Trabalho 01 de Data Mining
BGFriends

Gabriel Vinícius Heisler[1], **Renan Cecchin**[1]

[1]Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brazil

{gvheisler, rdcecchin}@inf.ufsm.br

1. Etapas

Neste trabalho foram usadas as etapas do KDD definidas por [Fayyad et al. 1996]

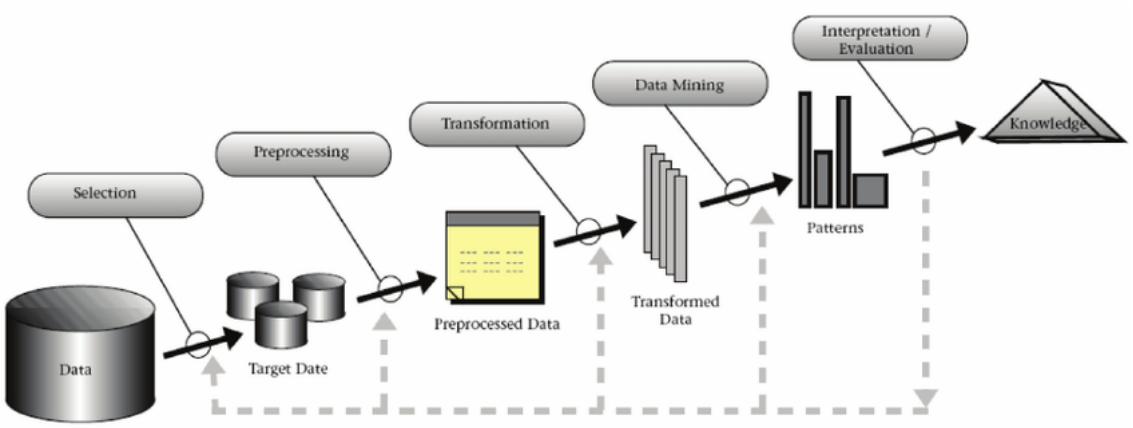


Figure 1. Figura retirada de [Fayyad et al. 1996]

1.1. Extração/Seleção

Os datasets[1] usados no trabalho foram disponibilizados no site da disciplina em formato CSV. Como eram dois arquivos, foi realizada a junção em um dataframe por meio do comando rbind(), que "cola" dois dataframes por linhas. [2]

```
df <- rbind(read.csv("~//_ASSOC_BGFriends_01.csv"),
            read.csv("~//_ASSOC_BGFriends_02.csv"))
```

1.2. Pré-Processamento

Como mencionado por [Assunção 2021], esta é a parte mais trabalhosa do processo de KDD. Por "sorte", nossos datasets já estão organizados em tabelas e sem dados faltantes, porém ainda temos alguns problemas. Primeiramente, vejamos as colunas:
Originalmente, os datasets continham 4 colunas:

- Partida: Cada linha contém um número diferente, que atua como identificador da partida
- Jogadore(a)s: Cada linha contém uma string, do tipo "character", com o nome dos jogadores (do time dos amigos) na partida
- Oponentes: A quantidade de pontos que os opontentes fizeram na partida
- Amigos: A quantidade de pontos que os amigos fizeram na partida

[1]Datasets da disciplina
[2]Código completo

1.2.1. Partida

A coluna partida foi retirada, pois não serve para nós, pois cada linha tem um número então podemos identificar a linha pelo número dela e não pela coluna Partida

1.2.2. Jogadore(a)s

Como em cada linha temos uma string, algumas contendo dois e outras três nomes, precisamos separar. Primeiramente, pegamos apenas a coluna de nomes, para facilitar.

```
nomes <- df[,1]
```

Agora, 'nomes' contém todas as strings de nomes do dataset original. Também criamos um novo dataframe, para que os nomes (já alterados) sejam salvos.

```
dfNomes <- data.frame()
```

Analisando, vemos que os nomes nesta string sempre são separados por vírgula. Sendo assim, podemos primeiramente passar por todas as linhas e separar essa string. Para fazermos isto, podemos utilizar a função do R chamada "strsplit()". Esta função, como visto na documentação da mesma, pode ser usada para separar uma string. Como argumentos, usamos a string que queremos separar (no nosso caso a linha i na coluna "Jogadores") e o separador em questão (no nosso caso, o separador é o caractere ','). Esta função nos devolve uma lista, por isto utilizamos a função "unlist()". Assim, temos um vetor do tipo "character".

```
nm <- unlist(strsplit(nomes[i], ','))
```

Como a linguagem R é case sensitive, quando comparamos strings com as mesmas letras porém diferenças nas maiúsculas e minúsculas, recebemos que a string é diferente. Para evitar problemas no futuro, transformamos todas as strings em letras minúsculas. Para isso fazer isto, podemos utilizar a função "tolower()"em todas as strings.

```
nm <- tolower(nm)
```

Outro problema que temos é a diferença nas strings. Por exemplo, nomes iguais foram salvos de maneiras diferentes (e.g., temos Francois e François). Para resolver este problema, podemos substituir as letras conflitantes por outras: padronizar as strings. Neste caso, substituimos:

- Todos os ' ' (espaços em branco) por '' (vazio), para tirar os espaços desnecessários;
- As letras 'ç' por 'c';
- Todos os caracteres 'e7' por 'c' (Caracter que no momento do load do dataset era 'ç' porém bugou);

Para realizarmos estas substituições, podemos utilizar a função "gsub()". Esta função recebe a substring a ser substituída, a substring substituinte e o vetor de strings no qual as substituições irão ocorrer (no nosso caso 'nm').

```
nm <- gsub(' ', '', nm)
```

```
nm <- gsub('c', 'c', nm)
nm <- gsub('<e7>', 'c', nm)
```

Após estas alteraçõs serem feitas, podemos salvar no nosso novo dataframe a nova linha (que agora contém os nomes separadamente). Um dos problemas que encontramos neste momento é que algumas strings terão 2 elementos e outras terão 3, então precisamos tratar isto. Caso a string da linha atual tenha 3 nomes, simplesmente adicionamos ela 'abaixo' do novo dataframe, com o rbind. Caso ela tenha 2 nomes, adicionamos ao fim dela mais um membro (para que ela tenha 3 elementos), o qual é NA. Então, adicionamos ela 'abaixo' do novo dataframe, com o rbind.

```
if(length(nm)==3){
    dfNomes <- rbind(dfNomes, nm)
}else if(length(nm)==2){
    nm[[3]]=NA
    dfNomes <- rbind(dfNomes, nm)
}
```

1.2.3. Oponentes/Amigos

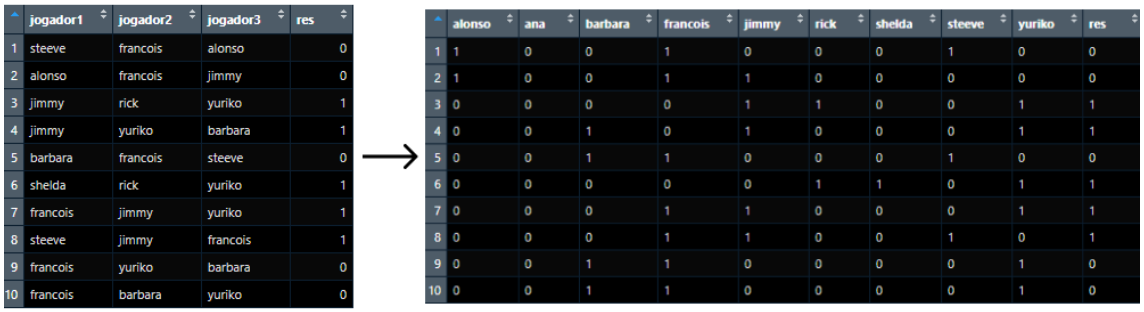
Como não levaremos em consideração as pontuações das partidas, transformamos estas duas colunas em apenas uma coluna, chamada 'res'. Esta nova coluna recebe um valor binário que significa vitória ou derrota para os amigos. Caso o valor da coluna seja 0, os amigos sofreram uma derrota naquela partida. Se for 1, os amigos ganharam. Analisando o dataset vemos que existe uma partida em que ocorre o empate. Neste caso, não consideramos a partida.

1.3. Transformação

[Han et al. 2022] define que nesta etapa de pré-processamento, os dados são transformados ou consolidados para que o processo de mineração resultante possa ser mais eficiente, e os padrões encontrados possam ser mais fáceis de entender. Aqui, precisamos transformar os nossos dados para que possamos aplicar os algoritmos de mineração de dados. Como estes dados são úteis para trabalharmos com algoritmos, devemos "moldar" (ou melhor, transformar) eles para servirem para os nossos algoritmos (Neste caso, o algoritmo "apriori")

1.3.1. One-hot-encoding

O algoritmo Apriori trabalha principalmente com tabelas no formato de "lista de presenças", ou "One-hot-encoding": A tabela tem colunas com todos os "itens" (neste caso todos os jogadores), e, comumente, o resultado (neste caso a coluna de vitória/derrota). Cada linha é uma partida, e se o jogador (que dá o nome a coluna) está presente na partida, a coluna é igual a 1 (caso contrário, 0). Todos estes itens devem ser 0 ou 1, ou seja, os levels de factor devem ser apenas 0 e 1. Este é o formato que o algoritmo usará para gerar as regras de associação. A figura 2 mostra a parte do dataframe antes e depois do One-hot-encoding.



	jogador1	jogador2	jogador3	res
1	steeve	francois	alonso	0
2	alonso	francois	jimmy	0
3	jimmy	rick	yuriko	1
4	jimmy	yuriko	barbara	1
5	barbara	francois	steeve	0
6	shelda	rick	yuriko	1
7	francois	jimmy	yuriko	1
8	steeve	jimmy	francois	1
9	francois	yuriko	barbara	0
10	francois	barbara	yuriko	0

	alonso	ana	barbara	francois	jimmy	rick	shelda	steeve	yuriko	res
1	1	0	0	1	0	0	0	1	0	0
2	1	0	0	1	1	0	0	0	0	0
3	0	0	0	0	1	1	0	0	1	1
4	0	0	1	0	1	0	0	0	1	1
5	0	0	1	1	0	0	0	1	0	0
6	0	0	0	0	0	1	1	0	1	1
7	0	0	0	1	1	0	0	0	1	1
8	0	0	0	1	1	0	0	1	0	1
9	0	0	1	1	0	0	0	0	1	0
10	0	0	1	1	0	0	0	0	1	0

Figure 2. Tabela antes e depois da transformação no formato One-hot-encoding

Para a criação do dataframe neste formato, primeiramente foram identificadas os nomes únicos e foi criada um novo dataframe com eles como nomes das colunas.

```
unicos <- sort(unique(unlist(dfNomes)))
ndf <- data.frame(matrix(ncol = length(unicos),
                         nrow = nrow(dfNomes)))
colnames(ndf) <- unicos
```

Após, este novo dataframe foi setado como 0 e, passando por todos os nomes únicos, foi checado se estava na partida em questão. Se sim, 1; Se não, 0.

```
for (i in 1:nrow(ndf)){
  for(j in 1:ncol(ndf)){
    ndf[i, j] <- 0
  }
}

for (indice in unicos) {
  for (i in 1:nrow(dfNomes)){
    for(j in 1:ncol(dfNomes)){
      if(!is.na(dfNomes[i,j])){
        if(dfNomes[i,j]==indice){
          ndf[i, indice] <- 1
        }
      }
    }
  }
}
```

Após, ndf foi colado, por colunas, com o dataframe de resultados, usando a função cbind().

Também, todos os itens foram transformados em factor (para o melhor funcionamento do algoritmo Apriori)

```
ndf <- cbind(ndf, res)
```

```
for (j in 1:ncol(ndf)) {
  ndf[,j] <- as.factor(ndf[,j])
}
```

1.4. Data-Mining

1.4.1. Apriori

Como visto em [Tan et al. 2016], o princípio Apriori sugere que

> "Se um conjunto de itens é frequente, então todos os seus subconjuntos também devem ser frequentes."

A maneira de funcionamento do Apriori é por meio de podas: Se um conjunto de itens for infrequente, todo os seus superconjuntos também devem ser infrequentes.

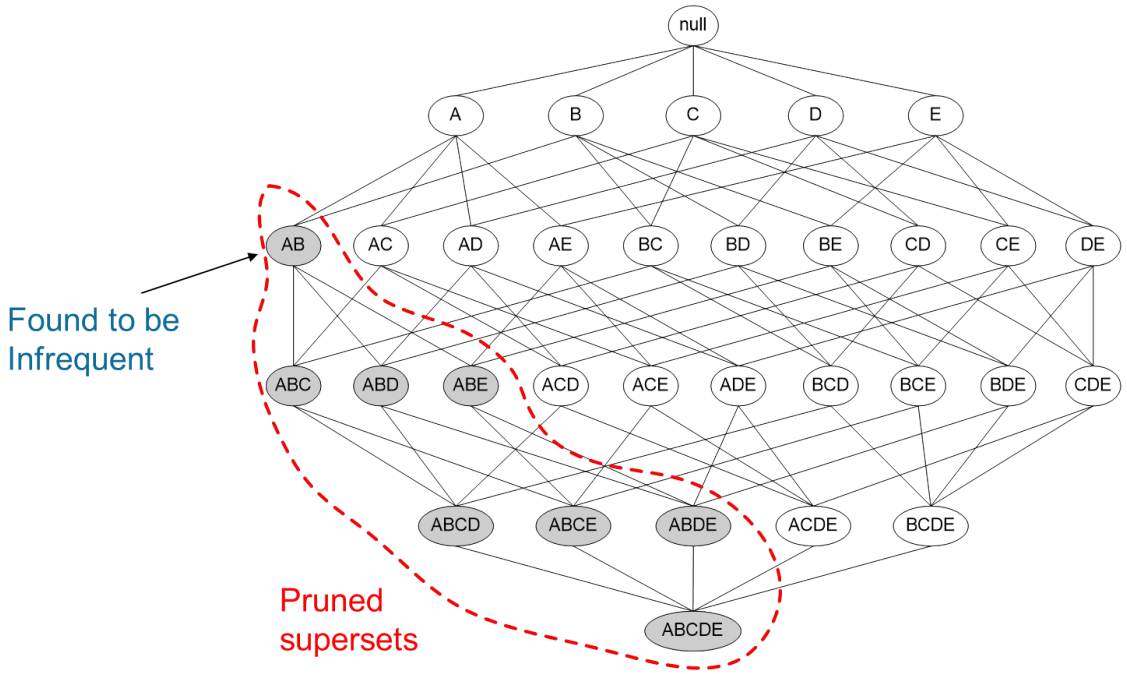


Figure 3. Ilustração de poda baseada em suporte, do livro de [Tan et al. 2016]

Neste trabalho, iremos usar a implementação do Apriori contida no pacote "Arules"

```
library('arules')
```

As regras de associação são criadas por meio da função "Apriori" Esta função recebe dataframe (em formato One-hot-encoding), os parâmetros das regras (tais como suporte, confiança, target, tamanho mínimo das regras, entre outros) e outros itens que não foram usados aqui. Abaixo, temos um exemplo de função de criação de regras, com suporte 0.1 e confiança 0.5). Também temos a função que ordena estas regras de maneira decrescente, baseado na confiança.

```
regras <- apriori(ndf, parameter = list(conf = 0.5, supp = 0.1,
                                        target = 'rules', minlen = 2, maxlen = 2))
regras <- sort(regras, by = 'confidence')
```

Também foi criado um subset de regras, onde apenas aparecem as regras em que o resultado é igual a 1 (para, por exemplo, ver qual o melhor jogador)

```
subc <- subset(regras, (rhs %in% 'res=1'))
subc <- sort(subc, by = 'confidence')
```

1.5. Visualização

Para visualizarmos as regras, utilizamos a função "Inspect()". Esta função recebe como argumentos as regras que serão visualizadas e mais alguns como modos de visualização, os quais não entraremos mais a fundo.

```
inspect(regras, ruleSep = '->', itemSep = '&')
inspect(subc, ruleSep = '->', itemSep = '&')
```

Esta função nos retorna todas as regras que cabem dentro dos parâmetros dados no uso da função apriori().

2. Resultados

Utilizando tudo que foi mencionado até aqui, analisando as regras de associação temos, baseado na confiança da regra, os seguintes resultados:

- Melhor jogador individual: yuriko
- Melhor dupla: jimmy e yuriko
- Pior dupla: barbara e francois

Na tabela a seguir[3] podemos ver as regras, seu suporte, confiança, coverage e lift, dados pelo algoritmo Apriori.

lhs	->	rhs	support	confidence	coverage	lift
{yuriko=1}	->	{res=1}	0.2500000	0.5937500	0.4210526	1.307971
{jimmy=1, yuriko=1}	->	{res=1}	0.1250000	0.7037037	0.1776316	1.550188
{barbara=1, francois=1}	->	{res=0}	0.1710526	0.7878788	0.2171053	1.4428624

References

[Assunção 2021] Assunção, J. V. C. (2021). *Uma breve introdução à Mineração de Dados. Bases para a ciência de dados, com exemplos em R.*, volume 1.

[Fayyad et al. 1996] Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–37.

[Han et al. 2022] Han, J., Pei, J., and Tong, H. (2022). *Data mining: concepts and techniques*. Morgan kaufmann.

[Tan et al. 2016] Tan, P.-N., Steinbach, M., and Kumar, V. (2016). *Introduction to data mining*. Pearson Education India.

[3] Resultados completos