1. Objetivo

- Familiarizar-se com um ambiente de apoio ao processo de descoberta de conhecimento;
- Executar ações necessárias durante o processo de descoberta de conhecimento usando um *dataset* simples;
- Praticar análise de dados com o uso de regras de associação.

2. *Dataset Titanic* e a Questão de Negócio

Para atingir os objetivos propostos, usaremos o *dataset Titanic*, que apresenta informações resumidas sobre os passageiros do navio Titanic. Cada linha dessa tabela representa uma pessoa.

Nossa pergunta de negócio será entender se existem fatores que determinaram a sobrevivência ou morte dos passageiros do Titanic. Vamos responder esta pergunta extraindo regras de associação.

Vamos executar um conjunto de comandos no ambiente *R* visando responder esta questão. A cada passo, você entenderá para que serve o comando. Ao longo deste laboratório, você deve responder a algumas perguntas, as quais devem ser submetidas em um arquivo PDF ao final desta atividade.

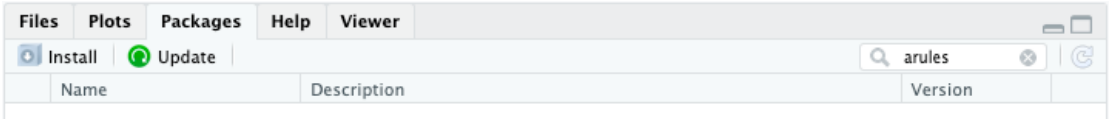
3. Preparando o ambiente de trabalho

3.1 Baixe os arquivos para a prática do laboratório

Crie um diretório na área de trabalho e salve nele os arquivos *titanic.raw.Rdata* e *titanic-script.R* disponíveis na tarefa do *Moodle*. O primeiro arquivo contém o conjunto de dados Titanic e o segundo, a sequência de comandos que será exercitada ao longo desta tarefa de mineração. Este diretório será considerado seu diretório de trabalho.

3.2 Baixe os pacotes necessário para a tarefa de mineração

Inicie o *RStudio*. Vamos utilizar o pacote arules para gerar regras de associação, o qual não está incluído na instalação *default* do RStudio. Verifique sua instalação: procure na janela no canto inferior da direta a aba *Packages*, digitando *arules*.



Se não estiver, clique no botão *install*. e instale o pacote arules.

3.2 Definir o *Script* e escolha o Diretório de Trabalho

- Vamos definir o diretório onde você colocou seus arquivos como diretório de trabalho. Acesse o menu *Session* → *Set Working Directory* → *Choose Directory*, e escolha este diretório.
- Abra o script, acessando o menu *File* → *Open File.* Selecione o arquivo *script-titanic.R* e clique em *open*. O *script* vai abrir no painel *Editor*.

  Este *script* contém um conjunto de comandos que você vai executar neste laboratório ao longo deste estudo dirigido. No restante deste documento, vamos executá-lo **passo a passo (UM COMANDO POR VEZ)**. ATENÇÃO: ao longo deste documento, vamos colocar em destaque os comandos que devem ser executados. Eles estão em sequencia no script: não precisa copiar eles a partir deste pdf.

  Para executar um comando de script, posicione o cursor na linha do comando no editor de código e clique no botão *Run*. Para executar o próximo comando, apenas clique novamente em *Run*. Se desejar re-executar algum comando, basta posicionar o cursor sobre ele, e clicar *Run*.

4. Exploração inicial do *dataset* Titanic

Execute o comando abaixo para carregar o conjunto de dados Titanic. **Ele supõe que este arquivo esteja em seu diretório de trabalho.**

```
load("titanic.raw.rdata")
```

Agora, vamos visualizar os dados existentes nesse conjunto. Execute o comando abaixo, o qual apresentará os dados em formato tabular em uma aba do painel *Editor*. Visualize o conteúdo da tabela.

```
View(titanic.raw)
```

Em todas as tarefas de mineração é necessário conhecer os seus dados, para saber se eles podem responder às suas perguntas de negócio, bem como para verificar sua sanidade. Além disto, frequentemente é necessário pré-processar os dados, tornando-os adequados para o uso de um determinado algoritmo de mineração. Vamos inspecionar a estrutura do conjunto de dados através do comando *str*. Execute a linha abaixo e observe o resultado.

```
str(titanic.raw)
```

Q1: Informe o número de registros do arquivo, o número de atributos, e o tipo de dado de cada atributo?

A compreensão dos dados é completada com uma avaliação dos dados e de sua qualidade. Muitas questões levantadas em relação ao objetivo de negócio podem ser respondidas através de uma simples análise estatística dos dados. Uma ferramenta que auxilia nessa etapa é o comando *summary*. Esse comando permite avaliar, para dados numéricos, questões como média, mediana, valores máximos e mínimos. Já para os atributos categóricos, permite analisar a freqüência de ocorrência. Na console, execute o comando *summary* e observe o resultado.

```
summary(titanic.raw)
```

Q2: Diga pelo menos dois fatos que o *summary* revelou sobre os passageiros do Titanic.

Dependendo do objetivo de negócio e da quantidade de dados, uma análise estatística pode consumir bastante tempo até que se visualize a resposta

desejada, ou algum aspecto interessante. Muitas vezes a visualização ressalta visualmente características embutidas nos dados. O ambiente *R* disponibiliza uma infinidade de pacotes para visualização de dados.

A título de exemplo, vamos gerar um gráfico em barras mostrando a distribuição dos passageiros por classe (atributo *class*). Execute o comando abaixo.

```
plot(titanic.raw$Class, main="Histograma - Distribuição das
Classes", xlab="Classes", ylab="Total passageiros",
ylim=c(0,1000), las=0)
```

Gere um gráfico similar para saber a distribuição dos passageiros por **idade**. **Dica**: copie o comando acima, e troque o nome do atributo, a legenda do gráfico e a legenda do eixo x. Se a figura ficar ruim, considere trocar a escala do eixo y aumentando o intervalo (parâmetro *ylim*).

Após esta análise você já sabe vários fatos sobre os passageiros, mas ainda não sabe relacionar estas características com a sua sobrevivência. A pergunta de negócio ainda não foi respondida por esta exploração, mas ela revela que os dados podem nos contar ainda mais coisas aplicando uma técnica de mineração de dados. A técnica escolhida foi a associação, que verificará **co-ocorrências comuns de valores** nestes dados.

5 Extraindo Regras de Associação

Existem diferentes implementações no ambiente *R* para a tarefa de associação, e vamos exercitar o popular algoritmo *apriori*. Existem muitos parâmetros para este algoritmo, e você pode consultar todos na aba help. Vamos exercitar alguns.

5.1 Executando o *apriori*: a primeira experiência

Para rodar o *apriori*, execute os comandos abaixo.

```
library(arules)
rules.all <- apriori(titanic.raw)
```

O primeiro comando, *library(arules)*, carrega a biblioteca necessária para usar o algoritmo de regras de associação *apriori*[1]. O segundo comando executa o algoritmo *apriori* sobre o conjunto de dados, e coloca as regras na variável *rules.all*. Observe na console a saída do *apriori*, e responda as questões abaixo.

Q3: Quantas regras o algoritmo *apriori* encontrou para este conjunto de dados? Analise a saída, e veja qual o valor de suporte e de confiança mínimos adotados pelo algoritmo *apriori* (parâmetros default)

Até este momento você não sabe as regras que foram geradas, que estão em sua variável *rules.all*. Para visualizar as regras geradas, execute o comando *inspect* e observe o resultado.

```
inspect(rules.all)
```

Analisando o resultado das regras listadas com o comando *inspect,* notamos que elas não seguem um critério de ordenação específico. Vamos ordenar estas regras segundo a informação suporte, listando as regras com maior suporte no topo da lista de regras. Para tanto execute os comandos abaixo e observe o resultado. As regras ordenadas foram atribuídas a uma nova variável *rules.sorted*.

```
rules.sorted <- sort(rules.all, by="support")
inspect(rules.sorted)
```

Q4: Observe as duas primeiras regras listadas após o ordenamento. Você considera que essas regras são uma associação relevante? Justifique sua resposta.

Q5: Expresse em português como você lê as regras [2] e [3], incluindo a interpretação do suporte e da confiança.

Q6: Analise as demais regras, e selecione uma que você considera relevante em relação ao problema de negócio definido. Informe o número da regra

[1] Se tiver uma mensagem de erro, verifique se o pacote está instalado (Seção 3.2)

escolhida, e justifique sua escolha considerando as métricas de suporte e confiança.

O parâmetro de ordenação aceita ordenar por outras métricas listadas na inspeção das regras. Experimente adaptar o comando acima para ordenar por confiança. **Dica**: copie o comando acima e troque o parâmetro necessário ("*confidence*")

Q7: Observe as quatro primeiras regras listadas após o ordenamento por confiança. Você considera que, porque estas regras têm confiança máxima, elas são mais relevantes para sua questão de negócio? Justifique sua resposta.

5.2 Executando o *apriori*: testando níveis mínimos de suporte/confiança

Em mineração por regras de associação é comum que muitas regras geradas não sejam interessantes ou que poucas regras sejam encontradas dependendo dos valores determinados para os parâmetros de suporte e confiança.

Vamos observar o que ocorre quando usamos os parâmetros suporte e confiança elevados (mínimo 90%), através da execução do comando abaixo.

```
rules.all <- apriori(titanic.raw,
                     parameter = list(supp=0.9, conf=0.9))
rules.sorted <- sort(rules.all, by="support")
inspect(rules.sorted)
```

Q8: Para suporte e confiança iguais a 90%, alguma regra relevante foi retornada?

Agora, execute o comando abaixo para buscar as regras de associação considerando suporte e confiança mais baixos, 10% e 30%, respectivamente.

```
rules.all <- apriori(titanic.raw,
                     parameter = list(supp=0.1, conf=0.3))
rules.sorted <- sort(rules.all, by="support")
inspect(rules.sorted)
```

Q9: Baixando o suporte e confiança mínimos, apareceram mais regras?

Observe as regras [1] a [5] da última execução (Q9) e responda a questão abaixo.

Q10: Elas são relevantes para sua questão de negócio? Explique sua resposta.

5.3 Filtrando Regras de interesse

Nosso objetivo de mineração é encontrar regras de associação que nos tragam *insights* quanto à sobrevivência de passageiros do Titanic. Uma forma de fazermos isto é usar *filtros* que retornem apenas regras de nosso interesse.

Vamos configurar a execução do *apriori* para filtrar:

- apenas regras em que o conseqüente da regra (lado direito, *rhs*) se relacione com o atributo sobrevivência (*Survived*). Todos os outros itens podem aparecer apenas no lado esquerdo da regra (lhs). Fazemos isto usando o parâmetro *appeareance*:
  `appearance=list(rhs=c("Survived=No","Survived=Yes"), default="lhs"`
- apenas regras com suporte mínimo 30%, confiança mínima 60%, e que possuam no mínimo dois itens (para evitar regras onde o lado esquerdo é vazio). Fazemos isto fixando o parâmetro *parameter*:
  `parameter=list(minlen=2, supp=0.3, conf=0.6)`
- eliminar os detalhes do progresso de execução do algoritmo usando o parâmetro *control* :
  `control=list(verbose=F)`

Vamos melhorar também a apresentação das regras reduzindo as métricas para apenas 3 casas após a vírgula, e ordená-las por suporte. Todos estes passos são representados pelos comandos abaixo, que você deve executar.

```
rules.all <- apriori(titanic.raw, control=list(verbose=F),
                     parameter=list(minlen=2, supp=0.3, conf=0.6),
                     appearance=list(rhs=c("Survived=No",
"Survived=Yes"), default="lhs"))
rules.sorted <- sort(rules.all, by="support")
```

Q11: As regras resultantes parecem ser orientadas ao problema de negócio? Explique sua resposta.

Q12: Por que não temos regras que explicam a sobrevivência? Que parâmetro você poderia alterar para conseguir regras que expliquem a sobrevivência?

Q13: Existem regras que lhe parecem redundantes entre si? Quais?

5.4 Eliminando regras redundantes

Vamos baixar o suporte para 5%, na tentativa de encontrar regras que expliquem também a sobrevivência. Vamos ainda executar uma sequência de comandos para eliminar regras redundantes. **Não explicaremos o procedimento de eliminação de redundância por ser complexo e fora do escopo da aula.**

```
rules.all <- apriori(titanic.raw, control = list(verbose=F),
                     parameter = list(minlen=2,supp=0.05,conf=0.6),
                     appearance = list(rhs=c("Survived=No",
                                             "Survived=Yes"),
                                             default="lhs"))
quality(rules.all) <- round(quality(rules.all), digits=3)
rules.sorted <- sort(rules.all, by="support")

# Encontrando regras duplicadas
subset.matrix <- is.subset(rules.sorted,rules.sorted,sparse=FALSE)
subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
redundant <- colSums(subset.matrix, na.rm=T) >= 1

# Listando as regras redundantes encontradas
which(redundant)

# Removendo regras redundantes e listando resultado
rules.pruned <- rules.sorted[!redundant]
inspect(rules.pruned)
```

Q14: Comparadas com as regras da execução anterior (Q11), cite dois exemplos distintos de redundâncias que foram eliminadas? Explique sua resposta em termo dos itens que aparecem nas regras, e suas métricas.

5.4 Usando o lift

Vamos rodar uma última vez o algoritmo *apriori*. Vamos baixar o suporte para 1% e a confiança para 50%, na tentativa de encontrar mais regras que expliquem a sobrevivência. Para interpretar os resultados, use agora o *lift* como critério de ordenamento. O lift seleciona os valores cuja presença **está relacionada à probabilidade** de co-ocorrência. Se o lift for maior que 1, os itens estão estatisticamente correlacionados, e probabilidade dos itens co-ocorrerem aumenta; se for menor que 1, a presença de um diminui a probabilidade de ocorrência do outros; se for em torno de 1, a ocorrência dos itens são independentes. Na maioria das aplicações, nos interessa um lift alto.

Gere novamente as regras de acordo com o comando abaixo, desta vez ordenadas por lift (e eliminando as regras duplicadas).

```
rules.all <- apriori(titanic.raw, control = list(verbose=F),
                     parameter = list(minlen=2,supp=0.01,conf=0.5),
                     appearance = list(rhs=c("Survived=No",
                                             "Survived=Yes"),
                                       default="lhs"))
quality(rules.all) <- round(quality(rules.all), digits=3)
rules.sorted <- sort(rules.all, by="lift")

# Eliminar as regras duplicadas
subset.matrix <- is.subset(rules.sorted,rules.sorted,sparse=FALSE)
subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA
redundant <- colSums(subset.matrix, na.rm=T) >= 1
rules.pruned <- rules.sorted[!redundant]

# Examine os resultados
inspect(rules.pruned)
```

Q15: Compare as regras [4] e [17]. Observe que têm confianças parecidas, mas lifts distintos. O que estas regras nos dizem sobre a sobrevivência?

Q16: Compare as regras [3] e [18]. Embora elas tenham confiança um pouco distintas, o que os respectivos lifts nos dizem sobre a não sobrevivência?

Q17: Com base nestas regras, liste 5 coisas que podem ser inferidas dos dados sobre sobreviventes e não sobreviventes do Titanic com o uso de regras de associação.

5.5 Cuidados ao Interpretar Regras

Olhando as regras da Seção 5.4, responda :

Q18: crianças da segunda classe tiveram mais chances de sobrevivência do que as da primeira classe?

Se você respondeu sim, deve tomar cuidado. A regra [1] simplesmente afirma que todas crianças da 2a classe sobreviveram, mas não provê nenhuma informação que permita comparar a taxa de sobrevivência das crianças das demais classes.

Para entender o porquê, vamos gerar e ordenar mais uma vez as regras, desta vez delimitando a regras que expliquem a sobrevivência de criança (ou não) em função classe. Vamos baixar ainda mais o suporte (2%) e a confiança (30%), devido à baixa prevalência de crianças em nosso dataset.

```
rules <- apriori(titanic.raw,
          parameter = list(minlen=3, supp=0.002, conf=0.3),
          appearance = list(rhs=c("Survived=Yes"),
                            lhs=c("Class=1st", "Class=2nd",
                                  "Class=3rd",
                                  "Age=Child", "Age=Adult"),
          default="none"),
          control = list(verbose=F))
rules.sorted <- sort(rules, by="confidence")
inspect(rules.sorted)
```

Responda

Q19: Responda novamente à questão Q18 utilizando as métricas de suporte/confiança para justificar sua reposta.

Q20: E as crianças da terceira classe? Como sua situação se compara (a) à das demais crianças? (b) dos adultos da primeira classe?

5.6 Resumindo

Em resumo, regras de associação

- com alto suporte têm relação com a prevalência da co-ocorrência na base de dados (crianças de primeira classe não têm suporte alto)

- com alta confiança têm relação com a prevalência da co-ocorrência entre os dados que apresentam os valores do antecedente da regra (dentre as crianças de primeira classe, todas sobreviveram)

- com lift maior que 1 alavanca a probabilidade de ocorrência de um valor dada a ocorrência de outro (relação simétrica – a probabilidade de ter sobrevivido aumenta devido ao fato de ser criança de primeira classe, bem como a probabilidade de ser criança de primeira classe aumenta devido ao fato de ter sobrevivido)

- cuidado com inferências comparativas

6. Visualização

O problema de associação tipicamente retorna muitas regras, e alguns recursos são usados para facilitar a tarefa. Já exercitamos um deles, que é parametrizar o algoritmo para gerar apenas regras com determinadas características (suporte/confiança mínimos, número de itens por regra, regras com determinada estrutura). Aqui vamos explorar algumas visualizações disponíveis no R. Elas estão no pacote *arulesViz*.

```
library(arulesViz)
```

Vamos executar novamente o algoritmo apriori, com suporte e confiançaa muito baixos (0.005 e 0.2 respectivamente), e limitar a regras que tenha como consequente os valores de sobrevivência. Vamos ordenar o resultado por lift e eliminar regras redundantes. O código correspondente está nas linhas 138-152 do script dado. **(Nao vamos repetir todo o procedimento: execute a partir do script, colocando o resultado na variável rules.pruned)**.

Poderíamos nos perguntar quais seriam bons valores de suporte/confiança neste caso. Uma visualização possível é um scatterplot que plota as regras por suporte (eixo x), confiança (eixo y), e lift (cores). Este grafo tem uma versão estática e outra dinâmica, que permite inspecionar a regra.

```
# plot estatico
plot(rules.pruned)
# plot dinamico
plot(rules.pruned,engine = "htmlwidget")
```

Q20: Qual a regra com suporte mais alto? Qual suporte resultaria em regras com maior confiança?

Podemos selecionar somente regras com determinados valores de suporte/confiança, como no comando abaixo.

```
# Selecionar apenas um conjunto de regras
subrules<-
rules.pruned[quality(rules.pruned)$confidence>0.4]
inspect(subrules)
```

Finalmente podemos ver na forma de grafos o conjunto de regras. Novamente, há uma versão estática e dinâmica do comando. Explore um pouco as regras

```
# plot estatico
plot(subrules, method = "graph")
# plot dinamico
plot(subrules, method = "graph",  engine = "htmlwidget")
```

Bibliografia

Este roteiro foi inspirado no estudo de caso apresentado no capítulo 9 do livro *R and Data Mining* (Zhao). Se tiver interesse em saber mais sobre regras de associação e recursos disponíveis no R e seus pacotes usando este dataset, consulte a bibliografia disponibilizada no moodle.