

データサイエンス

第9回

～アソシエーション分析～

情報理工学系研究科
創造情報学専攻
中山 英樹

本日の内容

- レポート課題講評
 - 上位者による紹介など
- アソシエーション分析
 - 相関ルール
 - 頻出パターンマイニング

第一回レポート課題（回帰）

- Wine quality prediction dataset

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

- 物理化学的特徴からワインの質（10段階）を予測（回帰）

- 説明変数

- 0 - red or white
- 1 - fixed acidity
- 2 - volatile acidity
- 3 - citric acid
- 4 - residual sugar
- 5 - chlorides
- 6 - free sulfur dioxide
- 7 - total sulfur dioxide
- 8 - density
- 9 - pH
- 10 - sulphates
- 11 - alcohol

- 目的変数

- 12 - quality (score between 0 and 10)



結果

バリデーション (200サンプル)

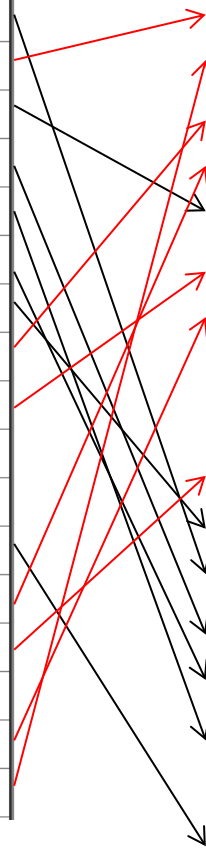
Score Board (on validation set)

Rank	Name	Mean ABS Error
1	Tomoya_Tsuda	0.355
2	KentaroNishi	0.37
3	Shimakaze	0.375
4	tkurokawa	0.38
4	DeepLearner	0.38
4	KenMiura	0.38
7	TakumaHatano	0.39
7	ngo	0.39
9	MutsukiKojima	0.395
9	6217_	0.395
9	nakachanOK	0.395
12	KeisukeMaeda	0.4
12	parameter_tuner	0.4
14	chika	0.405
14	flareon	0.405
14	MasayukiHirota	0.405
14	6217	0.405

最終結果 (800サンプル)

Score Board

Rank	Name	Mean ABS Error
1	KentaroNishi	0.50125
2	6217	0.50375
3	ngo	0.50625
4	parameter_tuner	0.50875
5	Shimakaze	0.51125
6	MutsukiKojima	0.5125
6	MasayukiHirota	0.5125
8	6217_	0.51375
9	Hiroki_Kojima	0.515
10	chika	0.51625
11	TakumaHatano	0.51875
12	Tomoya_Tsuda	0.52
13	tkurokawa	0.52125
14	KenMiura	0.53125
15	DeepLearner	0.5325
16	snagai	0.53375
17	KeisukeMaeda	0.54



講評

- 今回の課題で伝えたかったこと
 - オーバーフィッティングのこわさ
 - パラメータチューニングの重要性
 - クロスバリデーション、変数選択など基本的な道具
- 性能向上のポイント
 - リッジ回帰のパラメータ
 - 赤ワイン、白ワインのカテゴリ分け
 - 予測値の四捨五入
 - いつもよくなるとは限らないので注意！

その他、工夫してくれた例

- Lasso (L1正則化)
- Elastic Net (L1正則化+L2正則化)
- ウェーバー則 (感覚量は物理量の対数に比例)
- Support vector machine (SVM)
- Support vector regression (SVR)
- ランダムフォレスト
- カーネル回帰分析
- 順序プロビット・モデル
- Spike and slab model
- Deep Learning

参考：学習データを減らしてみる

(sample.py)

```
import numpy as np
import pandas as pd
import regression as reg
```

```
nsample = 30 #学習サンプル数を30
```

```
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")
```

(中略)

```
w = reg.ridgeRegres(XTrain,yTrain,0.0) #リッジ回帰
```

(略)

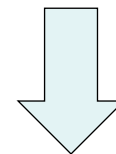
nsample = 300(デフォルト)

Training error

0.464144810379

Validation error

0.596275489056



nsample = 30

Training error

0.41561151019

Validation error

1.41699841724

参考：学習データを減らしてみる

- リッジ回帰のパラメータ λ の影響
 - $\lambda = 0$ (線形回帰)
Training error 0.41561151019
Validation error 1.41699841724
 - $\lambda = 10$
Training error 0.508888215592
Validation error 0.739827092093
 - $\lambda = 50$
Training error 0.544388436019
Validation error 0.555972813195
 - $\lambda = 100$
Training error 0.561393039126
Validation error 0.647175026268

次回の課題（予定）

- パターン認識（クラス分類）
- 問題の規模は今回より大きくします
- 前処理の工夫の余地も増やしたいと思います
- 年明け最初の講義か、遅くとも二週目までに発表

アソシエーション分析

- 大量のトランザクションデータ（販売、取引のデータなど）から興味ある規則や、頻出パターンを検出する
 - 商品推薦や、販売戦略の策定に利用
 - 教師なし学習の一つとも解釈できる

amazon.co.jp

クリック なか見! 便条



この商品を買った人はこんな商品も買っています



象徴的な伝説

- {オムツ} ⇒ {ビール}
 - 1992年頃、Oscor Drugsという小売ストア・チェーン（らしい）
 - 午後5時から7時の間、消費者がおむつとビールを買うということを発見した
 - オムツとビールの売り場を隣にしたら売上が上がった

※実際には、具体的にアクションに移すところまではいっていないらしい

応用例

- リコメンデーション
 - Amazon
 - Youtube
- 販売戦略など
 - セット商品の作成
 - 売り場の配置
 - 在庫管理

相関ルール

- トランザクションデータベースに頻出するアイテム間の組み合わせの規則
 - 連関ルール、関連ルールともよばれる
- 例) 商品(の集合)Xを買う人はYも買う可能性が高い

$$\{X\} \Rightarrow \{Y\}$$

XとYは互いに同じものを含まない
アイテムの集合

e.g., $\{\text{パン}, \text{牛乳}\} \Rightarrow \{\text{卵}\}$

相関ルール

- もう少し用語からちゃんと書くと...

アイテムの集合 $I = \{i_1, i_2, \dots, i_n\}$

バスケットデータ
(トランザクションの集合)

$$D = \{t_1, t_2, \dots, t_M\}$$

$t_j : I_j \in I$ で構成される
トランザクション

- 相関ルールの書き方： $\underbrace{\{X\}}_{\substack{\text{前提部} \\ \text{(antecedent)}}} \Rightarrow \underbrace{\{Y\}}_{\substack{\text{結論部} \\ \text{(consequent)}}} \quad X, Y \in I$
 $X \cap Y = \emptyset$

相関ルールの評価・抽出

- 「意味のあるルール」とは何か？
 - アイテムの組み合わせは膨大
 - 相関ルール抽出のための評価指標とアルゴリズムが重要
- 以下の指標を総合的に判断
 1. 支持度
 2. 確信度
 3. (リフト)

相関ルールの評価指標(1)

支持度 (support)

- 全トランザクション数に対する、条件を満たすトランザクションの割合
- $N(X)$: 条件 X ($X \in I$) を満たすトランザクション数
- M : 全トランザクション数

$$\text{sup}(X) = \frac{N(X)}{M} \quad \longleftarrow \text{Pr}(X)$$

ある条件 X がどれくらい起こりやすいか？

相関ルールの評価指標(2)

確信度 (confidence)

- 条件 X を満たすトランザクション数に対する、**条件 X と Y の両方を満たす**トランザクション数の割合
($X, Y \in I, X \cap Y = \emptyset$)

$$\text{conf}(X, Y) = \frac{N(X \cup Y)}{N(X)} = \frac{\text{sup}(X \cup Y)}{\text{sup}(X)} \quad \leftarrow \quad \frac{\text{Pr}(X, Y)}{\text{Pr}(X)} = \text{Pr}(Y | X)$$

ある条件 X が起こったもとで、条件 Y がどれくらい起こりやすいか？

相関ルール抽出の難しさ

- 「あるルールの良さ」は定義されたが、どうやってバスケットから価値のあるルールを見つけ出す？
- **直接的な解法 = 全探索**：作ることが可能な相関ルールを全て、一つ一つ検証する（支持度と確信度を検査）
- データベースが大きくなるとアイテムの組み合わせ数が爆発
 - アイテム数 n 個の時、作ることのできる組み合わせの数は $2^n - 1$
 - 例）商品が100種類の場合でも 1.26×10^{30} 通り！



支持度と確信度の性質を利用して効率よく探索

- 幅優先：アプリオリアルgorithm
- 深さ優先：FP-growth

アプリアリアルゴリズム

- 1. 頻出パターンの抽出

$$\text{sup}(X) \geq \text{minsup} \quad (\text{最小支持度})$$

を満たす全てのアイテムセット X を抽出する。この集合を F とする。

- 2. 見つかった集合 F の中から、

$$\text{conf}(X, Y) = \frac{\text{sup}(X \cup Y)}{\text{sup}(X)} \geq \text{minconf} \quad (\text{最小確信度})$$

を満たす全ての相関ルール $X \Rightarrow Y$ を抽出する。

R.Agrawal and R.Srikant, “Fast Algorithms for Mining Association Rules”,
In Proc. VLDB, 1994.

1. 頻出パターンの抽出

$$\text{sup}(X) \geq \text{minsup}$$

を満たす全てのパターン X を抽出する

これらの集合を F と表記する

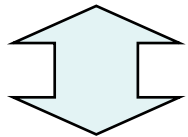
- 準備

- F のうち、ちょうど k 個のアイテムからなる頻出パターンの集合を F_k と表記する
- $F_1 \sim F_k$ が分かっているならば、 F_{k+1} は効率的に探索できる！

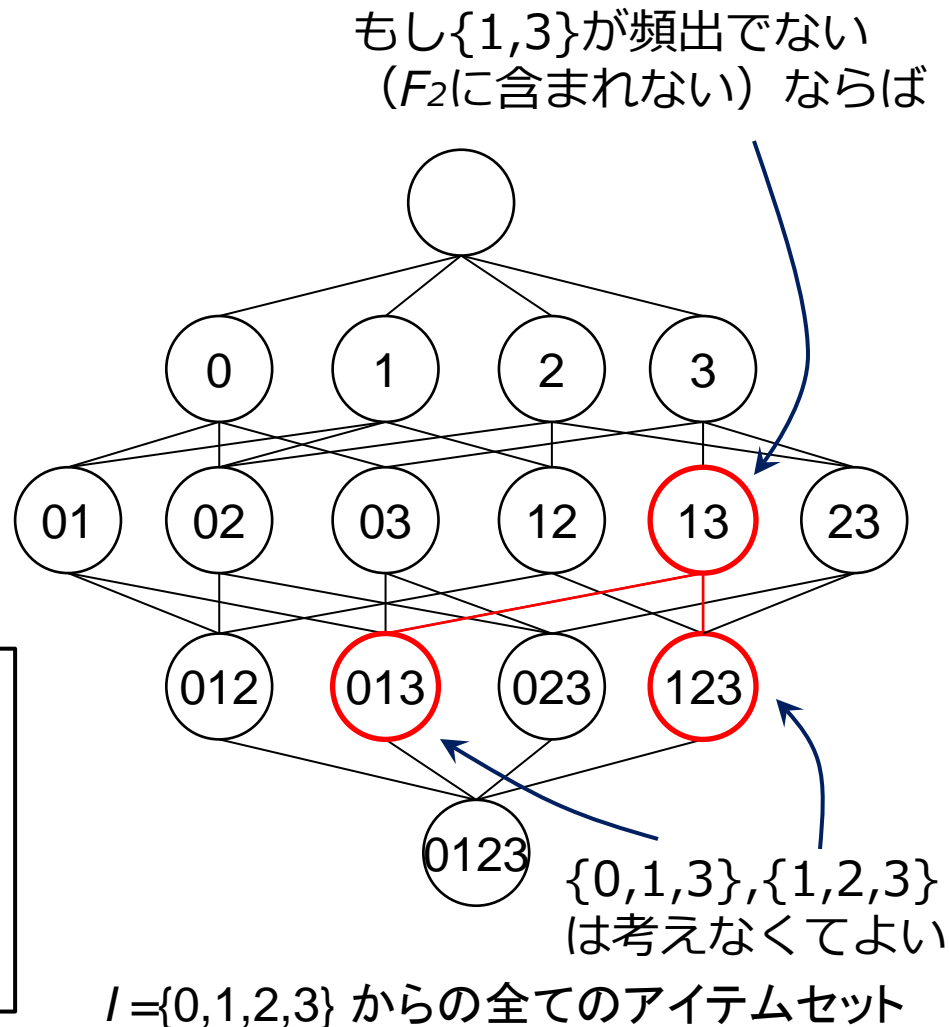
1. 頻出パターンの抽出

- Apriori principle

- あるアイテムセット X_{k+1} が頻出パターンであるなら、その部分集合は全て頻出パターンである



X_{k+1} から一つアイテムを除いてできるアイテム集合のいずれかが F_k の要素でないならば、 X_{k+1} は頻出ではない



頻出パターンの抽出アルゴリズム

- $k = 1$
- F_1 を生成（単純にデータベースを数え上げ）
- ループ先頭
 - F_k の要素からアイテムを一つ増やしたアイテムセットの集合を生成。これを C_{k+1} と表記する。
 - C_{k+1} 中の集合が実際に頻出かどうかを，データベースを数え上げて F_{k+1} を生成
 - F_{k+1} が空ならばループを終了
 - $k = k + 1$; ループ先頭に戻る
- 出力： $F_1, F_2 \cdots F_{k+1}$

頻出パターン抽出：コード(1/3)

```
def loadDataSet():    #ダミーデータ
    return [[1, 3, 4], [2, 3, 5], [1, 2, 3, 5], [2, 5]]

def createC1(dataSet):    #k=1の場合の候補集合（すなわち全アイテム一つ一つ）を生成
    C1 = []
    for transaction in dataSet:
        for item in transaction:
            if not [item] in C1:    #まだ入っていないアイテムをC1に追加
                C1.append([item])

    C1.sort()
    return map(frozenset, C1)#use frozen set so we can use it as a key in a dict
```

頻出パターン抽出：コード(2/3)

```
def scanD(D, Ck, minSupport):
    ssCnt = {}
    for tid in D:
        for can in Ck: #各候補について、バスケット中の出現数を数え上げ
            if can.issubset(tid):
                if not ssCnt.has_key(can): ssCnt[can]=1
                else: ssCnt[can] += 1
    numItems = float(len(D))
    retList = []
    supportData = {}
    for key in ssCnt:
        support = ssCnt[key]/numItems #各候補について支持度を計算
        if support >= minSupport: #最小支持度以上であれば結果に加える
            retList.insert(0,key)
        supportData[key] = support
    return retList, supportData
```

頻出パターン抽出：コード(3/3)

```
def aprioriGen(Lk, k): #creates Ck
    retList = []
    lenLk = len(Lk)
    for i in range(lenLk):
        for j in range(i+1, lenLk):
            L1 = list(Lk[i][:k-2]); L2 = list(Lk[j][:k-2])
            L1.sort(); L2.sort()
            if L1==L2: #if first k-2 elements are equal
                retList.append(Lk[i] | Lk[j]) #set union
    return retList
```

```
def apriori(dataSet, minSupport = 0.5): #最小支持度のデフォルト値は0.5
    C1 = createC1(dataSet) #k=1の場合の候補
    D = map(set, dataSet)
    L1, supportData = scanD(D, C1, minSupport) #最小支持度以上のアイテム
    L = [L1]
    k = 2
    while (len(L[k-2]) > 0): #最小支持度を超えるアイテムセットが見つからなくなったら終了
        Ck = aprioriGen(L[k-2], k)
        Lk, supK = scanD(D, Ck, minSupport) #scan DB to get Lk
        supportData.update(supK)
        L.append(Lk)
        k += 1
    return L, supportData
```

試してみる

```
>>> import apriori
>>> dataSet=apriori.loadDataSet()
>>> L,suppData=apriori.apriori(dataSet)
>>> L #見つかった頻出パターン
[[frozenset([1]), frozenset([3]), frozenset([2]), frozenset([5])], [frozenset([1, 3]),
frozenset([2, 5]), frozenset([2, 3]), frozenset([3, 5])], [frozenset([2, 3, 5])], []]
>>> L[0] #k=1の頻出パターン
[frozenset([1]), frozenset([3]), frozenset([2]), frozenset([5])]
>>> L[1] #k=2の頻出パターン
[frozenset([1, 3]), frozenset([2, 5]), frozenset([2, 3]), frozenset([3, 5])]
>>> L[2]
[frozenset([2, 3, 5])]
>>> L[3]
[]

>>> L,suppData=apriori.apriori(dataSet,minSupport=0.7)
>>> L
[[frozenset([3]), frozenset([2]), frozenset([5])], [frozenset([2, 5])], []]
```

2. 相関ルールの抽出

1. で見つかった頻出パターンの集合 F の中から、

$$\text{conf}(X, Y) = \frac{\text{sup}(X \cup Y)}{\text{sup}(X)} \geq \text{minconf}$$

を満たす全ての相関ルール $X \Rightarrow Y$ を抽出する。

- トリック

$X \cup Y = X' \cup Y'$ かつ $X \supset X'$ である二つの相関ルール

$X \Rightarrow Y$, $X' \Rightarrow Y'$ の確信度の関係は

$$\text{conf}(X, Y) \geq \text{conf}(X', Y')$$

つまり、同じアイテムセットから作るルールの場合、前提部から結論部にアイテムを移しても確信度が上がることはない

証明

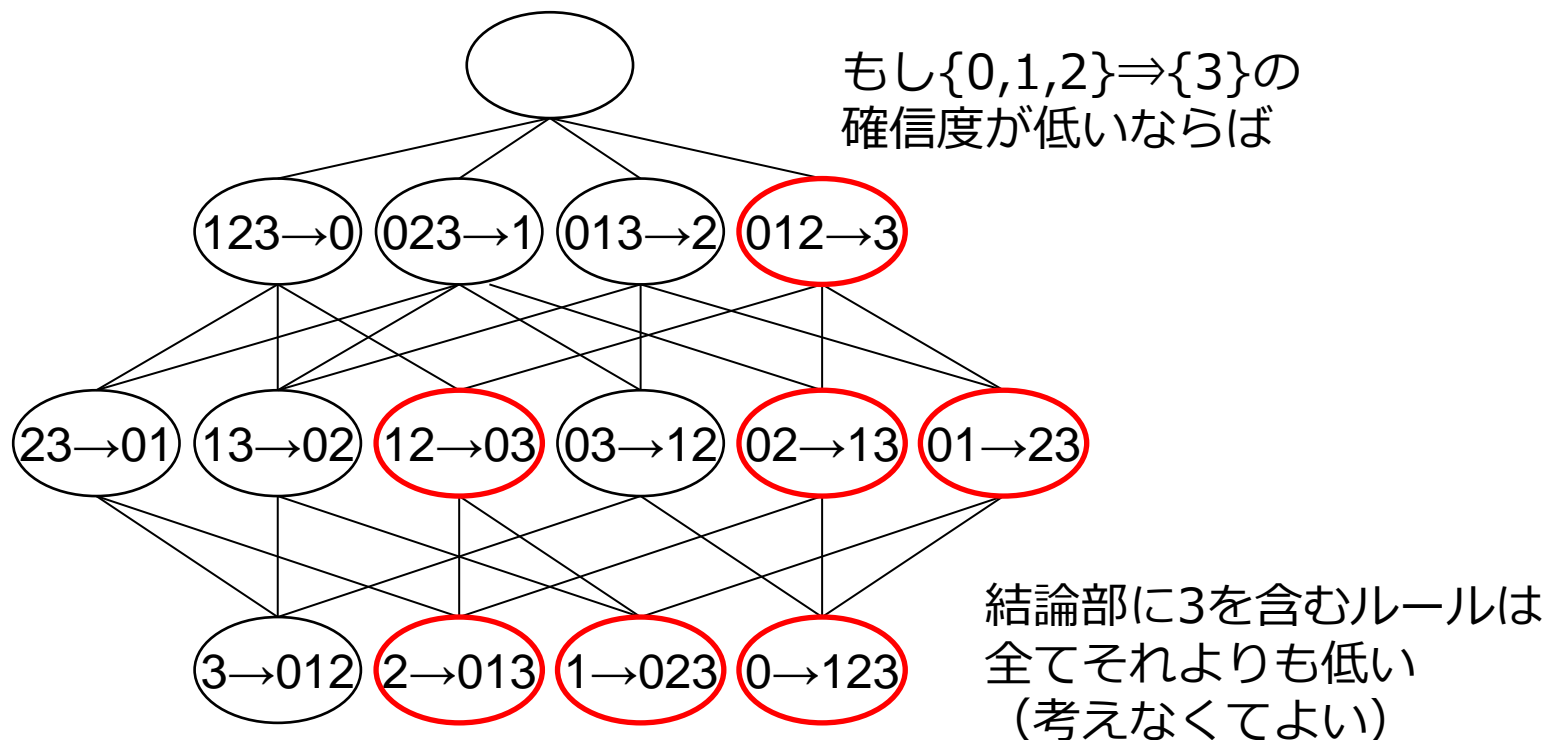
$$\text{conf}(X, Y) = \frac{\sup(X \cup Y)}{\sup(X)} \quad \text{conf}(X', Y') = \frac{\sup(X' \cup Y')}{\sup(X')}$$

$$X \cup Y = X' \cup Y' \quad \Rightarrow \quad \sup(X \cup Y) = \sup(X' \cup Y')$$

$$X \supset X' \quad \Rightarrow \quad \sup(X) \leq \sup(X')$$

$$\therefore \text{conf}(X, Y) \geq \text{conf}(X', Y')$$

2. 相関ルール抽出



$I = \{0,1,2,3\}$ からのアイテム4つによる
全ての相関ルール
(結論部にアイテムを追加していく)

相関ルールの抽出アルゴリズム

- F_2 中の頻出アイテム集合 $\{A, B\}$ については, $\{A\} \Rightarrow \{B\}$ と $\{B\} \Rightarrow \{A\}$ の相関ルールの確信度を検査
- $k \geq 3$ の F_k 中の各頻出アイテム集合について $\{k-1\text{個}\} \Rightarrow \{1\text{個}\}$ 形式の相関ルールの確信度を検査
- $j=1$: ループ開始
 - $\{k-\{j+1\}\text{個}\} \Rightarrow \{j+1\text{個}\}$ 形式の相関ルールの候補を, $\{k-j\text{個}\} \Rightarrow \{j\text{個}\}$ 形式のルールから求める
 - 実際に確信度が minconf 以上かどうかを検証
 - 新たなルールが見つからない場合は終了
 - $j = j + 1$ としてループを続行

```

def generateRules(L, supportData, minConf=0.7): #メイン。頻出パターン抽出の結果を受ける
    bigRuleList = []
    for i in range(1, len(L)): #アイテム一個のみのパターンは飛ばしてループ
        for freqSet in L[i]:
            H1 = [frozenset([item]) for item in freqSet]
            if (i > 1):
                rulesFromConseq(freqSet, H1, supportData, bigRuleList, minConf) #候補を作りながら検証
            else:
                calcConf(freqSet, H1, supportData, bigRuleList, minConf) #アイテム二つの集合はそのまま確信度を検証
    return bigRuleList

def calcConf(freqSet, H, supportData, brl, minConf=0.7):
    prunedH = [] #create new list to return
    for conseq in H:
        conf = supportData[freqSet]/supportData[freqSet-conseq] #calc confidence
        if conf >= minConf:
            print freqSet-conseq,'-->',conseq,'conf:',conf
            brl.append((freqSet-conseq, conseq, conf))
            prunedH.append(conseq)
    return prunedH

def rulesFromConseq(freqSet, H, supportData, brl, minConf=0.7):
    m = len(H[0])
    if (len(freqSet) > (m + 1)): #try further merging
        Hmp1 = aprioriGen(H, m+1) #create Hm+1 new candidates
        Hmp1 = calcConf(freqSet, Hmp1, supportData, brl, minConf)
        if (len(Hmp1) > 1): #need at least two sets to merge
            rulesFromConseq(freqSet, Hmp1, supportData, brl, minConf)

```

試してみる

```
>>> import apriori
>>> dataSet=apriori.loadDataSet()
>>> L,suppData=apriori.apriori(dataSet,minSupport=0.5) #まず頻出パターンを探す
>>> rules=apriori.generateRules(L,suppData,minConf=0.7) #相関ルール抽出
frozenset([1]) --> frozenset([3]) conf: 1.0
frozenset([5]) --> frozenset([2]) conf: 1.0
frozenset([2]) --> frozenset([5]) conf: 1.0
>>> rules
[(frozenset([1]), frozenset([3]), 1.0), (frozenset([5]), frozenset([2]), 1.0),
 (frozenset([2]), frozenset([5]), 1.0)]
# {1}⇒{3}、{5}⇒{2}、{2}⇒{5} の三つのルールが抽出された

>>> rules=apriori.generateRules(L,suppData,minConf=0.5)
(省略)
```

例) 毒キノコの共通特徴

- UCI mushrooms dataset

<http://archive.ics.uci.edu/ml/datasets/Mushroom>

- 様々なキノコの23種の特徴を離散値で記録
(ここでは整数値に変換してある)

1: 食用

2: 毒

1	3	9	13	23	25	34	36	38	40	52	54	59	63	67	76	85	86	90	93	98	107	113
2	3	9	14	23	26	34	36	39	40	52	55	59	63	67	76	85	86	90	93	99	108	114
2	4	9	15	23	27	34	36	39	41	52	55	59	63	67	76	85	86	90	93	99	108	115
1	3	10	15	23	25	34	36	38	41	52	54	59	63	67	76	85	86	90	93	98	107	113
2	3	9	16	24	28	34	37	39	40	53	54	59	63	67	76	85	86	90	94	99	109	114
2	3	10	14	23	26	34	36	39	41	52	55	59	63	67	76	85	86	90	93	98	108	114
2	4	9	15	23	26	34	36	39	42	52	55	59	63	67	76	85	86	90	93	98	108	115

例) 毒キノコの共通特徴

- 特徴"2"と頻繁に共起する特徴を探してみる

```
>>> import apriori
>>> mushDataSet = [line.split() for line in open('mushroom.dat').readlines()]
>>> L,suppData=apriori.apriori(mushDataSet,minSupport=0.3)
```

```
>>> for item in L[1]:
...     if item.intersection('2'): print item
...
frozenset(['2', '59'])
frozenset(['39', '2'])
frozenset(['2', '67'])
frozenset(['2', '34'])
```

```
>>> for item in L[3]:
...     if item.intersection('2'): print item
...
frozenset(['63', '59', '2', '93'])
frozenset(['39', '2', '53', '34'])
frozenset(['2', '59', '23', '85'])
frozenset(['2', '59', '90', '85'])
```

FP-growth

- アプリオリアルゴリズムの欠点を克服
 - 候補集合を生成しない
- FP-treeと呼ばれるツリー構造を用い、深さ優先型の探索で頻出アイテム集合を列挙する
- 次回、時間があったら

J.Han, J.Pei, and Y.Yin, "Mining Frequent Patterns without Candidate Generation", In Proc. SIGMOD, 2000.

まとめ

- アソシエーション分析
 - 頻出アイテム集合：データベース中に高頻度に存在するアイテム集合を列挙
 - 相関ルール： X が起きるときには Y も起きやすい
- 頻出パターンマイニング
 - Apriori
 - 基本的で実装が容易だが、余分な候補を多く生成する。
 - FP-growth
 - 余分な候補の数を抑制。通常Aprioriより速いがメモリを使う。