T \subseteq I$$

のように，アイテム集合Iの部分集合として表せる（下図参照）．



データベースＤ：トランザクションの集合
→「全ての顧客の買い物の記録」に相当

データベースD	
ＴＩＤ	アイテム
0001	A, C, D
0002	B, C, E
0003	A, B, C, E
0004	B, E

※TID･･･トランザクションのID

確信度(confidence)

データベースＤの中の，Ｘを含むトランザクションのうち，Ｙを含むものの割合がc%であるとき（すなわち**少なくともＸが含まれる**トランザクションのうち，**ＸとＹの両方が含まれる**ものの割合がc%であるとき）

「相関ルール Ｘ ⇒ Ｙ はＤにおいてc%の**確信度**で成立している」

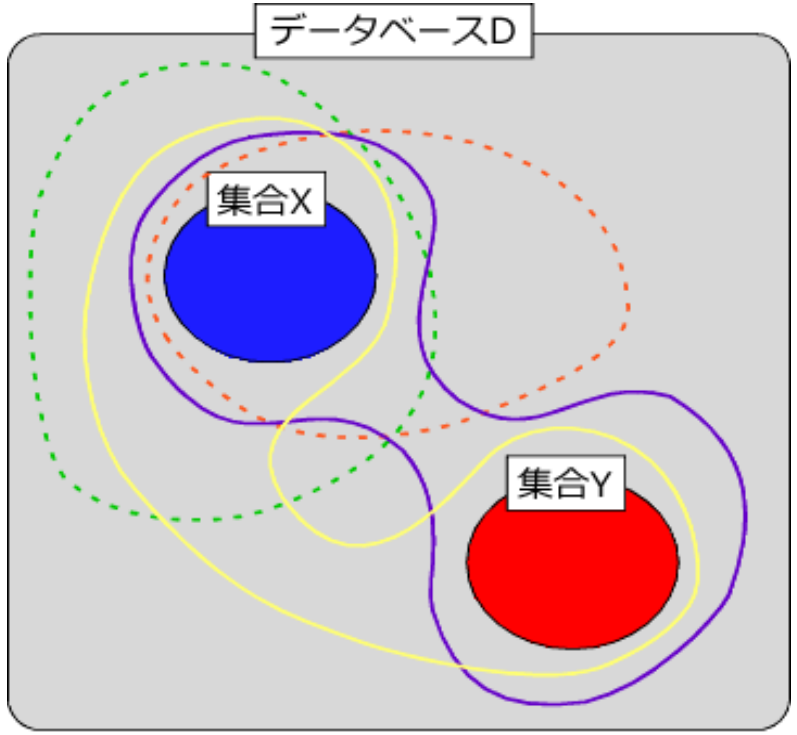
といい，

$$conf(X \Rightarrow Y) = c\%$$

と表現する．

◎ 確信度が小さいとルールとして一般化することが難しくなる．

重要なルールの確信度は高いものとなる．



確信度の具体的な計算例

表のデータベースでX={B, E}，Y={C}としたときのconf(X ⇒ Y)は？

Xを含むトランザクションはTIDで0002, 0003, 0004の3つ.

それらのうちYも含む（すなわち，XとYの両方を含む）のはTIDで0002と0003の2つ.

※TID 0001にもYは含まれるが，Xが含まれないのでここでの議論の対象外.

よって，Xを含む3つのトランザクション中，2つのトランザクションにYが含まれるので，

conf(X ⇒ Y) = 66.7%

となる.

データベースD	
TID	アイテム
0001	A, C, D
0002	B, C, E
0003	A, B, C, E
0004	B, E

サポート(support)

データベースDのうち，X ∪ Y（すなわちXとYの両方）を含むトランザクションの，全トランザクション（XやYの少なくとも一方を含まない（= 一方のみが含まれる，あるいは両方とも含まれない）トランザクションも考慮に入れる）に対する割合がs%であるとき，「相関ルール X ⇒ Y はDにおいてs%の**サポート**(support，支持度)で成立している」

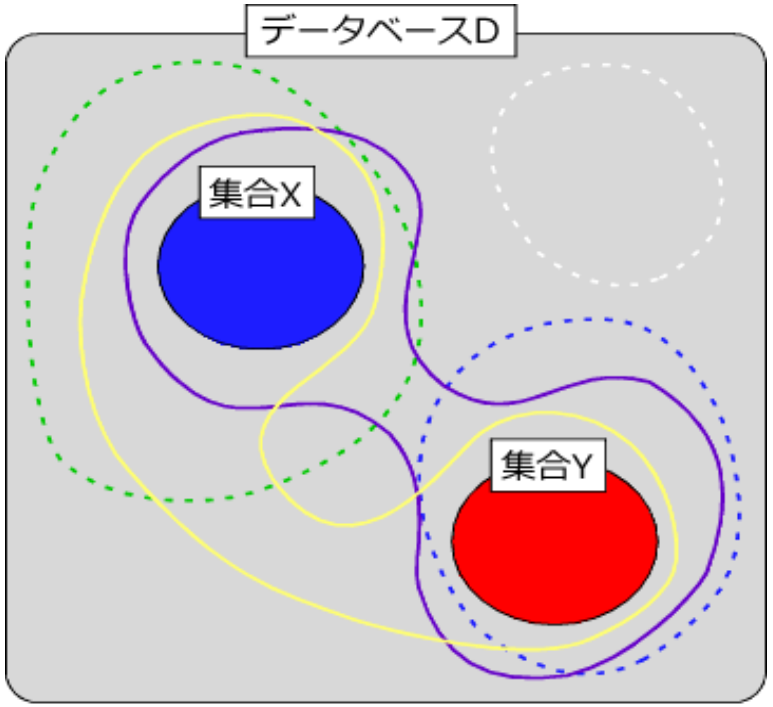
といい，

support(X ⇒ Y) ＝ s%

と表現する．

◎ 無数に作り出せる相関ルールのうち，サポートが有る程度大きいものが重要．

↑ サポートが小さいと，僅かなデータのみにしか当てはまらない「出番の少ない」ルールとなってしまうから



サポートの具体的な計算例

表のデータベースで，X={B, E}，Y={C}としたとき
support(X ⇒ Y)は？

全トランザクションはTIDで0001, 0002, 0003, 0004の4つ.
それらのうちXとYの両方を含むトランザクションはTIDで0002と0003の2つ.
よって，合計4つのトランザクション中，
XとY が同時に含まれるのは2つのトランザクションなので,

support(X ⇒ Y) = 50%

となる.

データベースD	
TID	アイテム
0001	A, C, D
0002	B, C, E
0003	A, B, C, E
0004	B, E

相関ルールの抽出

確信度とサポートがある程度大きい相関ルールに価値がある.

もしも価値の高い相関ルールをつくるアイテム集合が予めわかっているのであれば，下記のような簡単な処理で相関ルールの確信度とサポートを知ることができる.

相関ルールX ⇒ Yにおいて，下記のa, bをそれぞれ数え上げる.
a: 前提部Xを含むトランザクションの個数
b: 前提部と結論部が同時に（すなわちX ∪ Yが）出現するトランザクションの個数
これらa, bを用いて
確信度： b / a
サポート： b / N（N：トランザクション総数）
のように求まる.

しかし，どのアイテムの組み合わせによって確信度とサポートが大きい（＝価値のある）相関ルールができるかは事前にはわからない.

なぜなら，そのような相関ルールを求めること自体がここでのデータマイニングの目的だからである.

可能な全ての相関ルールを調べて，その中から重要なものを選ぶという方法も考えられなくも無いが，全ての相関ルールは膨大な数になるので実際的ではない.

そこで，価値のある相関ルールを自動的に効率よく見出す手法が求められる.

以下がその方法の代表的なものの１つ.

アプリオリアルゴリズム

Agrawalらは，1993年に次のような相関ルール抽出問題を提案した.

【相関ルール抽出問題】
・確信度の閾値（最小確信度）
・サポートの閾値（最小サポート）
の２つを入力したとき，それぞれの閾値以上の確信度・サポートを有する相関ルールを全て発見せよ.

翌年には効率よく相関ルールを抽出する**アプリオリアルゴリズム**と呼ばれる方法を提案した.

サポートと確信度の表現

データベースDの全トランザクションのうち，アイテム集合Xを含むトランザクションの割合を**Xのサポート**と呼び，**support(X)**と書く．

また，ユーザーが与えた最小サポート以上のサポートを持つアイテム集合を**頻出アイテム集合**と呼ぶ．

相関ルール「X ⇒ Y」のサポートsupport(X ⇒ Y)は

support(X ⇒ Y) ＝ support(X ∪ Y)

で与えられる．

また，相関ルール「X ⇒ Y」の確信度conf(X ⇒ Y)は

conf(X ⇒ Y) ＝ support(X ∪ Y) / support(X)

で与えられる．

したがって，条件を満たす相関ルールを導出するには，

support(X ∪ Y) ≧ 最小サポート

support(X ∪ Y) / support(X) ≧ 最小確信度

の2つを同時に満たすアイテム集合X∪Y，及びXを求め，そのサポートを求めればよい．

効率的な計算法は？

アイテム集合のサポートの逆単調性(1)

アイテム集合のサポートが**逆単調**という性質に着目する.
逆単調とは，アイテム集合X, Yが **X** ⊂ **Y** であれば,

support(X) ≧ **support(Y)**

となる性質のことである.

【具体例】例えば，X={パン，牛乳}，Y= {パン，牛乳，チョコレート} とすると， X ⊂ Yとなる.
そして，全ての客の買い物のうち，少なくともパンと牛乳を含むものの割合=support(X)よりも，
全買い物中，少なくともパンと牛乳とチョコレートを含むものの割合=support(Y)のほうが
小さくなる.

上記の性質より，以下が導かれる.

(i) {A, B, C}のサポートは必ず{A, B}のサポートより小さくなる.
したがって {A, B} が頻出アイテム集合ではないなら，それを含む{A, B, C}も
頻出アイテム集合ではないと言える.
(ii) {A, B, C} が頻出アイテム集合なら，その部分集合である{A, B}も必ず頻出アイテム集合
となる.

アイテム集合のサポートの逆単調性(2)

相関ルールX ⇒ Yが最小サポート以上であるとき，その相関ルールの確信度を求めるには，上で述べたようにsupport(X ∪ Y)とsupport(X)の両方が必要であるが，それらはいずれも最小サポートより大きい.

＜理由＞ 前提より，support(X ∪ Y) ≧ 最小サポート

また，X ⊂ X ∪ Y なので，逆単調の性質より，

support(X) ≧ support(X ∪ Y)

よって，support(X) ≧ support(X ∪ Y) ≧ 最小サポート

したがって，最小サポート以上のサポートを有する全ての相関ルールの確信度を求めるためには，すべての頻出アイテム集合（最小サポート以上のサポートを持つアイテム集合）についてサポートを求めておけばよい.

相関ルール抽出問題とアプリオリアルゴリズム

条件を満たす相関ルールの抽出問題･･･下記の２つの部分問題に帰着される.

(1)　頻出アイテム集合を全て見出し，サポートを求める
　　→ 2次記憶上にあるデータベースの繰り返しスキャンが必要 → 負荷：大

(2)　(1)で求めた頻出アイテム集合を使って最小確信度以上の相関ルールを求める → 負荷：小

処理の負荷の大きい上記の(1)のプロセスの効率化が重要となる.

アプリオリアルゴリズム･･･相関ルールの生成に必要な中間データ構造を主記憶内に作ることで効率化.

＜**特徴**＞

・1回のデータベーススキャンで，幾つかのアイテム集合のサポートをまとめて計算（k回目のスキャンで要素数kの頻出アイテム集合のサポートを求める）.

・要素数の少ないアイテム集合からサポートを調べる.
　最低サポートより小さいアイテム集合を1部分として包むアイテム集合は棄却.

アプリオリアルゴリズム(1)

Algorithm Apriori (DataSet D, $minsup$){　// カッコ内は引数

1) $C_1 := \{ \{i\} \mid i \in I \}$;　// 「要素数1のアイテム集合の候補」の集合をC_1とする.

2) $k := 1$;　// k： アイテム集合の要素数. 初期値を1とする.

3) while ($C_k \neq \phi$){　// C_kが空集合になるまで {･･･} 内の実行を続ける.

4) forall transaction $t \in D$ {

// D(トランザクションの集合)に含まれる各トランザクションtについて {･･･} 内を実行

// → C_k 中の各候補のサポートを調べる.

5) $C := \text{Subset}(C_k, t)$;

// C_k中のアイテム集合のうち t に含まれるもののみ関数Subset（というものがあるとする）により抽出

// その結果の集合をCとする → tに出現するアイテム集合候補のピックアップ

6) forall $c \in C$　// 集合Cに含まれる各アイテム集合(c)につき下記を実行

7) c.count++;　// cの出現度数（カウンタ）を1つ増やす.

8) }

アプリオリアルゴリズム(2)

9) $L_k := \{ c \in C_k \mid c.\text{count} \geqq minsup \}$;

// C_k中のアイテム集合のうちminsup以上の度数のもののみ抽出→それら集合をL_kとする

// → C_kの中の最小サポートを満足する部分をL_kとする.

10) $C_{k+1} := \text{AprioriGen}(L_k)$;

// AprioriGen関数でL_kより要素数k+1個のアイテム集合の候補群を生成→C_{k+1}とする.

11) $k := k + 1$; // 扱うアイテム集合の要素数を1増やす.

12) }

13) return $\cup_k L_k$;

// 得られた全てのL_kの和集合（L_1, L_2, L_3中のアイテム集合群からなる集合）を

// 結果として返す.

14) }

補足説明

・C_k は要素数kの頻出アイテム集合**の候補の集合**.
・L_k は要素数kの頻出アイテム集合**の集合**.
・*minsup*は最小サポートを満たすために必要なトランザクション数（最小サポート×トランザクション総数 $|D|$）
・トランザクション，候補アイテム集合，頻出アイテム集合中のアイテムは辞書順にソートされていると仮定.
・forall ～ {･･･}は，すべての ～ について{･･･}を実行する，という繰り返し処理.
・Subset(C_k, t)関数は，集合C_kに含まれている集合のうち，トランザクションtに含まれるものを抽出し，それらを集合として返す関数.
→ Subset(C_k, t)は「集合」の集合であり，集合C_kの部分集合となる.
・c.countは，集合cのカウント（角集合にカウンタが備わっていると考える）．初期状態では0と考える.
・AprioriGen(L_k)関数は，要素数kの頻出アイテム集合の集合であるL_kより，要素数$k+1$の頻出アイテム集合の候補の集合を求める関数.

AprioriGen関数について

アプリオリアルゴリズム中で呼び出される，要素数kのアイテム集合からなる集合L_kから要素数k+1のアイテム集合候補の集合C_{k+1}を生成する過程であるAprioriGen関数は，次の2段階の処理からなる．

(1) 結合フェーズ：

L_kの中のアイテム集合のうち最後（k番目）の要素のみが異なる 2つのアイテム集合同士を結合して（和集合をとる），要素数　k+1の候補アイテム集合を作る．これをC_{k+1}の要素に加える．

(2) 枝刈りフェーズ：

結合フェーズで生成された要素数k+1の候補アイテム集合のうち，それらから生成される要素数kの部分集合が元のL_k中に含まれないようなものがある集合を選び，C_{k+1}から削除する．

AprioriGen関数のアルゴリズム

0) Function AprioriGen(Itemsets Lk) {
1) //----- 結合フェーズ -----
2) $C_{k+1} := \phi$; // C_{k+1}を空集合にしておく
3) foreach p, q $\in L_k$ such that $(p.item_1 = q.item_1) \wedge \cdots \wedge (p.item_{k-1} = q.item_{k-1}) \wedge (p.item_k < q.item_k)$ {
// ↑1番目からk-1番目までの要素が同一＆k番目(一番最後)の要素のみ異なる全ての集合ペアp, qに関して下記を実行
4) c := p ∪ q.itemk; // 集合cは，集合pに集合qのk番目の要素を加えたものとする.
5) $C_{k+1} := C_{k+1} \cup \{c\}$; // 候補集合の集合$C_{k+1}$に上記の集合cを加える.
6) }
7) //----- 枝刈りフェーズ -----
8) foreach p $\in C_{k+1}$; // C_{k+1}に含まれる全ての集合pについて以下を実行.
9) foreach k-subsets s of p {
// 要素数k+1の集合pから，要素数kの部分集合sを生成．それらの全てについて以下を実行.
10) if (s $\notin L_k$) pをC_{k+1}から削除する; // もしsがL_kに含まれていなければ，pをC_{k+1}から削除
11) }
12) }
13) return C_{k+1};
14) }

※ a.itembとは，要素であるアイテムが辞書順にソートされた集合aのうち，前からb番目のアイテムを差すものとする.

AprioriGen関数 実行例

＜AprioriGen関数 実行例１＞
L_2 ：｛{B, C}, {B, E}, {C, E}｝→ C_3の生成
結合フェーズ：{B, C} と {B, E} とから候補 {B, C, E} を生成.
枝刈りフェーズ： 候補 {B, C, E} の要素数2の部分集合 {B, C} も {B, E} も {C, E} もいずれもL_2に含まれる.
よって，C_3は｛{B, C, E}｝.

＜AprioriGen関数 実行例２＞
L_3 ：｛{A, B, C} ,{A, B, D}, {A, C, D}, {A, C, E}, {B, C, D}｝ → C_4の生成
結合フェーズ： {A, B, C} と {A, B, D} とから候補 {A, B, C, D} を生成.
また {A, C, D} と {A, C, E} とから候補 {A, C, D, E} を生成
枝刈りフェーズ： 候補 {A, B, C, D} のうち要素数3の部分集合 {A, B, C}, {A, B, D}, {A, C ,D}, {B, C, D} のいずれL_3に含まれるので，候補 {A, B, C, D} は残す.
候補 {A, C, D, E} のうち要素数3の部分集合 {A, C, D}, {A, C, E} のいずれもL_3に含まれるが，{A, D ,E}, {C, D, E} のいずれもL_3に含まれない． よって候補 {A, C, D, E} は削除する.
以上によりC_4は｛{A, B, C, D}｝.

＜AprioriGen関数 実行例３＞
L_1：｛{A}, {B}, {C}｝→ C_2の生成
結合フェーズ：{A}, {B}, {C} から候補 {A, B}, {A, C}, {B, C}を生成.
枝刈りフェーズ： 候補 {A, B} の部分集合 {A} も {B} もL_1に含まれる.
同様に，{A, C}の部分集合{A}, {C}や，{B, C}の部分集合{B}, {C}もL_1に含まれる.
よってC_2は {{A, B}, {A, C}, {B, C}}.

Aprioriアルゴリズム全体の実行例

以下は最小サポートを５０％（トランザクションで2回分・・・*minsup* ＝|D| ×0.5＝4×0.5＝2）に設定した場合．カウント1回のアイテム集合は$C_k \rightarrow L_k$導出の際に除去される．
また,アイテム集合中の各アイテムはアルファベット順にソーティングされているとする．

C_1

データベースD	
TID	アイテム
0001	A, C, D
0002	B, C, E
0003	A, B, C, E
0004	B, E

データベースＤをスキャン．カウントが*minsup*（2回）未満のアイテム集合は除去．
ここでは{D}が削除される．

L_1

アイテム集合	カウント
{A}	2
{B}	3
{C}	3
{E}	3

候補生成
（AprioriGen関数）

C_2

アイテム集合
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

Ｄをスキャン．カウントが*minsup*未満のアイテム集合を除去

L_2

アイテム集合	カウント
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

候補生成
（AprioriGen関数）

C_3

アイテム集合
{B, C, E}

Ｄをスキャン．カウントが*minsup*未満のアイテム集合を除去

L_3

アイテム集合	カウント
{B, C, E}	2

求める集合は，
$L_1 \cup L_2 \cup L_3$
＝ { {A}, {B}, {C}, {E}, {A, C}, {B, C}, {B, E}, {C, E}, {B, C, E} } となる．