Sistemas Inteligentes e Mineração de Dados

2ª Edição: Do Weka ao Python

José Artur Quilici-Gonzalez
Francisco de Assis Zampirolli
Fábio Rezende de Souza

19 de março de 2026

Índice

Lista de Figuras

Lista de Tabelas

Projeto de construção do livro de Sistemas Inteligentes e Mineração de Dados (2ª Edição)

Bem-vindo ao material didático em construção! Este livro utiliza o **Quarto** para integrar teoria e prática.

Status do Projeto

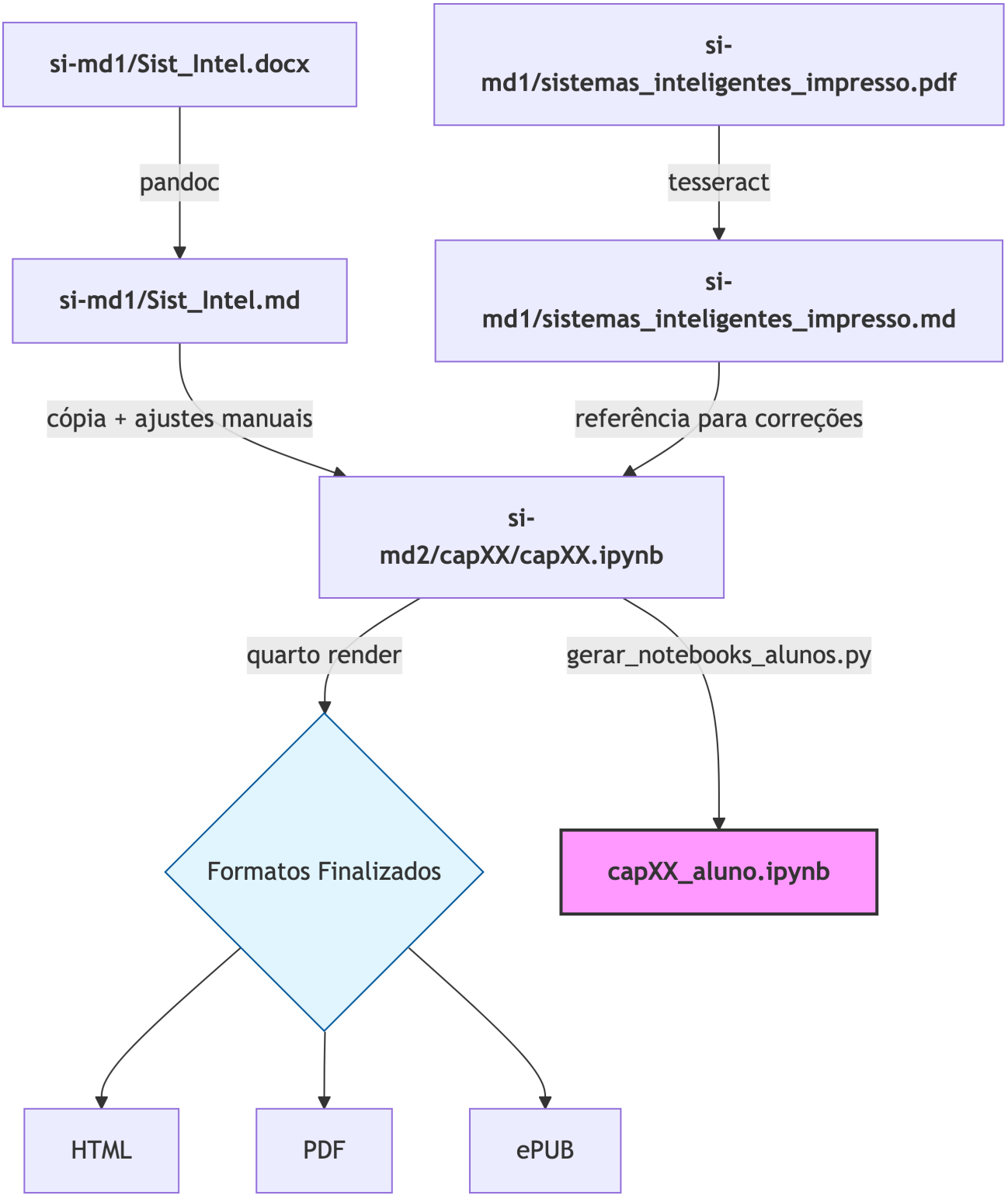
- **Capítulos 1 e 2:** Em fase de revisão final e melhorias nos exemplos em Python.
- **Capítulos 3, 4 e 5:** Falta incluir exemplos em Python.
- **Capítulo 6:** Em construção.
- **Foco:** Simplificar os laboratórios em Python para criar exemplos curtos, práticos e **extremamente motivacionais**.

Estrutura Simplificada

- `si-md1/`: Material legado (PDF/Docx). **Não editar.**
- `si-md2/`: **Pasta de trabalho ativa.**
  - `_quarto.yml`: Configuração mestre (sumário e temas).
  - `capXX/`: Pasta de cada capítulo (**Editar** – contém o `capXX.ipynb` e `/images`).
  - `references.bib`: Base de dados bibliográfica global.
  - `notebooks_alunos/`: Versão processada para distribuição (gerada via script).

Processo:

1. Gerado `si-md1/Sist_Intel_j.md` de `si-md1/Sist_Intel_j.docx` com pandoc.
2. Copiado `si-md1/Sist_Intel_j.md` para `si-md2/capXX/capXX.ipynb` (e ajustes manuais).
3. Gerado `si-md1/sistemas_inteligentes_impresso.md` de `si-md1/sistemas_inteligentes_impresso.pdf` com tesseract.
4. Pequenas correções em `si-md2/capXX/capXX.ipynb` usando `si-md1/sistemas_inteligentes_impresso.md`.
5. Ajustes manuais para funcionar `quarto render` em html, pdf e epub.
6. O formato ipynb renderizado ocorre através do método `gerar_notebooks_alunos.py`.



Workflow Rápido (Terminal)

1. **Atualizar:** `git pull origin main`
2. **Limpar caches:** `./limpar.sh`
3. **Gerar Versão Aluno:** `python gerar_notebooks_alunos.py --batch references.bib`
4. **Publicar Tudo (HTML+PDF+Git):** `./publish_all.sh`

5. **Manual (Git):**

```
git add .
git commit -m "Explique sua alteração"
git push origin main
```

Checklist de Qualidade

- ☐ Imagens estão na pasta `images/` interna do capítulo?
- ☐ Labels começam com o prefixo correto (`fig-`, `tbl-`, `eq-`) e o número do capítulo?
- ☐ O arquivo `references.bib` foi atualizado com as novas obras?
- ☐ O código do notebook executa do início ao fim sem erros?

Prefácio

Esta obra é a evolução natural da **primeira edição (2014)**, expandindo o foco do software Weka para o ecossistema Python. O livro mantém sua visão pragmática da Mineração de Dados, unindo a teoria clássica à prática interativa via Google Colab e tutoria por IA. Através de seis capítulos sequenciais, o leitor desenvolve a sensibilidade necessária para explorar algoritmos e selecionar a técnica mais apropriada para cada base de dados.

Prefácio da Primeira Edição

O início do século XXI é caracterizado pela era da informação e crescimento exponencial de dados gerados em praticamente todas as áreas de atividade humana. Cada vez mais se torna necessário o conhecimento e aperfeiçoamento de ferramentas apropriadas para extrair informações úteis destes dados. A Mineração de Dados combina inteligência artificial, aprendizagem de máquina, estatística, base de dados e técnicas avançadas de programação para tratar grandes volumes de dados.

Este livro foi escrito com o objetivo de oferecer uma visão pragmática da Mineração de Dados. Ele é apropriado para ser utilizado como livro texto de curso na área, trazendo vários exercícios práticos utilizando o pacote gratuito e aberto Weka de rotinas escritas em Java, incluindo exercícios no final de cada capítulo, juntamente com uma boa lista de referências bibliográficas.

O livro procura passar uma experiência prática de uso do pacote de software permitindo que o leitor seja introduzido na área vivenciando as várias modalidades de ferramentas disponíveis. Como nenhum algoritmo ou técnica tem um desempenho superior para qualquer base de dados, é importante que o leitor adquira uma sensibilidade para poder explorar e escolher a técnica mais apropriada em cada caso.

O livro está organizado em seis capítulos que devem ser lidos na sua sequência.

O Capítulo 1, Sistemas Inteligentes traz as principais definições necessárias, como sistemas, inteligência, dados, informação, conhecimento e desempenho. É feita uma breve descrição das três etapas da Descoberta ou Extração de Conhecimento em Base de Dados: pré-processamento, mineração de dados e pós-processamento. A Mineração de Dados, por sua vez é dividida nas tarefas de Associação, Classificação, Agrupamento e Detecção de Anomalias. O capítulo é finalizado fazendo considerações sobre a questão ética da Mineração de Dados, mencionando que o uso destas informações podem levar à medidas preventivas de segurança pública.

O Capítulo 2, Mineração de Dados e Regras de Associação, explica como os dados devem ser estruturados através de suas transações ou exemplos, juntamente com seus atributos. O capítulo é dedicado à identificação de Regras de Associação também denominadas regras IF-THEN. Os indicadores de suporte e confiança ou acurácia são utilizados na avaliação das melhores regras de associação. É visto o algoritmo apriori utilizado para criar as regras de identificação. É visto um exemplo completo, passo-a-passo de identificação de Regras de Associação utilizando a ferramenta Weka.

O Capítulo 3, Classificação e Árvores de Decisão, utiliza o famoso exemplo de classificação de 3 espécies de Flor Íris utilizando a árvore de decisão, onde cada nó da árvore é uma pergunta a ser testada. O nó raiz é a pergunta inicial e as folhas são as classes resultantes da aplicação da árvore de decisão.

É introduzido o algoritmo ID3, baseado na noção de informação do Shannon e comparado a um algoritmo de escolha aleatória na construção da árvore de decisão, ilustrando a compacticidade da árvore gerada pelo algoritmo ID3. É ilustrado também o processo de extração de regras de decisão a partir da árvore de decisão. Ainda neste capítulo, o conceito de aprendizado supervisionado é introduzido, juntamente com o conceito de dados de treinamento e de testes e o processo de classificação. A avaliação dos resultados é feita com o uso da Matriz de Confusão. O capítulo termina com um exemplo completo utilizando o Weka para gerar o classificador via Árvore de Decisão e avaliar seu resultado com a Matriz de Confusão.

O Capítulo 4, Classificação e Regras de Classificação, trata da Classificação de dados através de Regras de Classificação e o processo de geração automática dessas Regras. O objetivo desse capítulo é passar uma ideia geral do funcionamento dos algoritmo oneR (uma regra), classificadores lineares, redes neurais e Máquinas de Vetores de Suporte (MSVS), cujo entendimento é indispensável para o ajuste adequado de seus parâmetros e para a correta interpretação de seus resultados. Pela simplicidade, inicialmente é visto o algoritmo oneR e em seguida o algoritmo PRISM, que utiliza o princípio da cobertura para a criação das regras. Os indicadores de resultados especificidade e sensibilidade são também introduzidos neste capítulo que termina com uma visão sobre as melhores formas de utilizar os dados para as fases de treinamento e de teste: Técnica da ressubstituição ou uso do conjunto de treinamento; método da divisão da amostra; método da validação cruzada; e método deixe-um-de-fora.

No Capítulo 5, Máquina de Vetores de Suporte (MVS), são vistos inicialmente os classificadores lineares como o Perceptron e os conceitos de otimização da função custo, conjuntos convexos, mínimo global e mínimo local. Nos Classificadores não lineares, a reta é substituída por uma função polinomial. As Máquinas de Vetores de Suporte são introduzidas através do Princípio da margem máxima que é uma das principais características da MVS que traz mais estabilidade e desempenho deste classificador. O capítulo traz ainda os conceitos de kernel da MVS, o truque do kernel, Parâmetro de Complexidade C e o conceito da Praga da dimensionalidade. O capítulo termina ensinando a utilizar o Weka para visualizar as bordas de decisão do MVS.

O Capítulo 6, Aplicações de SVM Usando Imagens traz uma introdução à classificação de imagens digitais utilizando SVM. Uma série de ilustrações utilizando o SVM para reconhecimento de expressões faciais com imagens de apenas 49 pixels ajuda o leitor a entender problemas mais complexos de classificação que utilizam imagens de vários megapixels. Além do uso dos próprios pixels da imagem como atributos, são utilizados o seu histograma e atributos topológicos tais como perímetro, área, excentricidade, orientação, razão de aspecto, entre outros. O capítulo traz ainda exemplos de classificação dos tecidos adiposo e epitelial.

Roberto de Alencar Lotufo, professor titular Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas – Unicamp

Prefácio da Segunda Edição

Bem-vindo à segunda edição do livro **Sistemas Inteligentes e Mineração de Dados: Do Weka ao Python**.

Esta edição reflete a evolução recente da área de inteligência artificial e mineração de dados, incorporando novas práticas, ferramentas e abordagens utilizadas tanto na academia quanto na indústria. Em especial, o material foi atualizado para integrar o ecossistema Python, hoje amplamente utilizado para análise de dados e aprendizado de máquina.

Como acessar este livro

	Formato	Acesso
	HTML *(recomendado)*	fzampirolli.github.io/si-md2/
	PDF	fzampirolli.github.io/si-md2/livro.pdf
	Notebook (.ipynb)	Botão *Executar Colab* no início de cada capítulo

Sugestão: estude pela versão HTML, pratique em Python nos notebooks e use o PDF como referência no NotebookLM ou para impressão.
Nota (Colab): imagens carregadas do GitHub podem demorar alguns segundos — se não aparecerem, **recarregue a página**.

Motivação

A primeira edição deste livro tinha como foco principal a ferramenta **Weka**. Com o crescimento do ecossistema **Python** e de bibliotecas como *pandas*, *NumPy* e *scikit-learn*, tornou-se necessário atualizar o material didático para acompanhar as práticas contemporâneas da área.

Esta segunda edição mantém os fundamentos conceituais da mineração de dados, mas apresenta exemplos e atividades práticas utilizando Python, favorecendo uma experiência mais atual e integrada com ferramentas amplamente

utilizadas.

Estrutura do Livro

O conteúdo está organizado em seis capítulos:

- **Capítulo 1:** Introdução aos Sistemas Inteligentes
- **Capítulo 2:** Fundamentos de Mineração de Dados
- **Capítulo 3:** Aprendizado de Máquina Supervisionado
- **Capítulo 4:** Aprendizado Não Supervisionado
- **Capítulo 5:** Avaliação e Validação de Modelos
- **Capítulo 6:** Aplicações Práticas e Estudos de Caso

Como Estudar Este Material

Este livro foi projetado para combinar **leitura teórica, experimentação prática e apoio de ferramentas de IA**.

Prática Interativa com Notebooks

Cada capítulo possui exemplos práticos que podem ser executados no **Google Colab** (pelo botão *Executar Colab* no início do capítulo) ou localmente via **Jupyter Lab**. Recomenda-se executar e modificar os códigos apresentados, explorando diferentes parâmetros e observando seus efeitos nos resultados.

Tutor Inteligente com NotebookLM

O **NotebookLM** pode ser utilizado como um tutor auxiliar para revisão, geração de resumos e esclarecimento de dúvidas. Como a ferramenta ainda não processa arquivos `.ipynb` diretamente, utilize a **versão em PDF de cada capítulo** como fonte.

> 🚀 **Tutor de IA do Livro**
>
> Cada capítulo possui um projeto configurado no NotebookLM.
> Exemplo para o Capítulo 1:
> **Link do Projeto NotebookLM do Capítulo 1**

Sugestões ao Leitor

- **Experimente:** modifique parâmetros e explore variações dos exemplos apresentados.
- **Investigue:** utilize ferramentas de IA para aprofundar conceitos e revisar conteúdos.
- **Conecte teoria e prática:** observe como os conceitos discutidos se refletem nos resultados obtidos nos experimentos.

Agradecimentos

Agradecemos aos alunos e instituições que contribuíram para o desenvolvimento e aprimoramento deste material.

Quilici-Gonzalez, Zampirolli e Souza, 2026

Capítulo 1

Sistemas Inteligentes

1.1 Introdução

Para entender o significado de "**Sistemas Inteligentes**", talvez seja melhor iniciar com uma acareação sobre "Sistema" e "Inteligente". Do grego, o termo "sistema" significa "combinar", "ajustar", "formar um conjunto", como por exemplo em "sistema respiratório", que reúne vários órgãos responsáveis pelo fornecimento de oxigênio para as células, ou "sistema financeiro", que compreende pessoas, prédio, regras etc., combinados para realizar operações financeiras. Um sistema pode ser formado por um conjunto de pessoas, recursos, instalações e métodos direcionados a atingir um fim.

Embora seja muito difícil definir "inteligência", verifica-se que no "comportamento inteligente" dos seres humanos estão presentes ao menos a habilidade de resolver novos problemas, de adaptar-se às mudanças do ambiente, de fazer previsão, de raciocinar, de planejar, de tomar decisões compatíveis com determinada situação, de melhorar seu desempenho com a experiência, de comunicar, de entender, de fazer inferências lógicas, entre outras. Inteligência também tem sido definida como síntese de conhecimentos, ou seja, um **agente** possuindo a capacidade de sintetizar conhecimentos pode atingir um objetivo identificável exercitando algumas das habilidades acima enumeradas, como a do raciocínio.

Nas máquinas, a simulação de comportamento inteligente geralmente reproduz apenas partes das habilidades humanas, dentre elas a capacidade de "aprender". Basicamente, em uma abordagem operacional que mantenha somente os atributos funcionais do conceito, por Sistema Inteligente entende-se aquele sistema capaz de melhorar seu desempenho a partir da própria experiência. Em outras palavras, um Sistema Inteligente deve ter a capacidade de "aprender" com as informações disponíveis, ou com seus erros.

Note que na primeira tentativa de definição, inteligência estava associada à síntese de conhecimento e raciocínio para atingir um objetivo, enquanto que nesta segunda definição, inteligência está associada à melhora de desempenho e comportamento. E melhora de desempenho não necessariamente envolve conhecimento.

Entre os seres humanos, o processo cognitivo de *aprender* geralmente pressupõe *intencionalidade* ou *propósito* por parte do **sujeito**, caso contrário, utiliza-se o termo *treinamento*, quando então a intencionalidade passa a ser associada ao treinador e não ao agente sendo treinado. Porém, falar de intencionalidade em máquinas é uma questão filosoficamente controversa. Quando uma máquina produz respostas que fazem sentido para o **usuário**, pode-se detectar intencionalidade nestas respostas, mas trata-se de uma *intencionalidade derivada* da interpretação do usuário. Por isso, quando afirmamos que determinada máquina, determinado agente ou programa tem a capacidade de "aprender", estamos usando o conceito num sentido mais raso, i.e., estamos usando uma abordagem operacional ou funcional.

Um sistema inteligente pode ser um sistema físico (essencialmente hardware com capacidade adaptativa), um sistema computacional (predominantemente um software que comanda uma máquina) ou um sistema híbrido (um robô com sensores, atuadores, sistema operacional, programas computacionais etc.). Assim, por exemplo, um sistema inteligente pode ser aquele capaz de estabelecer conexão automaticamente com a internet, executar **aplicações nativas** ou **em nuvem**, fazer a análise de dados coletados e tomar uma decisão. Para o escopo deste livro, vamos nos limitar aos sistemas computacionais inteligentes, que geralmente utilizam *aprendizado de máquina* para tomar decisões inteligentes em áreas científicas, comerciais, de segurança, entre outras.

Independentemente da natureza do sistema inteligente, ele terá que demonstrar habilidade para adaptar-se a mudanças em seu ambiente não previstas pelo projetista e tomar decisões baseadas em novos conhecimentos adquiridos com experiência própria. Para ilustrar a diferença entre um sistema tradicional e um inteligente, primeiramente pense num programa capaz de emitir extratos bancários a milhões de correntistas. Embora a implementação de tal programa possa ser algo nada simples, as condições em que ele vai operar podem ser minuciosamente antecipadas, e seu comportamento será essencialmente algorítmico. Agora pense num programa reconhecedor de voz humana. O projetista do sistema não pode prever quem vai utilizar o sistema, muito menos o conteúdo de sua fala. No comportamento desse sistema deve haver um componente empírico. Ou pense num sistema capaz de ler endereços preenchidos manualmente. Neste caso, não basta tentar utilizar apenas um tradicional reconhecedor óptico de caracteres, porque cada ser humano tem seu próprio estilo de escrita. É preciso desenvolver um programa que "aprenda" a reconhecer caracteres por meio de um treinamento com muitas variações de um mesmo caractere.

Ao tentar definir inteligência, acabamos usando conceitos como *conhecimento* e *desempenho*. Precisamos, portanto, agora discorrer como se produz conhecimento e o que se entende por desempenho. Como se sabe, esta sequência de definições remete a uma recursão infinita. Por isso, para simplificar nossa tarefa e torná-la factível, vamos adotar a estratégia de definir *dado* e *informação* para chegar a *conhecimento*, e daí retornar ao conceito de *inteligência* associada a *desempenho*.

1.2 Dado, Informação, Conhecimento, Desempenho e Inteligência

Ao tentar definir conceitos muito abrangentes é conveniente limitar o domínio de aplicação, principalmente pela dificuldade de se chegar a um consenso entre especialistas, e depois porque, em nosso caso, estamos interessados em direcionar as definições ao domínio de **Tecnologia e Sistemas de Informação**. Pode parecer paradoxal afirmar que em plena "Era da Informação" não haja uma visão clara do que é *informação*. Convém, no entanto, relembrar que este mesmo cenário já ocorreu em outros períodos da história humana.

A genética mendeliana do século XIX, por exemplo, descobriu experimentalmente que certas características das ervilhas poderiam ser transmitidas através da hibridação das plantas. Mas foi preciso esperar por um século até a descoberta da estrutura do DNA para que uma teoria robusta explicasse o mecanismo da herança genética. Antes disso, as explicações dos fenômenos de transmissão de herança genética eram meramente funcionais.

É certo que vivemos na era da informação, mas ainda não temos pleno domínio sobre ela, justamente porque não conseguimos caracterizá-la adequadamente. Mesmo assim, é importante definir Dado, Informação e Conhecimento porque eles estão na base do Aprendizado.

1.2.1 Dado

É um **fato registrado** ou uma quantidade (ou qualidade) conhecida sem a necessidade de elaboração. Por exemplo, o peso (ou massa!) de uma pessoa, o valor da dívida de um cliente, os registros de temperatura dos últimos anos, os conceitos ou as notas de provas de alunos, o nome de um empregado etc.

1.2.2 Informação

Quando os dados apresentam alguma relação entre si, eles podem ser contextualizados ou interpretados, adquirindo um **significado**. Neste caso, a informação envolve **dados contextualizados** ou padrões de associação escondidos numa coleção de dados. Alguns dados podem fazer referência a outros dados, e não a fatos ou objetos. Esses dados são conhecidos como metadados e se confundem com o conceito de informação, ou estão na origem da informação.

Por exemplo, associando-se o peso (ou massa) de uma pessoa à sua altura, obtém-se o Índice de Massa Corporal (IMC). Nesse contexto, peso e altura são dados, IMC é informação. Apenas o valor da dívida de um cliente não permite a um gerente de banco decidir sobre um empréstimo. É necessário levar em conta a renda do cliente, ou talvez o histórico de suas movimentações, para que uma interpretação confiável de seus dados possa ser feita. A taxa de reprovação em uma disciplina é uma informação que se obtém a partir dos dados reunidos dos conceitos de provas dos alunos.

Note que o que consideramos informação em um contexto, pode tornar-se dado em outro, dependendo da referência utilizada. Por exemplo, o IMC quando relacionado à idade (ou ao sexo) de um paciente pode funcionar como um dado, que associado a outro dado, pode revelar importante informação sobre sua saúde.

1.2.3 Conhecimento

As **informações úteis a um propósito** permitem ao indivíduo compreender uma situação, ou formar uma crença justificada sobre um fato. Portanto, o conhecimento se forma a partir das informações necessárias para o **entendimento** de uma situação. Nesse contexto, conhecimento é o resultado da análise das informações relacionadas a um fato ou evento, ou ainda a percepção de como certa informação pode ajudar na realização de uma tarefa específica.

No mundo corporativo, o conhecimento produzido com regras de inferência sobre as informações analíticas da empresa ou com mineração de dados operacionais torna-se um importante aliado durante o processo de tomada de decisões gerenciais.

Embora a diferenciação entre dados, informação e conhecimento seja tênue, é comumente aceito que estes três conceitos podem ser relacionados em uma estrutura hierárquica, como a representada pela Figura 1.1.

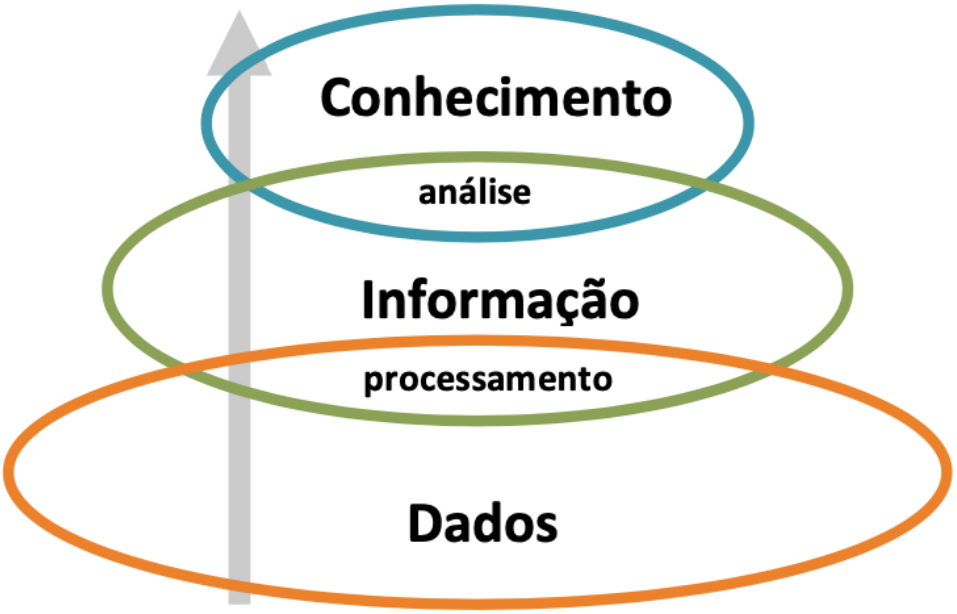


Figura 1.1: Relação entre Dados, Informação e Conhecimento.

1.2.4 Desempenho

O desempenho de um sistema pode ser aferido comparando-se o resultado obtido com o esperado, ou com os recursos empregados para chegar àquele resultado. Quando alguém toma um remédio para curar uma doença, e se cura, geralmente dizemos que tal remédio foi **eficaz** porque o resultado obtido se aproximou do esperado. Note que neste caso, para medir o desempenho do sistema, comparamos sua saída com o resultado esperado.

Por outro lado, quando dizemos que a **eficiência** de um motor a combustão está em torno de 30%, estamos comparando a energia de entrada (combustível) com a de saída (potência mecânica). Agora o desempenho foi caracterizado pela relação entre entrada e saída do sistema.

1.2.5 Inteligência

Para manipular com eficiência o grande volume de dados atualmente gerados pelos processos operacionais de empresas ou instituições foram desenvolvidos Sistemas de Software Inteligentes, ou seja, sistemas capazes de aplicar conhecimentos adquiridos e melhorar seu desempenho a partir da própria experiência. No mundo corporativo, inteligência, portanto, pode ser vista como a aplicação do conhecimento gerado para a obtenção de um fim determinado, que traga uma vantagem competitiva ou evite o comprometimento de interesses.

Nas últimas décadas foram criados vários sistemas para otimizar cada etapa do processo de extração de conhecimento. Esses sistemas cuidam desde a coleta de dados operacionais de uma organização, incluindo os dados fornecidos por um especialista, passando pela Mineração de Dados, até a análise dos resultados. Atualmente o processo geral que abrange os sistemas citados para a extração de conhecimento é chamado de *Descoberta ou Extração de Conhecimento em Bases de Dados*, ou *Knowledge Discovery in Databases*, ou simplesmente *KDD*.

1.3 Descoberta de Conhecimento em Bases de Dados, ou KDD

Modelos ideais podem não resolver plenamente problemas de sistemas reais, porque dados reais geralmente apresentam redundâncias, ausências de valores, erros, inconsistências etc. Processamento de dados geralmente requer pré-processamento, pois a qualidade dos dados de entrada pode ter um impacto significativo sobre a etapa seguinte, a de Mineração de Dados.

O que se espera com a Mineração de Dados é obter conhecimento, ou uma representação de conhecimento na forma de regras ou de estruturas equivalentes, que oriente uma decisão. Além disso, quando aplicado de modo inteligente, esse conhecimento alarga horizontes, permitindo fazer previsões (ou modelagem preditiva), descobrir novas associações (ou modelagem descritiva), refinar agrupamentos efetuados por critério de semelhança ou certificar-se de anomalias de comportamento.

E, com a representação do conhecimento em mãos, há a necessidade de um pós-processamento para interpretar e validar os resultados obtidos. Considere o caso de Sistemas Inteligentes conhecidos como Sistemas Especialistas, utilizados por exemplo para diagnóstico médico, que precisam expor de forma inteligível a um especialista todas as etapas do encadeamento lógico de inferências que levaram àquele resultado. Caso contrário, o médico poderá não se sentir seguro para acolher o diagnóstico produzido pelo sistema.

A Figura 1.2 ilustra as principais etapas do processo de Descoberta de Conhecimento em Bases de Dados. O propósito do ***Pré-processamento*** é eliminar eventuais problemas nos dados brutos e colocá-los num formato apropriado para a etapa seguinte, a da ***Mineração de Dados***, visando com isso melhorar significativamente a eficiência dos algoritmos que serão usados. Os dados originais podem ter sido coletados e reunidos por diferentes departamentos, apresentando valores espúrios ou ausentes, ou contendo redundâncias.

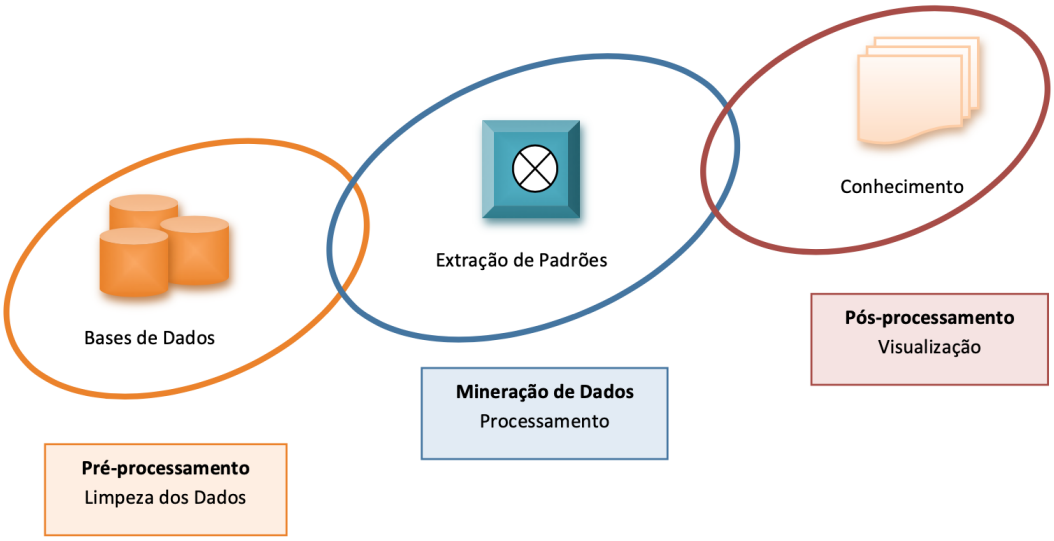


Figura 1.2: Processo de Descoberta de Conhecimento ou KDD.

Para as grandes empresas, em que geralmente há múltiplas fontes de bases de dados fisicamente separadas, é comum introduzir uma etapa intermediária de pré-processamento. As principais transformações nos dados envolvem limpeza e fusão dos dados brutos, eliminação de ruídos e redundâncias, diminuição do número de variáveis e otimização da forma de acesso.

Essas transformações podem ser feitas reunindo todos os dados num grande "depósito" ou repositório de dados, conhecido como ***Data Warehouse***, que normalmente permite a pesquisa por assunto, por período de tempo, por

cliente, entre outras. Dependendo da quantidade de dados envolvidos, esta etapa pode se tornar a mais demorada e trabalhosa das três etapas.

Na etapa de ***Mineração de Dados***, o objetivo é descobrir de forma automatizada relações ou padrões implícitos em grandes quantidades de dados, ou comprovar alguma hipótese a partir de informações até então não facilmente perceptíveis nos dados. Antes do desenvolvimento da Mineração de Dados, a Estatística já oferecia várias técnicas para análise de dados, porém isso era feito de forma manual, restringindo sua aplicação a bases de dados relativamente pequenas. A Mineração de Dados evoluiu com a disseminação generalizada de sistemas computacionais, que quando associados a técnicas de Inteligência Artificial, por exemplo aplicadas na área de Banco de Dados, permitiriam a geração automática de conhecimentos implícitos nos dados.

Finalmente, na etapa de ***Pós-processamento***, é possível visualizar e interpretar as regras ou os padrões obtidos com a Mineração de Dados, e eliminar os resultados equivocados ou pouco representativos. Nesta fase é bem comum a aplicação de testes estatísticos para validação dos resultados. Dependendo dos valores obtidos, o processo de Descoberta de Conhecimento pode sugerir uma nova iteração, com uma volta à etapa de Mineração de Dados, quando então o processo é repetido com valores de parâmetros modificados ou com a utilização de novos algoritmos.

Como se vê, a etapa em que efetivamente se dá a descoberta de conhecimento é a da **Mineração de Dados**. Muitas ferramentas originalmente implementadas apenas para a etapa de Mineração de Dados atualmente permitem fazer também o pré e o pós-processamento. Por isso, na prática, vários autores consideram os termos Descoberta de Conhecimento e Mineração de Dados como equivalentes. A seguir, vamos focar de modo mais detalhado as principais tarefas da Mineração de Dados.

1.4 Mineração de Dados

O crescimento exponencial de dados gerados em praticamente quase todas as áreas de atividade humana, seja ela científica, comercial, lazer, industrial, entre outras, acabou tornando inviável, a certa altura, a análise sistemática baseada em técnicas estatísticas manuais de grandes bases de dados. É nesse contexto que pesquisadores da área de Inteligência Artificial, combinando técnicas da Estatística e de programação avançada, começaram a desenvolver programas para extração e sumarização automática de informação útil de grandes Bases de Dados e criaram uma nova disciplina chamada *Mineração de Dados*.

Na Mineração de Dados, dependendo do objetivo a ser atingido, ou seja, do tipo de conhecimento a ser gerado, diferentes tarefas poderão ser executadas sobre a Base de Dados. As principais tarefas da Mineração de Dados são:

Associação – na tarefa de Associação, partindo de um conjunto de itens o objetivo é encontrar **Regras de Associação** entre itens que ocorrem simultaneamente. Um conjunto de itens pode ser uma cesta de artigos com código de barras vendidos num supermercado ou numa livraria on-line, e o fato de dois artigos serem frequentemente comprados conjuntamente é de grande interesse para o proprietário. Note que a ocorrência simultânea de dois ou mais itens não implica necessariamente relação de causalidade. Note também que na prática o número de regras de associação de uma cesta pequena de artigos pode ser proibitivamente elevado. Isso nos obriga a lançar mão de medidas de qualidade ou de desempenho para eliminar as inúmeras regras que se aplicam a poucas transações desses artigos, e selecionar apenas as Regras de Associação que tenham uma grande abrangência ou cobertura sobre o total de transações.

Classificação – na tarefa de Classificação, a partir de uma série de exemplos previamente rotulados em duas ou mais classes, o objetivo é aprender a classificar um novo exemplo, cuja classe é desconhecida. As classes apresentam resultados discretos, como sim/não, ou baixo, médio e alto risco etc. Por exemplo, para as classes remédio e vitamina, com base nas características de determinado item X, interessa saber qual a classe este item melhor se encaixa. Quando os resultados esperados não pertencerem a classes discretas, ou seja, quando a variável de predição for real, a classificação recebe o nome de **Regressão**.

Clusterização – na tarefa de Clusterização ou Agrupamento, um grupo de registros diversos de uma Base de Dados deve ser segmentado em subgrupos contendo registros similares. Ao contrário da Classificação, na Clusterização não há classes previamente definidas. O critério de agrupamento entre registros é a similaridade dos atributos ou das características dos registros.

Detecção de Anomalias – na tarefa de Detecção de Anomalias, o objetivo é detectar desvios de um comportamento considerado normal, e caracterizar uma situação como anormal ou não. Empresas de cartão de crédito utilizam os registros de movimentação de um cartão para impedir que um fraudador decidido a fazer muitas compras em pouco tempo se passe pelo proprietário legítimo do cartão.

Em linhas gerais, as tarefas da Mineração de Dados são de natureza ***descritiva*** ou ***preditiva***. Numa rede de supermercados pode ser extremamente valioso descobrir quais produtos são comprados juntos. Por exemplo, ao se descobrir que os produtos A, G e M são quase sempre comprados juntos, a simples alteração da posição das gôndolas desses produtos pode aumentar as vendas. Em situações desse tipo, em que o conhecimento extraído auxilia na interpretação da massa de dados, os padrões de associação descobertos nos produtos oferecidos têm um caráter **essencialmente descritivo**.

Já em certos ramos empresariais, pode ser muito importante descobrir qual o comportamento típico de clientes que estão prestes a mudar de fornecedor, e, com base em suas características, construir um *modelo* que auxilie na previsão de comportamentos futuros. Geralmente vale mais a pena usar técnicas de fidelização para reter um cliente bom e antigo do que investir na busca de novos clientes, cuja fidelidade ainda é incerta. Usando o modelo de comportamento previamente desenvolvido, juntamente com o histórico de compras de um cliente, é possível fazer algumas inferências e prever qual será o possível desfecho. Tarefas que procuram predizer se um item pertence a uma classe ou não, são **essencialmente preditivas**.

A Figura 1.3 ilustra a classificação das principais tarefas de Mineração de Dados em Atividades Descritivas ou Preditivas. Vale ressaltar que, dependendo de como for implementada a "Detecção de Anomalia", esta tarefa poderá ser mais bem caracterizada como de natureza descritiva.

Figura 1.3: Classificação da Natureza das Tarefas da Mineração de Dados.

1.5 Mineração de Dados e suas Implicações Éticas

Como o uso de dados pessoais na Mineração de Dados pode afetar a privacidade de pessoas ou discriminar grupos socialmente fragilizados, suas implicações éticas têm sido amplamente discutidas na imprensa, na academia, nos meios jurídicos e no mundo corporativo. Quando se pensa na política das companhias de seguro, por exemplo, argumenta-se que um cliente é geralmente julgado mais pelos atributos do grupo ao qual ele se enquadra e nem tanto pelas suas próprias características.

Os dados sobre pagamentos feitos com cartão de crédito podem expor as preferências religiosas de seu dono. Seus hábitos de compras de livros podem revelar suas preferências políticas, ou gastos elevados podem colocá-lo inadvertidamente no grupo de clientes de alto risco para empréstimo bancário. O CEP de um candidato pode apontar que ele vive em uma região considerada problemática ou num bairro nobre, seus dados médicos podem lhe custar uma vaga numa grande empresa.

Por outro lado, as estatísticas sobre determinada modalidade de crime podem ajudar as autoridades policiais a adotar medidas preventivas. O levantamento estatístico de regiões carentes frequentemente serve de orientação aos formuladores de políticas públicas na hora de conceber ações compensatórias localizadas. A análise dos dados de gasto do governo pode ajudar a corrigir distorções ou denunciar mazelas. O conhecimento prévio de que algumas doenças e problemas de saúde parecem estar mais fortemente associados a uma etnia que a outra auxilia o médico na hora de solicitar exames médicos, mesmo que o paciente não apresente nenhum sintoma.

Como se vê, não é nada simples traçar uma linha divisória que condene ou justifique o uso ético ou legal de dados armazenados. E se os dados tiverem sido coletados sem o conhecimento do usuário, a questão torna-se ainda mais

controversa. Por esta razão, e por se tratar de uma tecnologia relativamente jovem, recomenda-se que a Mineração de Dados seja empregada com cautela e, se possível, com a anuência prévia das pessoas cujos dados foram coletados.

1.6 🧪 Prática em Python - Ambiente Inicial

Abaixo, apresentamos códigos simples para verificar se as bibliotecas fundamentais do ecossistema Python (Pandas e Matplotlib) estão funcionando.

Se você estiver no **Google Colab**, basta clicar no ícone de play ao lado da célula. No **Jupyter**, pressione `Shift + Enter`.

1.6.1 🔹 Célula de Texto (Markdown)

Esta é uma célula de **texto**. Para editá-la no Jupyter ou no Colab, basta dar um **duplo clique** aqui. Células de texto são utilizadas para explicações, fórmulas matemáticas e referências.

1.6.2 🔹 Células de Código Python

Abaixo, você encontrará uma célula de **código**. Diferente do texto, as células de código podem ser executadas para realizar cálculos e gerar gráficos.

Como executar: 1. Clique dentro da célula abaixo. 2. Pressione **`Shift + Enter`** (ou clique no botão **Play** no Google Colab).

```
# Exemplo de execução para testar o ambiente
print("Se você está lendo esta mensagem, o código foi executado com sucesso!")

# Teste de cálculo simples
a = 10    # Variável inteira 'a' com valor 10
b = 20.5  # Variável float 'b' com valor 20.5
print(f"O resultado da soma de {a} + {b} é: {a + b}")
```

```
Se você está lendo esta mensagem, o código foi executado com sucesso!
O resultado da soma de 10 + 20.5 é: 30.5
```

O código a seguir mostra a execução de um laço de repetição (`for`) para iterar sobre uma sequência numérica de 0 a 4, gerada pela função `range(5)`. Do ponto de vista da Mineração de Dados, esse fluxo representa o processamento sequencial de **Dados** brutos para a geração de **Informação** estruturada, servindo como base para a identificação de padrões em sistemas inteligentes.

```
print("Esta é uma célula de código Python para testar um laço!  ")

lista = list(range(5)) # cria uma lista de números de 0 a 4
print("Lista de números de 0 a 4:", lista)

# Teste simples de laço para percorrer a lista e imprimir cada valor
for i in lista:
    print("Valor:", i)
```

```
Esta é uma célula de código Python para testar um laço!
Lista de números de 0 a 4: [0, 1, 2, 3, 4]
Valor: 0
Valor: 1
Valor: 2
Valor: 3
Valor: 4
```

1.6.3 🔹 Função Simples

O código a seguir mostra o fluxo técnico de transformação de **Dados** (sequência numérica) em **Informação** (valores ao quadrado). Embora a extração de **Conhecimento** seja simplificada neste cenário, o exemplo ilustra a base

necessária para que algoritmos de Mineração de Dados identifiquem padrões de comportamento e tendências em grandes volumes de informação, permitindo que o sistema aprenda a lógica subjacente aos dados para melhorar seu desempenho futuro.

```
# 1. Definição da Função
def quadrado(x):
    """
    Recebe um número 'x' e retorna o seu valor ao quadrado (x elevado a 2).
    Esta é uma 'Docstring', usada para documentar o que a função faz.
    """
    return x**2

# 2. Processamento de dados (List Comprehension)
# Criamos uma lista de quadrados para números de 0 a 5
valores = [quadrado(x) for x in range(6)]

# 3. Exibição formatada usando strings de múltiplas linhas
print(f"""
Resultados obtidos:
-------------------
A lista gerada foi: {valores}
O primeiro elemento é: {valores[0]}
O último elemento é: {valores[-1]}
""")
```

```
Resultados obtidos:
-------------------
A lista gerada foi: [0, 1, 4, 9, 16, 25]
O primeiro elemento é: 0
O último elemento é: 25
```

1.6.4 🔹 De Números a Leis da Natureza: O Salto para o Conhecimento

A **Figura 1.4** ilustra mais do que uma simples função matemática; ela representa a base para a **modelagem preditiva**. Na Mineração de Dados, a extração de conhecimento ocorre quando o sistema identifica que a relação entre os dados segue um padrão constante.

Se esses dados representassem a posição de um objeto em queda livre ao longo do tempo, o "conhecimento" extraído seria a descoberta da aceleração constante da gravidade. Com essa regra em mãos, o sistema deixa de ser um calculador de trajetórias passadas e torna-se um **agente preditivo**, capaz de antecipar onde o objeto estará em um futuro ainda não registrado nos dados.

```
# Visualização de dados usando Matplotlib
import matplotlib.pyplot as plt

# 1. DADOS (Fatos Brutos): Segundos decorridos
tempo = list(range(6))

# 2. INFORMAÇÃO (Contexto): Distância percorrida (d = t²)
distancia = [t**2 for t in tempo]

# 3. VISUALIZAÇÃO (Pós-processamento do KDD)
plt.figure(figsize=(6, 4))
plt.plot(tempo, distancia, marker='o', linestyle='-', color='g')
plt.title("Simulação de Aceleração (Conhecimento Preditivo)")
plt.xlabel("Tempo (segundos)")
plt.ylabel("Distância (metros)")
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

```
# 4. CONHECIMENTO (Inferência)
print(f"""
ANÁLISE DE MINERAÇÃO:
=====================
Padrão Identificado: Crescimento Quadrático.
Conhecimento Extraído: O sistema detectou uma aceleração constante.
Capacidade Preditiva: Para t=10, a distância será de 100m (sem precisar medir).
""")
```

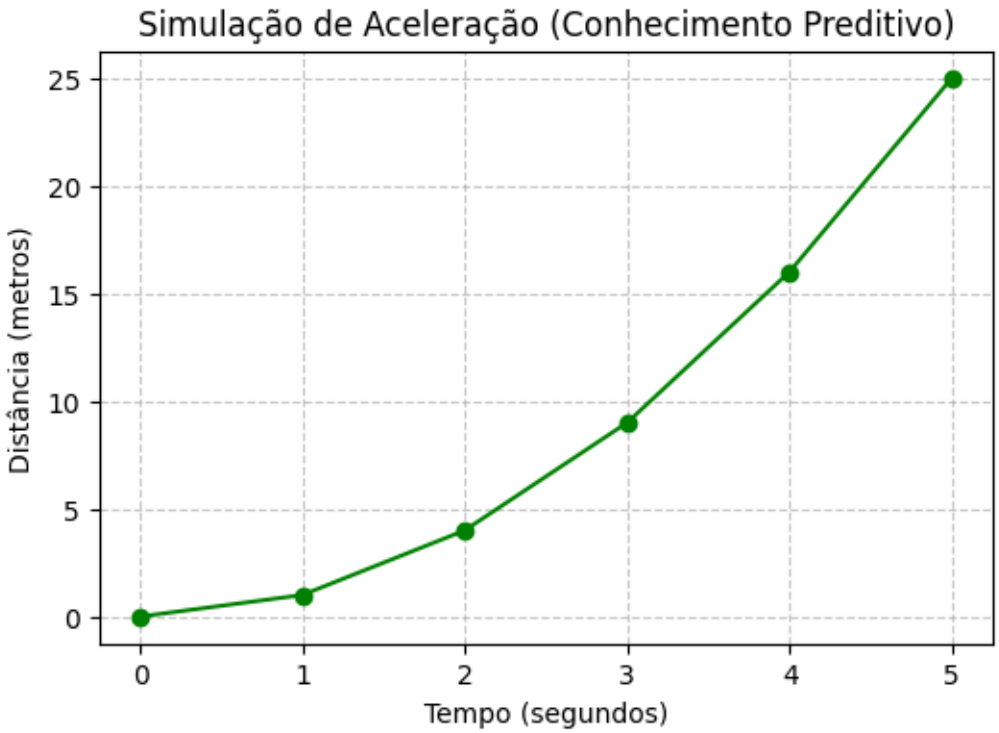


Figura 1.4: Representação da aceleração: Onde os dados revelam leis físicas.

```
ANÁLISE DE MINERAÇÃO:
=====================
Padrão Identificado: Crescimento Quadrático.
Conhecimento Extraído: O sistema detectou uma aceleração constante.
Capacidade Preditiva: Para t=10, a distância será de 100m (sem precisar medir).
```

A transição entre os estágios da hierarquia **DIKW** é evidenciada pela interpretação da curva parabólica. Enquanto o processamento das listas gera **Informação**, o reconhecimento da natureza quadrática do fenômeno caracteriza o **Conhecimento**.

No processo de **KDD**, essa visualização permite validar se o padrão descoberto pela **Mineração de Dados** é consistente com a realidade física. Para um aluno iniciante, a motivação reside em perceber que a programação é o meio pelo qual se ensina a máquina a "enxergar" as regras invisíveis que controlam os dados.

1.6.5 🔹 Transformando Dado em Informação (IMC)

A relação entre peso e altura é dada pelo Índice de Massa Corporal (IMC), conforme a Equação 1.1:

$$\text{IMC} = \frac{\text{peso}}{\text{altura}^2} \tag{1.1}$$

Na biblioteca **Pandas**, o **DataFrame** constitui a estrutura de dados central. Trata-se de uma tabela bidimensional — análoga a uma planilha de cálculo ou a uma tabela de banco de dados — na qual os dados são organizados em **linhas** e **colunas**.

No exemplo apresentado na Figura 1.5, o código a seguir executa as seguintes operações:

- `pd.DataFrame(...)`: O construtor da biblioteca Pandas é invocado para a criação de uma nova tabela.
- `dados_pacientes`: O dicionário contendo nomes, pesos e alturas é utilizado como a fonte de dados primária.
- **Mapeamento**: As **chaves** do dicionário (`'Paciente'`, `'Peso_kg'`, `'Altura_m'`) são automaticamente convertidas em **cabeçalhos de colunas**, enquanto as **listas de valores** passam a compor as **linhas** da tabela.
- `df`: A tabela resultante é armazenada na variável `df` (abreviação convencional para *DataFrame*), permitindo a execução de cálculos complexos e filtragens de alto desempenho.

```
import pandas as pd
import matplotlib.pyplot as plt

"""
1. DADOS BRUTOS (Data):
Fatos registrados sem contexto. Isolados, não permitem tomada de decisão.
"""
dados_pacientes = {
    'Paciente': ['Ana', 'Bruno', 'Carla', 'David'],
    'Peso_kg': [55, 95, 68, 82],
    'Altura_m': [1.65, 1.80, 1.70, 1.75]
}

df = pd.DataFrame(dados_pacientes)

"""
2. TRANSFORMAÇÃO EM INFORMAÇÃO (Information):
O cálculo do IMC contextualiza peso e altura, conferindo significado aos dados.
Uma nova coluna 'IMC' é criada, permitindo comparações e análises posteriores.
"""
df['IMC'] = (df['Peso_kg'] / (df['Altura_m'] ** 2)).round(2)

# 3. RUMO AO CONHECIMENTO (Knowledge):
# Aplicamos uma regra de negócio (Classificação) para extrair padrões.
def classificar_saude(imc):
    if imc < 18.5: return 'Abaixo do peso'
    elif imc < 25: return 'Normal'
    else: return 'Sobrepeso'

# Criamos uma nova coluna 'Status' com a classificação de saúde baseada no IMC.
df['Status'] = df['IMC'].apply(classificar_saude)

print("=== Tabela Processada (Informação) ===")
print(df.round(2)) # Apenas duas casas decimais para melhor visualização

# 4. VISUALIZAÇÃO PARA TOMADA DE DECISÃO (Inteligência/Ação)
plt.figure(figsize=(7, 4))
cores = ['skyblue' if s == 'Normal' else 'salmon' for s in df['Status']]
plt.bar(df['Paciente'], df['IMC'], color=cores)
plt.axhline(y=25, color='red', linestyle='--', label='Limite Normalidade (OMS)')

plt.title('Análise de Saúde dos Pacientes')
plt.ylabel('Índice de Massa Corporal (IMC)')
plt.legend()
plt.show()
```

```
print(f"""
RELATÓRIO DE EXECUÇÃO:
======================
- Média de IMC: {df['IMC'].mean():.2f}
- Casos de Atenção: {len(df[df['Status'] != 'Normal'])}
- Total de Pacientes: {len(df)}
""")
```

```
=== Tabela Processada (Informação) ===
  Paciente  Peso_kg  Altura_m    IMC     Status
0      Ana       55      1.65  20.20     Normal
1    Bruno       95      1.80  29.32  Sobrepeso
2    Carla       68      1.70  23.53     Normal
3    David       82      1.75  26.78  Sobrepeso
```

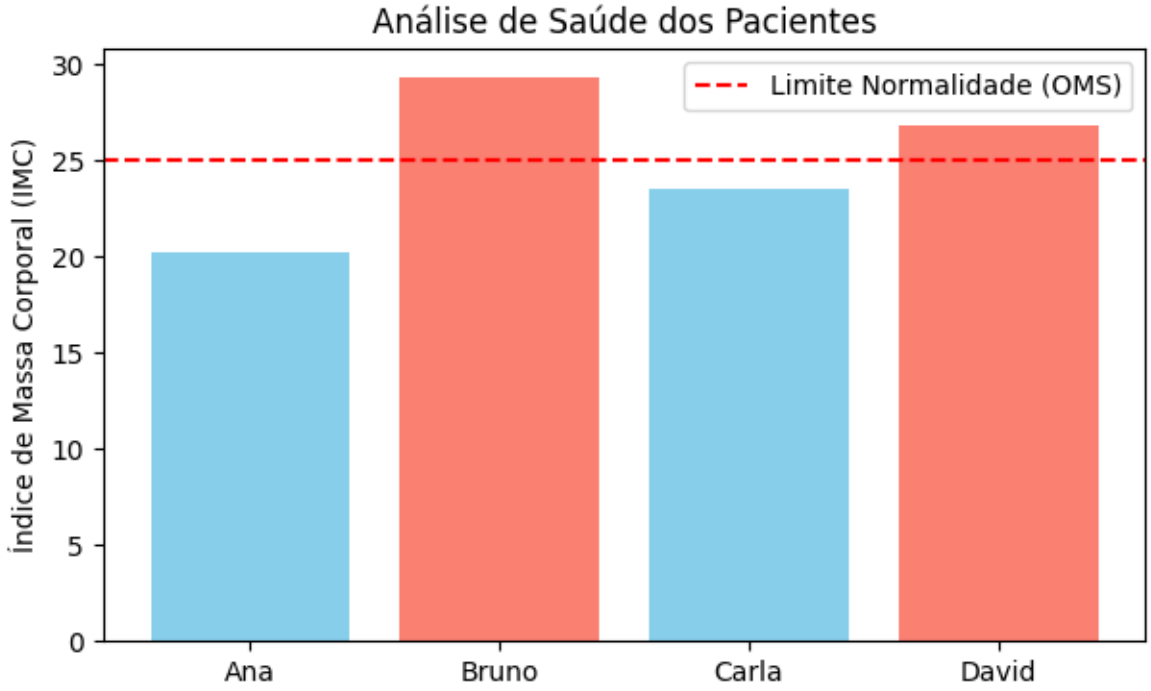


Figura 1.5: Análise de IMC e classificação de saúde dos pacientes.

```
RELATÓRIO DE EXECUÇÃO:
======================
- Média de IMC: 24.96
- Casos de Atenção: 2
- Total de Pacientes: 4
```

A utilização do comando `pd.DataFrame(dados_pacientes)` representa uma etapa de **estruturação e organização dos dados**, convertendo listas ou dicionários em uma estrutura tabular que permite operações analíticas.

Esse processo pode ser interpretado à luz da hierarquia **DIKW** (Dado, Informação, Conhecimento e Sabedoria). Os **dados brutos** (peso, altura) são organizados e processados para gerar **informação**, como o cálculo do IMC. Quando aplicamos uma técnica de **classificação** (por exemplo, para rotular o estado nutricional), produzimos um modelo ou regra interpretável — aproximando-nos do **conhecimento**. A **sabedoria** emerge quando esse conhecimento é utilizado de forma contextualizada e ética na **tomada de decisão**, como na avaliação clínica realizada por um profissional de saúde.

1.7 🤖 Uso do NotebookLM como Tutor Complementar

Nesta segunda edição, além dos notebooks interativos no Google Colab, incentiva-se o uso do **NotebookLM** como ferramenta complementar de aprendizagem. Essa ferramenta de IA utiliza exclusivamente os documentos fornecidos pelos autores como base de conhecimento, garantindo respostas alinhadas ao conteúdo do livro.

Para cada capítulo, foi preparado um projeto específico na plataforma. Para uma experiência de estudo ampliada, utilize o acesso abaixo:

🎓 Estude com o Tutor Inteligente

Para interagir com o conteúdo deste capítulo, acesse o link abaixo. O ambiente contém materiais didáticos em diferentes formatos, gerados a partir do **PDF** do capítulo. Na plataforma, explore especialmente as opções **Guia de Estudo** e **Conversa** para aprofundar sua compreensão.
🚀 ACESSAR NOTEBOOKLM: CAPÍTULO 01

Funcionalidades Disponíveis na Plataforma

O **NotebookLM** oferece uma suíte avançada de ferramentas baseadas em IA para transformar o conteúdo estático do livro em uma experiência de aprendizado dinâmica e multimídia. Conforme ilustrado na Figura 1.6, a plataforma utiliza técnicas de **RAG** (*Retrieval-Augmented Generation*), fundamentadas no trabalho de **lewis2020retrieval**, para basear as respostas estritamente nos documentos fornecidos e minimizar a ocorrência de alucinações.

As principais funcionalidades incluem:

- **Resumos Multimodais (Áudio e Vídeo)**: Geração de conversas naturais entre especialistas no formato de **Resumo em Áudio** (estilo podcast) e **Resumo em Vídeo**, discutindo os temas centrais do capítulo.
- **Visualização de Estruturas (Mapa Mental e Infográfico)**: Criação automática de diagramas que conectam visualmente os conceitos, como as etapas do processo KDD ou a pirâmide DIKW.
- **Ferramentas de Avaliação (Teste e Cartões Didáticos)**: Geração de **Testes** de múltipla escolha e **Cartões Didáticos** (*flashcards*) para fixação de conhecimento, baseados no texto autoral.
- **Apoio à Apresentação (Slides e Relatórios)**: Auxílio na estruturação de **Apresentações de Slides** e na redação de **Relatórios** técnicos e **Briefings**.
- **Análise de Dados (Tabela de Dados)**: Organização de dados extraídos do texto em tabelas estruturadas, auxiliando na compreensão de exemplos práticos como o cálculo do IMC.
- **Chat Contextualizado**: Permite o questionamento direto sobre o código e a teoria, como: *"Onde o dado bruto se transforma em informação no script do IMC?"*.

Figura 1.6: Estúdo do NotebooLM.

💡 Reflita e Pratique — Desafio de Classificação

Pergunta: Imagine que uma instituição bancária precisa decidir se um cliente adimplirá (pagará) ou não um empréstimo. Você saberia dizer qual tarefa de mineração de dados está sendo executada?

Resposta: Trata-se de uma tarefa de **Classificação** (um tipo de Tarefa Preditiva). Como o objetivo é prever um rótulo categórico discreto (*adimplente* ou *inadimplente*), utilizamos modelos de classificação. Se o objetivo fosse prever um valor numérico exato (como o valor da parcela), a tarefa seria de **Regressão**.

1.8 Lista de Exercícios

1. (20%) Defina com suas próprias palavras e exemplos o que é **Dado**, **Informação** e **Conhecimento**.
2. (30%) Considerando a **definição meramente operacional** de que **aprender é mudar o comportamento com base em sua própria experiência de forma a melhorar o desempenho futuro**, justifique se um sapato amaciado pode ter aprendido alguma coisa.
3. (20%) Qual a diferença entre **Mineração de Dados** e **Recuperação de Dados *(Data Retrieval)***?
4. (30%) Explique com suas próprias palavras as **principais tarefas da Mineração de Dados**.

PS1 – Como pode ser visto, não há uma "resposta correta" para as perguntas formuladas. Seu trabalho vai ser avaliado considerando muito mais a coerência da argumentação, e a riqueza dos exemplos, do que a concordância ou não com os conceitos passados. Procure ser conciso, porque normalmente textos prolixos só dificultam a compreensão do argumento, mas evite a linguagem telegráfica, porque você precisa deixar bem claro seu ponto de vista.

PS2 – Há um ditado popular, segundo o qual "uma figura fala mais do que mil palavras". Nem sempre este ditado é verdadeiro: por exemplo, tente expressar este mesmo ditado com uma figura! Mas, em inúmeras situações ele é muito útil. Se você achar que um desenho e/ou uma figura pode(m) ajudar suas respostas, lance mão desse recurso!

1.9 Referências do Capítulo

Este capítulo baseou-se principalmente em obras de referência clássicas da área: **forouzan2011** para fundamentos de computação; **goldschmidt2005**, **han2008**, **tan2009** e **witten2005** para mineração de dados; **padhy2010** e **rezende2005** para sistemas inteligentes; **russell2004** para inteligência artificial; **pinheiro2008** para inteligência analítica e descoberta de conhecimento; e **lewis2020retrieval** para geração aumentada por recuperação (RAG).

Mineração de Dados e Regras de Associação

2.1 Introdução

A Mineração de Dados é uma disciplina tão vasta que qualquer publicação sobre o tema obriga o autor a selecionar alguns tópicos em detrimento de outros não menos importantes. A atividade de **Regras de Associação** foi o tópico escolhido para iniciarmos a apresentação das principais atividades da Mineração de Dados por envolver ideias bem intuitivas. A analogia entre Regras de Associação e Cesta de Compras facilita o entendimento de como descobrir padrões de associação entre itens de um conjunto qualquer.

2.2 Mineração de Dados

Durante o processo de **Descoberta de Conhecimento em Bases de Dados**, **KDD**, é na etapa de **Mineração de Dados** que efetivamente são encontrados os padrões de associação implícitos nos dados. A análise automatizada dessa massa de dados visa detectar regularidades, ou quebra de regularidade, que constitui informação implícita, porém desconhecida, e útil para determinado fim.

A Mineração de Dados pode ser vista como a sistematização de teorias, técnicas e algoritmos desenvolvidos em outras disciplinas já consagradas, como a Estatística, a Inteligência Artificial, o Aprendizado de Máquina, a Base de Dados etc. O propósito da Mineração de Dados é detectar automaticamente padrões de associação úteis e não óbvios em grandes quantidades de dados, veja Figura 2.1.

No dia a dia, uma quantidade incalculável de dados é gerada na forma de registros de vendas, textos brutos, imagens, sons, gráficos etc., tanto por sistemas computacionais como por seres humanos, constituindo uma espécie de informação não estruturada. Embora esta forma de registro de dados seja adequada para o ser humano, quando se trata de analisar grandes quantidades de dados de forma automatizada, é comum e conveniente que se introduza alguma **estrutura** que facilite o acesso e o processamento sistemático.

Para que parte de toda essa informação não estruturada possa ser utilizada na Mineração de Dados, geralmente é feita uma seleção e um pré-processamento visando transformar dados brutos em coleções estruturadas de dados. Em termos práticos, para nossas considerações iniciais, a estrutura de representação de uma Base de Dados pode ser semelhante a uma tabela de dados, sendo cada linha dessa tabela uma **transação** ou um **exemplo**. Cada **transação** é composta por um ou mais **itens** ou, visto de outra forma, cada **exemplo** é caracterizado por seus **atributos**.

As Tabelas 2.1, 2.2 e 2.3 ilustram formas de dados estruturados convenientes para a Mineração de Dados.

Tabela 2.1: Cestas de Compras.

TID	Itens
1	{Arroz, Feijão, Óleo}
2	{Queijo, Vinho}

TID	Itens
3	{Arroz, Feijão, Batata, Óleo}
4	{Arroz, Água, Queijo, Vinho}
5	{Arroz, Feijão, Batata, Óleo}

Na Tabela 2.1 cada uma das Transações possui uma IDentificação (TID), e seus itens representam artigos vendidos em um supermercado. Se a tabela de itens for muito extensa, como costuma ser em casos reais, pode ser ainda mais conveniente representar cada um de seus itens na forma de um atributo associado a um valor booleano, como mostra a Tabela 2.2.

Tabela 2.2: Representação Booleana de Cestas de Compras.

TID	Arroz	Feijão	Batata	Óleo	Água	Queijo	Vinho
1	y	y	n	y	n	n	n
2	n	n	n	n	n	y	y
3	y	y	y	y	n	n	n
4	y	n	n	n	y	y	y
5	y	y	y	y	n	n	n

Um exemplo clássico de uma Base de Dados usada em artigos sobre Mineração de Dados é apresentada na Tabela 2.3 (**quinlan1986** *apud* **witten2005**), composta por dados fictícios sobre as condições de tempo para que ocorra ou não a partida de um esporte não especificado. A tabela é composta por 14 exemplos (linhas), cada um com cinco atributos (colunas): Dia, Temperatura, Umidade, Vento e Partida. A tabela pode também ser interpretada de outra forma, como sendo composta por quatro atributos (Dia, Temperatura, Umidade e Vento) e uma classe (Partida), que representa o resultado da combinação dos quatro atributos.

Tabela 2.3: Tabela do Tempo.

Dia	Temperatura	Umidade	Vento	Partida
Ensolarado	Elevada	Alta	Falso	Não
Ensolarado	Elevada	Alta	Verdadeiro	Não
Nublado	Elevada	Alta	Falso	Sim
Chuvoso	Amena	Alta	Falso	Sim
Chuvoso	Baixa	Normal	Falso	Sim
Chuvoso	Baixa	Normal	Verdadeiro	Não
Nublado	Baixa	Normal	Verdadeiro	Sim
Ensolarado	Amena	Alta	Falso	Não
Ensolarado	Baixa	Normal	Falso	Sim
Chuvoso	Amena	Normal	Falso	Sim
Ensolarado	Amena	Normal	Verdadeiro	Sim
Nublado	Amena	Alta	Verdadeiro	Sim
Nublado	Elevada	Normal	Falso	Sim
Chuvoso	Amena	Alta	Verdadeiro	Não

Uma análise mais atenta dos exemplos das Tabelas 2.1 e 2.3 revela que alguns desses atributos sempre aparecem juntos e que, portanto, várias **Regras de Associação** podem ser extraídas dessas tabelas.

Figura 2.1: A Relação da Mineração de Dados com Algumas Disciplinas Correlatas.

2.3 Regras de Associação

A representação do conhecimento através de regras, também conhecidas como regras *IF-THEN* ou Regras de Produção, é largamente utilizada porque, entre outras vantagens sobre formas alternativas de representação do conhecimento, regras são facilmente compreendidas pelo ser humano, fáceis de serem alteradas, validadas e verificadas, e de baixo custo para a criação de sistemas baseados em regras (**padhy2010**).

Regras de Associação são uma forma específica de representação de conhecimento que descrevem padrões de associação implícitos entre um conjunto de atributos ou itens de uma Base de Dados, e que podem ajudar a predizer com alta probabilidade a presença, ou não, de outro conjunto de atributos ou itens.

Dito de forma equivalente, uma Regra de Associação revela que a presença de um conjunto $\mathbf{X}$ de itens numa transação implica outro conjunto $\mathbf{Y}$ de itens, i.e., $\mathbf{X} = \{a, b, ...\} \Rightarrow \mathbf{Y} = \{p, \ldots, z\}$. Note que o fato de um conjunto de itens $\mathbf{X}$ (antecedente) estar sempre associado a outro $\mathbf{Y}$ (consequente) não significa obrigatoriamente que um seja a causa de outro, i.e., não há necessariamente relação de causalidade entre antecedente e consequente e sim mera ocorrência simultânea de itens com certa probabilidade.

A estrutura geral de uma **Regra de Associação** assume a seguinte forma:

$$\textbf{If}\ (\text{Conjunto } \mathbf{X} \text{ de Itens})\ \ \textbf{then}\ \ (\text{Conjunto } \mathbf{Y} \text{ de Itens}), \quad \text{sendo } \mathbf{X} \cap \mathbf{Y} = \emptyset$$

Com base na Tabela 2.3, várias Regras de Associação podem ser formuladas:

$$\textbf{If}\ (\text{Temperatura} = \text{Baixa})\ \ \textbf{then}\ \ (\text{Umidade} = \text{Normal}) \tag{2.1}$$

$$\textbf{If}\ (\text{Umidade} = \text{Normal})\ \ \textbf{and}\ \ (\text{Vento} = \text{Falso})\ \ \textbf{then}\ \ (\text{Partida} = \text{Sim}) \tag{2.2}$$

$$\textbf{If}\ (\text{Dia} = \text{Ensolarado})\ \ \textbf{and}\ \ (\text{Partida} = \text{Não})\ \ \textbf{then}\ \ (\text{Umidade} = \text{Alta}) \tag{2.3}$$

$$\textbf{If}\ (\text{Vento} = \text{Falso})\ \ \textbf{and}\ \ (\text{Partida} = \text{Não})\ \ \textbf{then}\ \ (\text{Temperatura} = \text{Elevada})\ \ \textbf{and}\ \ (\text{Umidade} = \text{Alta}) \tag{2.4}$$

Estas são apenas algumas das muitas Regras de Associação que podem ser formuladas com base na Tabela 2.3. Para selecionar as Regras de Associação mais representativas, i.e., aquelas que se apliquem a um grande número de exemplos com alta probabilidade de acerto, precisaremos de métricas para avaliar o alcance ou a força de cada regra. Dois dos mais conhecidos indicadores são **Suporte** e **Confiança**.

Suporte — para cada regra do tipo $\mathbf{X} \Rightarrow \mathbf{Y}$, este parâmetro indica a quantos exemplos da tabela esta regra satisfaz (i.e., contém) tanto ao conjunto de itens de $\mathbf{X}$ quanto ao de $\mathbf{Y}$, ou seja, indica sua **cobertura** com relação ao número total N de exemplos da tabela. Portanto,

$$\textbf{Sup}\,(\mathbf{X} \rightarrow \mathbf{Y}) = \frac{|\mathbf{X} \cup \mathbf{Y}|}{\mathbf{N}}$$

Por exemplo, com relação à primeira Regra 2.1, há quatro exemplos na Tabela 2.3 em que $\mathbf{X} \cup \mathbf{Y} = \{\text{Temperatura} = \text{Baixa},\ \text{Umidade} = \text{Normal}\}$. Portanto,

$$\textbf{Sup}\,(\text{Regra 2.1}) = \frac{|X \cup Y|}{N} = \frac{|\{\text{Temperatura} = \text{Baixa, Umidade} = \text{Normal}\}|}{N} = \frac{4}{14} = 0{,}29$$

A Regra 2.2 também possui **Sup**(Regra 2.2) $= \frac{4}{14}$; a terceira regra apresenta **Sup**(Regra 2.3) $= \frac{3}{14}$, enquanto a quarta regra possui **Sup**(Regra 2.4) $= \frac{1}{14}$.

Confiança — a confiança de uma regra reflete o número de exemplos que contêm $\mathbf{Y}$ dentre todos aqueles que contêm $\mathbf{X}$ (veja bem, além de $\mathbf{X} \Rightarrow \mathbf{Y}$, podem existir regras do tipo $\mathbf{X} \Rightarrow \mathbf{Z}$, $\mathbf{X} \Rightarrow \mathbf{W}$ etc.). Em outras palavras, o

parâmetro Confiança determina quantos são os exemplos em que $\mathbf{X}$ implica $\mathbf{Y}$, comparado com aqueles exemplos em que $\mathbf{X}$ pode ou não implicar $\mathbf{Y}$. A este parâmetro costuma-se também dar o nome de **Acurácia**.

$$\textbf{Conf}(\mathbf{X} \Rightarrow \mathbf{Y}) = \frac{|\mathbf{X} \cup \mathbf{Y}|}{|\mathbf{X}|} = \frac{\textbf{Sup}(\mathbf{X} \Rightarrow \mathbf{Y})}{\textbf{Sup}(\mathbf{X})}$$

Por exemplo, com relação à primeira Regra 2.1, há quatro exemplos na Tabela 2.3 em que $\mathbf{X} \cup \mathbf{Y} = \{\text{Temperatura} = \text{Baixa},\ \text{Umidade} = \text{Normal}\}$ e, coincidentemente, quatro exemplos em que $\mathbf{X} = \{\text{Temperatura} = \text{Baixa}\}$. Portanto,

$$\textbf{Conf}(\text{Regra 2.1}) = \frac{|\mathbf{X} \cup \mathbf{Y}|}{|\mathbf{X}|} = \frac{|\{\text{Temperatura} = \text{Baixa},\ \text{Umidade} = \text{Normal}\}|}{|\{\text{Temperatura} = \text{Baixa}\}|} = \frac{4}{4} = 1{,}0$$

A Regra 2.2 também possui $\textbf{Conf}(\text{Regra 2.2}) = \frac{4}{4}$; a terceira regra apresenta $\textbf{Conf}(\text{Regra 2.3}) = \frac{3}{3}$, enquanto a quarta regra possui $\textbf{Conf}(\text{Regra 2.4}) = \frac{1}{2}$.

> **i** Nota sobre notação: $|\cdot|$
>
> Seguindo a convenção de **agrawal1993mining** e **han2008**, o símbolo $|\cdot|$ nessas fórmulas **não** representa a cardinalidade matemática do conjunto (número de itens), mas sim a **frequência absoluta** — ou seja, o número de transações da base de dados que contêm aquele conjunto de itens. Assim, $|\mathbf{X}|$ significa "quantas transações contêm todos os itens de $\mathbf{X}$", equivalente a $\textbf{Sup}(\mathbf{X}) \times N$.

Regras de Associação são particularmente úteis para analisar o comportamento de clientes e propor "vendas casadas". A informação de que clientes que compram o item A geralmente compram o item B pode aumentar significativamente as vendas de uma loja ou livraria, já que toda vez que um cliente manifestar a intenção de comprar o item A, a loja pode também lhe oferecer o item B.

Mas o fato de um simples conjunto de itens poder gerar muitas regras de associação faz com que o número de regras associadas a uma base de dados seja tão grande a ponto de a maioria dessas regras não ter qualquer interesse prático. Para contornar esta situação, antes de começar a gerar as regras de associação, é comum que sejam estabelecidos um valor de Suporte Mínimo (**SupMin**) e de Confiança Mínima (**ConfMin**). Regras com suporte muito baixo podem ser resultado de compras feitas ao acaso e, portanto, não fornecem informações de interesse. Por outro lado, regras com confiança muito baixa podem indicar que seu poder de predição é baixo e, portanto, não é muito aconselhável assumir que $\mathbf{X}$ implica $\mathbf{Y}$ com base nessas regras.

agrawal1993mining ao introduzir o conceito de Regras de Associação propôs um algoritmo denominado **Apriori** no qual Regras de Associação são geradas em duas etapas:

- Dado um conjunto de transações $\mathbf{T}$, primeiramente são criados conjuntos de itens frequentes, chamados de **Conjuntos Frequentes**, que devem satisfazer o limite de **SupMin**;
- a partir desses Conjuntos Frequentes são geradas **Regras de Associação** com confiança maior ou igual **ConfMin**.

2.4 Etapa 1: Geração de Conjuntos Frequentes com Suporte $\geq$ SupMin

As Tabelas 2.4 e 2.5 mostram versões simplificadas da Tabela 2.2, aqui adaptada para que cada item possa ser representado por apenas uma letra.

De acordo com o algoritmo Apriori, para se obter os possíveis Conjuntos Frequentes relacionados a um conjunto de transações, inicialmente devem ser criados Conjuntos Frequentes com 1 item apenas e que satisfaçam o critério de Suporte Mínimo. A seguir são criados recursivamente Conjuntos Frequentes com 2 itens, depois com 3 itens, e assim sucessivamente.

Os possíveis Conjuntos Frequentes com 1 item apenas, e seus respectivos valores de Suporte, estão representados na Tabela 2.6.

Tabela 2.4: Versão simplificada da Tabela 2.2.

TID	A	B	C	D	E	F	G
1	1	1	0	1	0	0	0
2	0	0	0	0	0	1	1
3	1	1	1	1	0	0	0
4	1	0	0	0	1	1	1
5	1	1	1	1	0	0	0

Tabela 2.5: Versão alternativa da Tabela 2.2.

TID	Itens
1	{A, B, D}
2	{F, G}
3	{A, B, C, D}
4	{A, E, F, G}
5	{A, B, C, D}

Tabela 2.6: Possíveis Conjuntos Frequentes com 1 Item.

Itens	Suporte
{A}	4/5
{B}	3/5
{C}	2/5
{D}	3/5
{E}	1/5
{F}	2/5
{G}	2/5

Suponhamos que o **SupMin** tenha sido definido como 2/5, ou seja, 40%. De acordo com este critério, o conjunto {E} não satisfaz **SupMin** e deve ser eliminado. Portanto os Conjuntos Frequentes com 1 Item que satisfazem o critério de **SupMin** maior ou igual a 2/5 estão representados na Tabela 2.7.

Ao adotar o procedimento de poda dos candidatos a Conjunto Frequente que não satisfazem o critério de **SupMin**, o número total de Conjuntos Frequentes gerados pode cair significativamente. Em princípio, dada uma Base de Dados com k itens, o número de possíveis Conjuntos Frequentes é $|\mathbf{CF}| = 2^k - 1$ (excluindo o conjunto vazio). Como em nossa Tabela 2.4 há 7 itens, então $|\mathbf{CF}| = 2^7 - 1 = 127$.

Tabela 2.7: Conjuntos Frequentes com 1 Item e **SupMin** $\geq \frac{2}{5}$.

Itens	Suporte
{A}	4/5
{B}	3/5
{C}	2/5
{D}	3/5
{F}	2/5
{G}	2/5

A seguir devem ser formados novos Conjuntos Frequentes com 2 Itens, partindo-se dos Conjuntos Frequentes com 1 Item. Note que o Suporte de um Conjunto Frequente com 2 Itens pode ter no máximo o menor valor de Suporte de cada um de seus subconjuntos, i. e., dos respectivos Conjuntos Frequentes com 1 Item. De acordo com esta mesma propriedade, conhecida como **Princípio Apriori** ou antimonotônico, qualquer subconjunto de um Conjunto Frequente também será um Conjunto Frequente. Por exemplo, se $\{A, B\}$ for um Conjunto Frequente, então $\{A\}$ e $\{B\}$ também são Conjuntos Frequentes e têm **Suporte** $\geq$ **SupMin**.

A Tabela 2.8 mostra os possíveis Conjuntos Frequentes com 2 Itens e os respectivos valores de Suporte.

Os conjuntos de 2 itens foram obtidos por combinação dos conjuntos de 1 item, enquanto que os valores de Suporte foram obtidos inspecionando-se as Tabelas 2.4 e 2.5.

Note que os detalhes de como os conjuntos de itens são efetivamente gerados dependem da forma como o algoritmo Apriori foi implementado, e diferem um pouco da exposição simplificada que se adotou aqui por razões didáticas.

Para calcular o Suporte dos Conjuntos Frequentes foi necessário ler a Base de Dados, que em nosso caso é pequena e está representada pela Tabela 2.4. Em uma implementação computacional é altamente desejável que toda a Base de Dados possa ser lida na memória principal do computador. Porém, se a Base de Dados for muito grande ela provavelmente terá de ser lida no disco rígido. Para minimizar o número de vezes que a Base de Dados é consultada, muitos candidatos a Conjuntos Frequentes podem ser inicialmente criados e depois eliminados, obtendo-se assim significativos ganhos de tempo.

Tabela 2.8: Possíveis Conjuntos Frequentes com 2 Itens.

Itens	Suporte
{A, B}	3/5
{A, C}	2/5
{A, D}	3/5
{A, F}	1/5
{A, G}	1/5
{B, C}	2/5
{B, D}	3/5
{B, F}	0
{B, G}	0
{C, D}	2/5
{C, F}	0
{C, G}	0
{D, F}	0
{D, G}	0
{F, G}	2/5

Para nós, neste momento, o importante é compreender como os Conjuntos Frequentes com k Itens podem ser gerados de forma relativamente simples pela combinação de Conjuntos Frequentes com $k-1$ Itens.

Aplicando-se novamente o critério de **SupMin** $\geq \frac{2}{5}$, restam apenas os Conjuntos Frequentes com 2 Itens apresentados na Tabela 2.9.

Tabela 2.9: Conjuntos Frequentes com 2 Itens e **SupMin** $\geq \frac{2}{5}$.

Itens	Suporte
{A, B}	3/5
{A, C}	2/5
{A, D}	3/5
{B, C}	2/5
{B, D}	3/5
{C, D}	2/5
{F, G}	2/5

O próximo passo agora consiste em criar novos Conjuntos Frequentes com 3 Itens, partindo-se dos Conjuntos Frequentes com 2 Itens, cujo resultado é mostrado na Tabela 2.10.

Tabela 2.10: Possíveis Conjuntos Frequentes com 3 Itens.

Itens	Suporte
{A, B, C}	2/5
{A, B, D}	3/5
{A, C, D}	2/5
{A, F, G}	1/5
{B, C, D}	2/5
{B, F, G}	0
{C, D, F}	0
{C, F, G}	0

Neste caso, novamente, alguns Conjuntos Frequentes não satisfazem o critério de **SupMin** $\geq \frac{2}{5}$, havendo a necessidade de poda para que esses conjuntos não participem da etapa seguinte.

Tabela 2.11: Conjuntos Frequentes com 3 Itens e **SupMin** $\geq \frac{2}{5}$.

Itens	Suporte
{A, B, C}	2/5
{A, B, D}	3/5
{A, C, D}	2/5
{B, C, D}	2/5

Vamos agora gerar novos Conjuntos Frequentes com 4 Itens, partindo-se dos Conjuntos Frequentes com 3 Itens (Tabela 2.11), cujo resultado é mostrado na Tabela 2.12.

Tabela 2.12: Possíveis Conjuntos Frequentes com 4 Itens.

Itens	Suporte
{A, B, C, D}	2/5

Se houvesse ao menos dois Conjuntos Frequentes com 4 Itens poderíamos ainda tentar gerar Conjuntos Frequentes com 5 Itens. Mas como há apenas um Conjunto Frequente com 4 Itens, esta primeira etapa do algoritmo Apriori termina aqui.

2.5 Etapa 2: Geração de Regra de Associação a partir dos Conjuntos Frequentes

Uma vez obtidos os Conjuntos Frequentes com Suporte $\geq$ **SupMin**, é possível extrair de cada Conjunto Frequente com k itens um total de $2^k - 2$ Regras de Associação (excluindo-se o conjunto vazio na posição de antecedente $\emptyset \Rightarrow$ **CF** ou de consequente **CF** $\Rightarrow \emptyset$). Na Etapa 1 foram gerados os seguintes Conjuntos Frequentes:

Conjuntos Frequentes com 1 Item (total de 6 CFs)

$$\{A\},\ \{B\},\ \{C\},\ \{D\},\ \{F\},\ \{G\}$$

Conjuntos Frequentes com 2 Itens (total de 7 CFs)

$$\{A, B\},\ \{A, C\},\ \{A, D\},\ \{B, C\},\ \{B, D\},\ \{C, D\},\ \{F, G\}$$

Conjuntos Frequentes com 3 Itens (total de 4 CFs)

$$\{A, B, C\},\ \{A, B, D\},\ \{A, C, D\},\ \{B, C, D\}$$

Conjunto Frequente com 4 Itens (total de 1 CF)

$$\{A, B, C, D\}$$

Para extrair as Regras de Associação de um Conjunto Frequente é necessário, primeiramente, gerar todos os subconjuntos não vazios desse Conjunto Frequente **CF** e, para cada subconjunto $\mathbf{S} \subset \mathbf{CF}$, produzir uma Regra de Associação do tipo $\mathbf{S} \Rightarrow (\mathbf{CF} - \mathbf{S})$ que satisfaça o critério de **Confiança** $\geq$ **ConfMin**.

Por exemplo, dado $\mathbf{CF} = \{A, B, C\}$, seus subconjuntos não vazios possíveis são $\mathbf{S} = \{\{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}\}$. Portanto, é possível extrair seis Regras de Associação do $\mathbf{CF} = \{A, B, C\}$ que envolvam os três itens:

$$\{A\} \Rightarrow \{B, C\},$$

$$\{B\} \Rightarrow \{A, C\},$$

$$\{C\} \Rightarrow \{A, B\},$$

$$\{A, B\} \Rightarrow \{C\},$$

$$\{A, C\} \Rightarrow \{B\},$$

$$\{B, C\} \Rightarrow \{A\}.$$

Como o Suporte de todos os subconjuntos já terá sido calculado na Etapa 1, não será necessário percorrer novamente a Base de Dados para calcular a Confiança de cada Regra de Associação. Basta reutilizar esses valores já calculados, pois

$$\mathbf{Conf}(\mathbf{S} \rightarrow \mathbf{CF} - \mathbf{S}) = \frac{|\mathbf{S} \cup (\mathbf{CF} - \mathbf{S})|}{|\mathbf{S}|} = \frac{|\mathbf{CF}|}{|\mathbf{S}|} = \frac{\mathbf{Sup}(\mathbf{CF})}{\mathbf{Sup}(\mathbf{S})}$$

Voltando ao exemplo inicial da Tabela 2.5, como o Suporte de $\mathbf{CF} = \{A, B, C\}$ é $\frac{2}{5}$ (veja Tabela 2.11), e considerando os Suportes de seus subconjuntos,

$\mathbf{Sup}(\{A\}) = \frac{4}{5}$ (veja Tabela 2.6)

$\mathbf{Sup}(\{B\}) = \frac{3}{5}$ (veja Tabela 2.6)

$\mathbf{Sup}(\{C\}) = \frac{2}{5}$ (veja Tabela 2.6)

$\mathbf{Sup}(\{A, B\}) = \frac{3}{5}$ (veja Tabela 2.8)

$\mathbf{Sup}(\{A, C\}) = \frac{2}{5}$ (veja Tabela 2.8)

$\mathbf{Sup}(\{B, C\}) = \frac{2}{5}$ (veja Tabela 2.8)

a Confiança de cada uma das seis regras possíveis será:

$$\mathbf{Conf}(A \Rightarrow B, C) = \frac{\frac{2}{5}}{\frac{4}{5}} = 0{,}50$$

$$\textbf{Conf}(B \Rightarrow A, C) = \frac{\frac{2}{5}}{\frac{3}{5}} = 0{,}66$$

$$\textbf{Conf}(C \Rightarrow A, B) = \frac{\frac{2}{5}}{\frac{2}{5}} = 1{,}00$$

$$\textbf{Conf}(A, B \Rightarrow C) = \frac{\frac{2}{5}}{\frac{3}{5}} = 0{,}66$$

$$\textbf{Conf}(A, C \Rightarrow B) = \frac{\frac{2}{5}}{\frac{2}{5}} = 1{,}00$$

$$\textbf{Conf}(B, C \Rightarrow A) = \frac{\frac{2}{5}}{\frac{2}{5}} = 1{,}00$$

Suponha que, para o problema em questão, tenham sido adotados **SupMin** = 40% e **ConfMin** = 90%. Nesse caso, apenas três das regras acima seriam aproveitadas:

$$\textbf{Conf}(C \Rightarrow A, B) = 1{,}00 \quad \{Batata\} \Rightarrow \{Arroz, Feijo\}$$

$$\textbf{Conf}(A, C \Rightarrow B) = 1{,}00 \quad \{Arroz, Batata\} \Rightarrow \{Feijo\}$$

$$\textbf{Conf}(B, C \Rightarrow A) = 1{,}00 \quad \{Feijo, Batata\} \Rightarrow \{Arroz\}$$

Aplicando-se o procedimento explicado acima para todos os 18 **CFs** obtidos na Etapa 1, seriam geradas aproximadamente 30 Regras de Associação com **SupMin** = 40% e **ConfMin** = 90% (na realidade, chegamos ao número 30 por meio de simulação no Weka, como será mostrado na Atividade Prática com o Weka).

2.6 Como Gerar Regras de Associação Usando a Ferramenta Weka

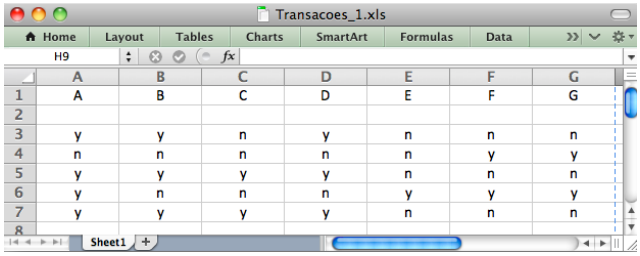
Nesta seção será apresentado um pequeno tutorial sobre a geração de Regras de Associação usando o algoritmo "Apriori" implementado na ferramenta de Aprendizado de Máquina para tarefas de Mineração de Dados Weka (**frank2016weka**). A versão utilizada é a 3.6.7 (**versão 3.8.6 ou 3.9.* ?**). Para fazer uma simulação no Weka, a Base de Dados terá de ser escrita ou no formato CSV *(Comma-Separated Value)* (".csv") ou no formato "ARFF" *(Attribute-Relation File Format)*, um formato bastante simples e intuitivo dessa ferramenta. Com o arquivo ".arff" carregado, podemos ajustar os parâmetros Suporte e Confiança e rodar o algoritmo Apriori.

Passo 1 - Vamos supor que nossa Base de Dados tenha sido retirada de uma planilha eletrônica (".xls") e salva no formato ".csv", como mostra a Figura 2.2.

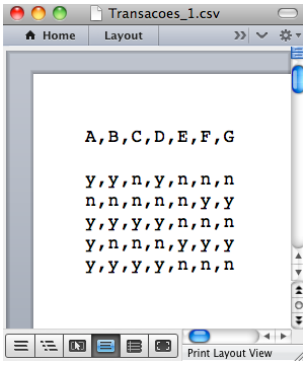
Como o Weka tem um conversor interno do formato ".csv" para ".arff", vamos primeiramente usar este recurso. Depois vamos mostrar como transformar manualmente o arquivo ".csv" para ".arff".

Obs.: Certifique-se que em seu arquivo ".csv" o separador de células seja efetivamente a vírgula "," e não ";". Se o arquivo ".csv" gerado pela sua planilha utilizar ";", faça a substituição para ",". Caso contrário, ocorrerá um erro de leitura no Weka e o arquivo será interpretado de forma completamente diferente do esperado.

A Tabela 2.13 apresenta a representação booleana das cestas de compras, em que cada coluna (A, B, C, ...) indica a presença (`y`) ou ausência (`n`) de um item em cada transação. Esse formato é amplamente utilizado em tarefas de mineração de dados, como a extração de regras de associação.



(a) Planilha ".xls"



(b) ".csv"

Figura 2.2: A Base de Dados Transacoes_1 na Forma (a) Planilha ".xls" e (b) ".csv".

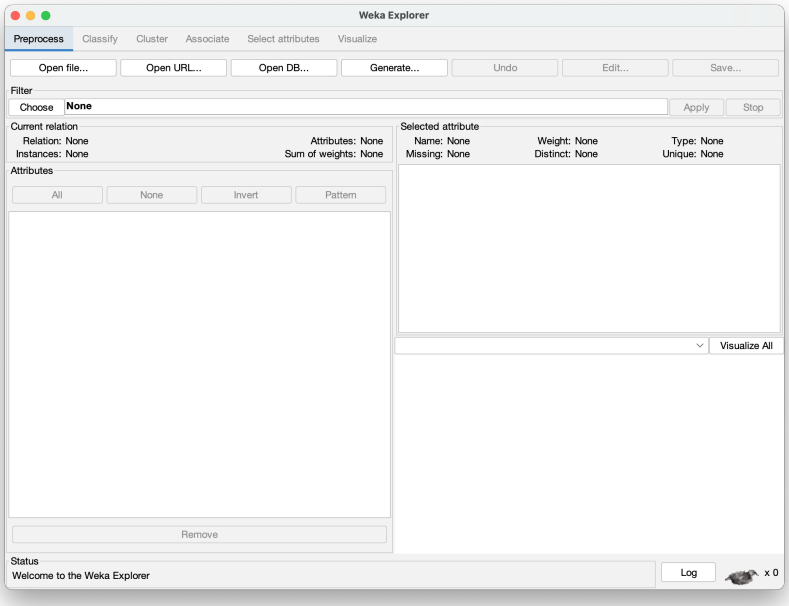
Tabela 2.13: Representação Booleana de Cestas de Compras para utilizar no Weka.

A	B	C	D	E	F	G
y	y	n	y	n	n	n
n	n	n	n	n	y	y
y	y	y	y	n	n	n
y	n	n	n	y	y	y
y	y	y	y	n	n	n

```
<IPython.core.display.HTML object>
```

Passo 2 – Dispare o Weka ("GUI Chooser") e tome a opção "Explorer", que corresponde à versão com recursos gráficos e ícones (em vez de linha de comando). Veja Figura 2.3.

(a) GUI Chooser



(b) Explorer

Figura 2.3: Telas Iniciais do Weka (a) GUI Chooser e (b) Explorer.

Passo 3 – Com a aba superior “Preprocess” escolhida, dê um clique em “Open file...”. Uma janela denominada “Open” deve se abrir. Ajuste a opção de “File Format:” para “.csv”, e escolha o arquivo “Transacoes_1.csv”, conforme mostra a Figura 2.4.

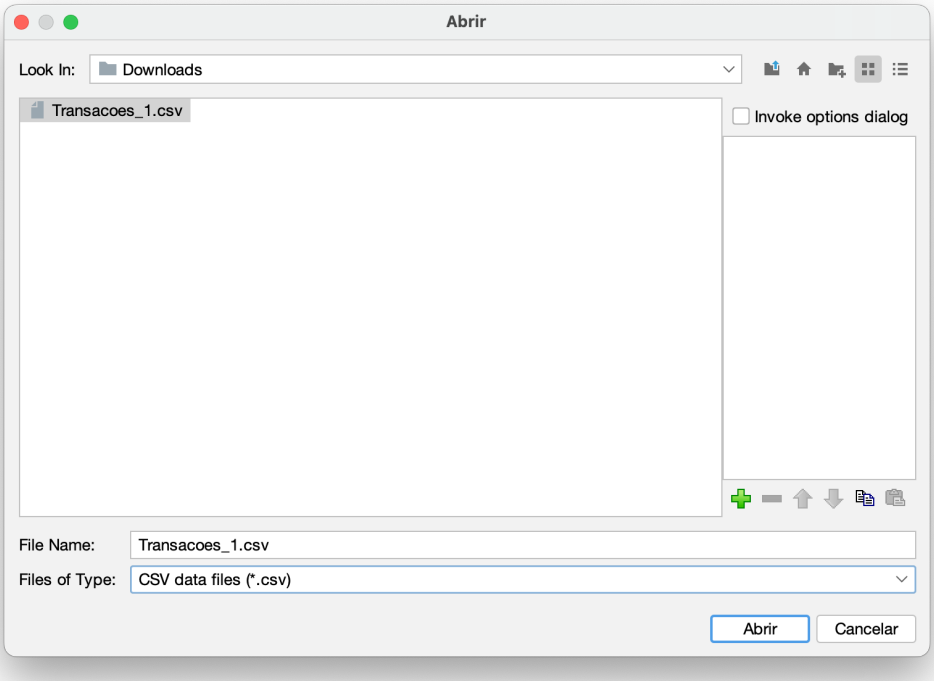


Figura 2.4: Janela *Open* com a Opção *File Format:* em `.csv`.

Passo 4 – A tela do Weka Explorer deve apresentar os sete atributo do arquivo `Transacoes_1`, como mostra a Figura 2.5.

Passo 5 – Como nossa “Base de Dados” é muito pequena, a conversão manual do arquivo “.csv” para “.arff” pode ser feita muito rapidamente.

Digite no arquivo “Transacoes_1.csv” as palavras-chave “@relation”, “@attribute” e “@data”, de acordo com a Figura 2.6, salve e feche o arquivo “.csv”. Mude a terminação do arquivo de “.csv” para “.arff”. Há ainda outras alternativas: Crie um arquivo “Transacoes_1.txt” com o conteúdo mostrado abaixo na Figura 2.6 (certifique-se de que se trata efetivamente de arquivo tipo “.txt” e não, por exemplo, “Transacoes_1.txt.doc” ou “Transacoes_1.txt.rtf”). Feche o arquivo e mude a terminação para “.arff”, ou seja, para “Transacoes_1.arff”.

```
<IPython.core.display.HTML object>
```

Passo 6 – Com o arquivo “Transacoes_1.arff” pronto, disparar o Weka, selecionar a aba “Preprocess”, depois clicar na opção “Open file...” e escolher o arquivo “Transacoes1_.arff”, conforme mostra a Figura 2.7.

Passo 7 – Depois de abrir o arquivo “Transacoes_1.arff”, ainda com a aba “Preprocess” selecionada, escolha “No class” (ao lado de “Visualize all”), conforme ilustra a Figura 2.8. (Como vamos gerar Regras de Associação, qualquer um dos atributos pode funcionar como “classe”. Este conceito vai ser melhor explicado quando formos estudar Regras de Classificação.).

Passo 8 – Na aba superior do Weka, escolher “Associate” e ao lado de “Choose” clicar duas vezes sobre o algoritmo “Apriori”, conforme mostra a Figura 2.9.

Passo 9 – Na janela que se abre, ajustar o **SupMin** (“lowerBoundMinSupport”) para 0.4, a ConfMin (minMetric) para 0.9 e o número de regras mostradas (“numRules”) para 1000, conforme mostra a Figura 2.10. Clicar em “OK”.

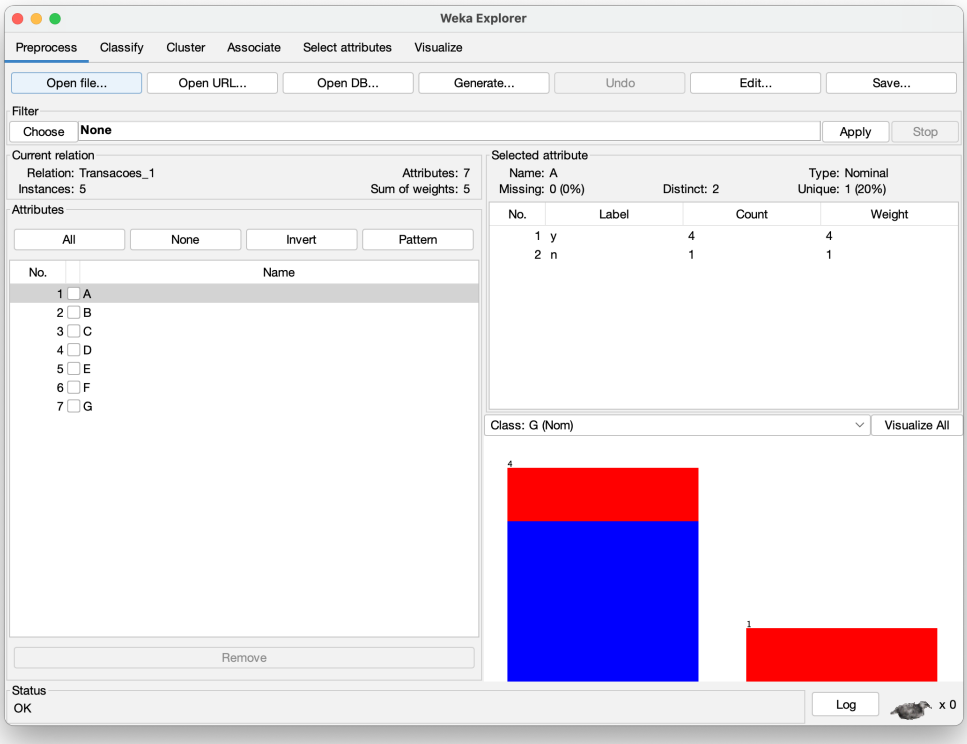


Figura 2.5: Os Sete Atributos do Arquivo `Transacoes_1` São Mostrados.

Passo 10 – Ao clicar em "Start" (ver Figura 2.9) centenas de Regras de Associação serão geradas, a maioria delas sem qualquer interesse, conforme ilustra a Figura 2.11. Um dos riscos da geração de Regras de Associação é que muitas delas podem não ter qualquer significado prático. Para contornar este tipo de problema, é possível introduzir pequenas mudanças na forma como os atributos são declarados e reduzir significativamente o número de regras geradas.

Passo 11 – Uma forma de diminuir o número de regras é substituir os valores ausentes de atributo "n" por "?". Crie um arquivo "Transacoes_2.arff" conforme mostra a Figura 2.12.

```
<IPython.core.display.HTML object>
```

Isso evitará que o Weka crie regras sem qualquer significado prático, envolvendo itens ausentes, como por exemplo $\{F = n\} \Rightarrow \{G = n\}$ (Regra 20 na Figura 2.11). Embora a regra $\{F = y\} \Rightarrow \{G = y\}$ (isto é, "quem compra queijo também costuma comprar vinho") possa ser de interesse, a regra segundo a qual "quem não compra queijo também não compra vinho" dificilmente trará alguma informação prática. Em uma Base de Dados muito grande, regras desse tipo podem aparecer em quantidades proibitivamente elevadas.

Com o arquivo "Transacoes_2.arff" foram geradas 30 Regras de Associação (Figura 2.13), sendo que as regras ilustrativas do texto teórico do Capítulo 2, envolvendo o $\mathbf{CF} = \{A, B, C\}$, aparecem na Figura 2.13 como as regras 15, 16 e 17.

Há outras formas de melhorar a qualidade dos resultados e controlar o número de regras geradas, por exemplo, através do parâmetro **Lift**, cujo significado fica como lição de casa.

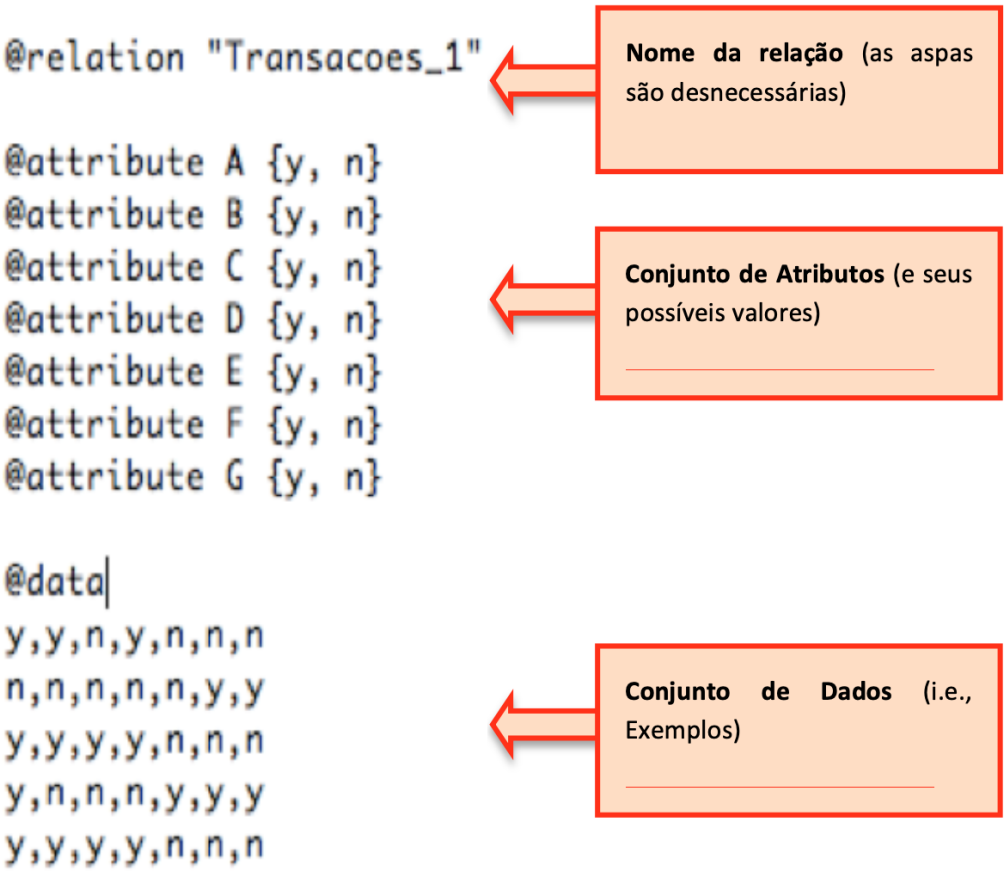


Figura 2.6: Arquivo ARFF (Transacoes_1.arff), com Itens Ausentes Representados por "n".

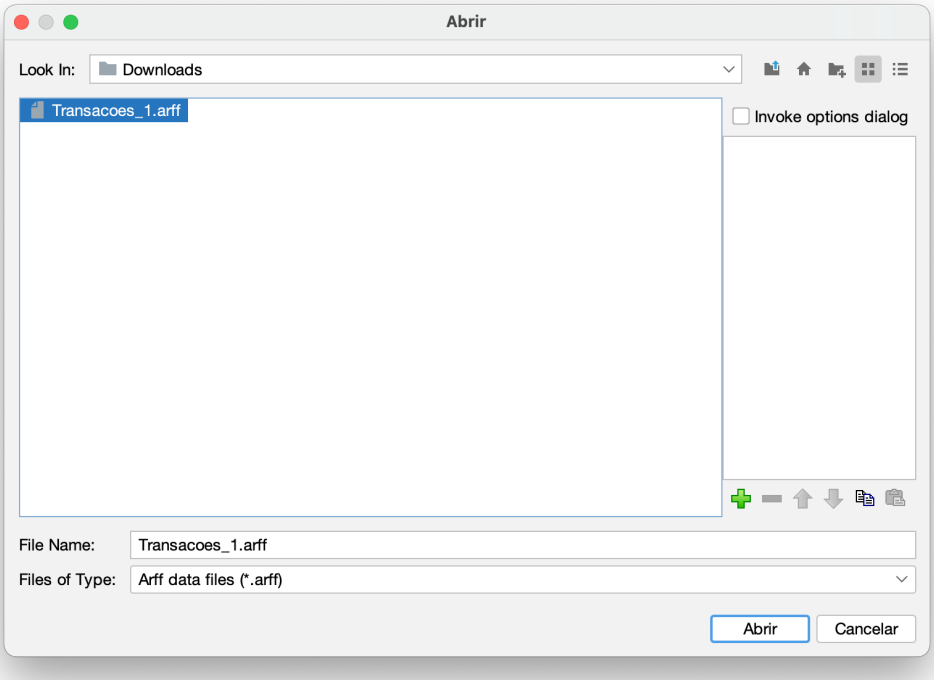


Figura 2.7: Aba "Preprocess" + "Open file..." para Escolha do Arquivo ARFF.

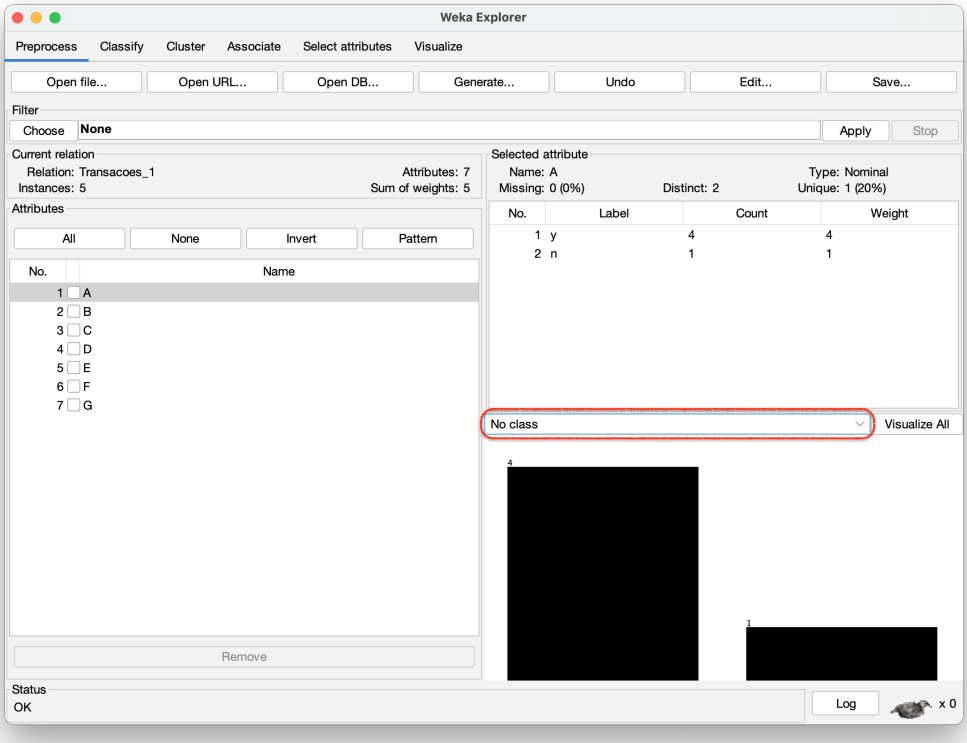


Figura 2.8: Seleção da Opção “No class” para Regras de Associação.

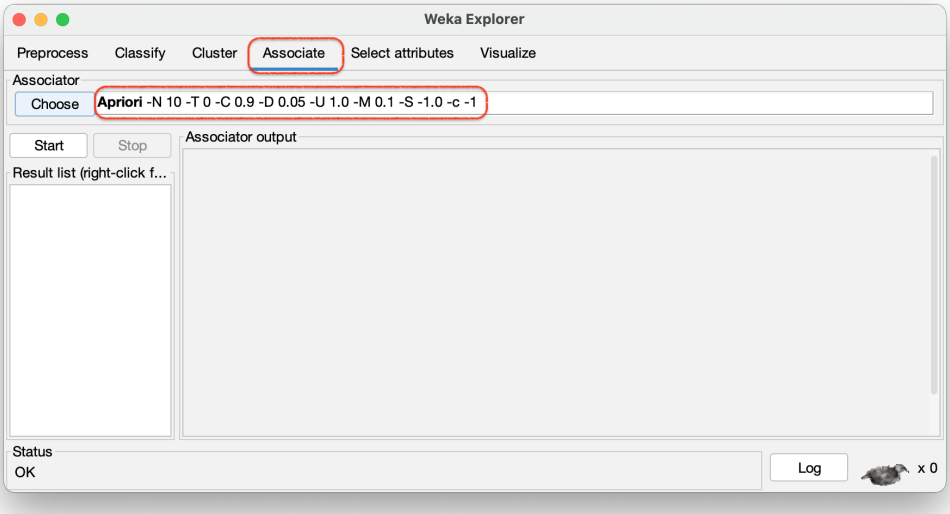


Figura 2.9: Ajuste dos Parâmetros de Entrada do Algoritmo Apriori.

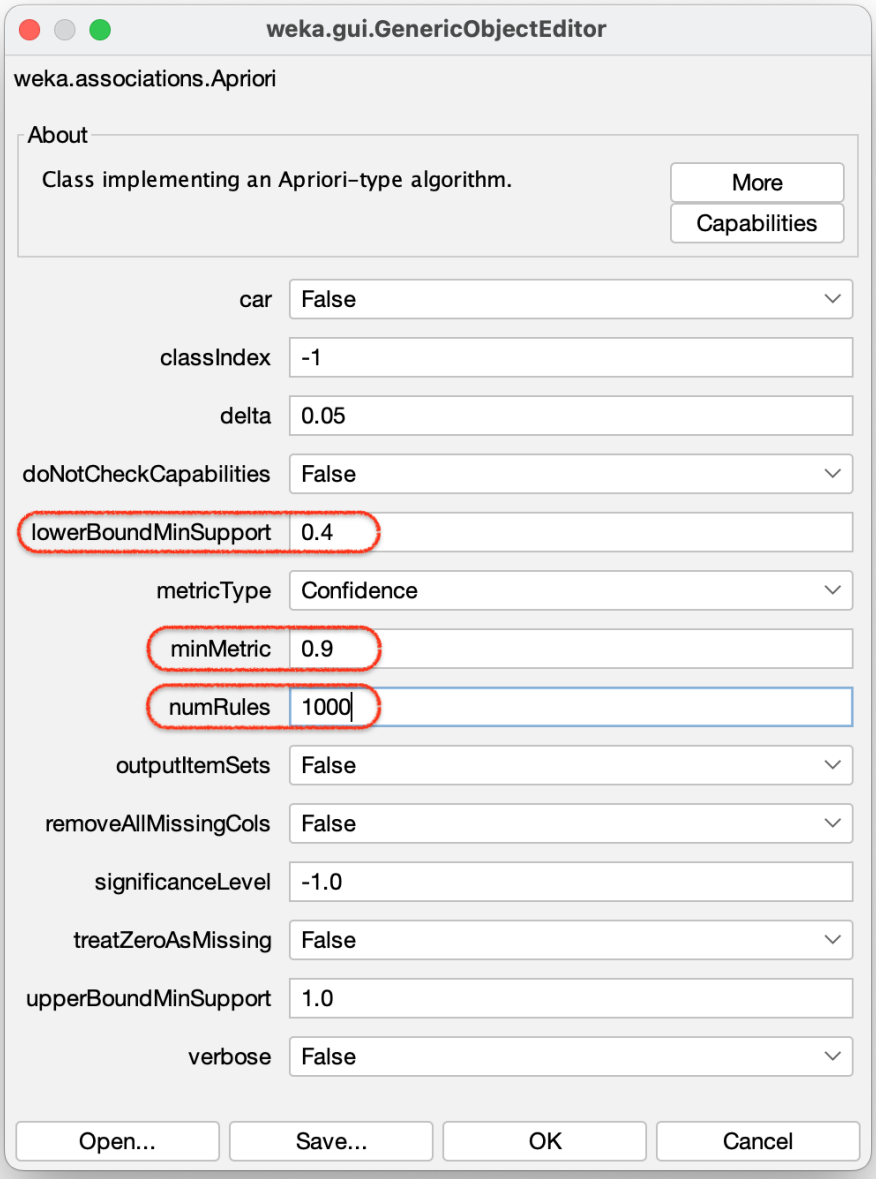


Figura 2.10: Ajuste dos Parâmetros SupMin e ConfMin.

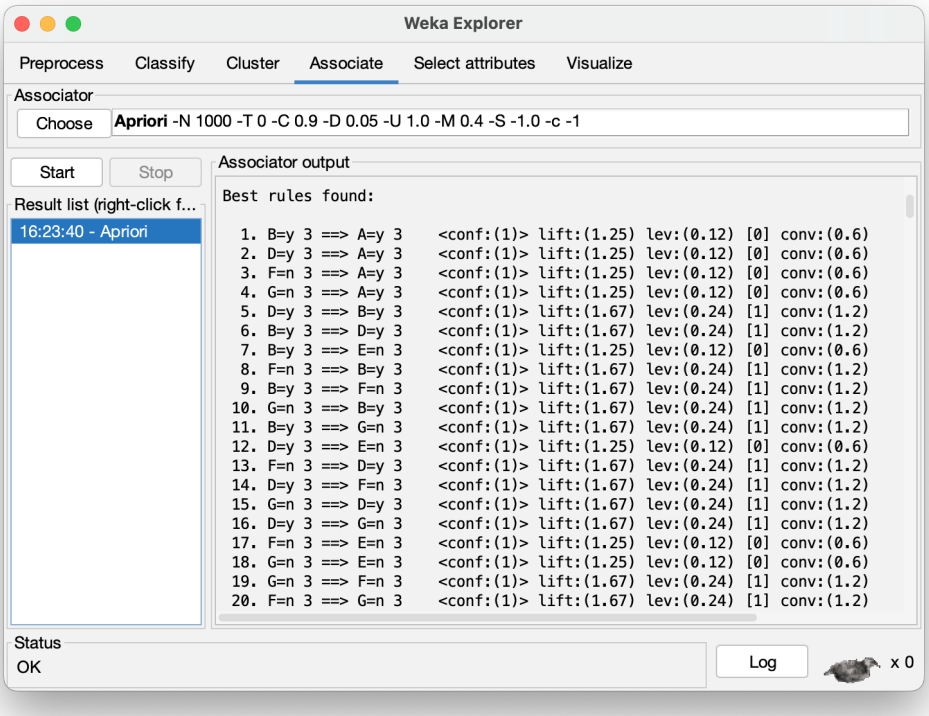


Figura 2.11: Algumas Regras de Associação Geradas com o Arquivo “Transacoes_1.arff”.

2.7 Prática em Python

2.7.1 Introdução

A atividade de **Regras de Associação** foi o tópico escolhido para iniciarmos as principais atividades da Mineração de Dados por envolver ideias bem intuitivas. A analogia entre Regras de Associação e **Cesta de Compras** facilita o entendimento de como descobrir padrões de associação entre itens de um conjunto qualquer.

Uma **Regra de Associação** possui a forma:

$$\mathbf{X} \Rightarrow \mathbf{Y}, \quad \text{sendo } \mathbf{X} \cap \mathbf{Y} = \emptyset$$

onde $\mathbf{X}$ é o **antecedente** e $\mathbf{Y}$ o **consequente**. Note que a associação não implica **causalidade** — apenas ocorrência simultânea com certa probabilidade.

2.7.2 🔹 Configuração do Ambiente

Retomamos o padrão do capítulo anterior, todos os `import` no topo, verificação de versões e uso de **dicionários** para estruturar os dados antes de convertê-los em `DataFrames`.

```
# ============================================================
# BLOCO DE IMPORTAÇÕES
# ============================================================
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)

!pip install networkx -q --no-cache-dir 2>/dev/null

import sys
```

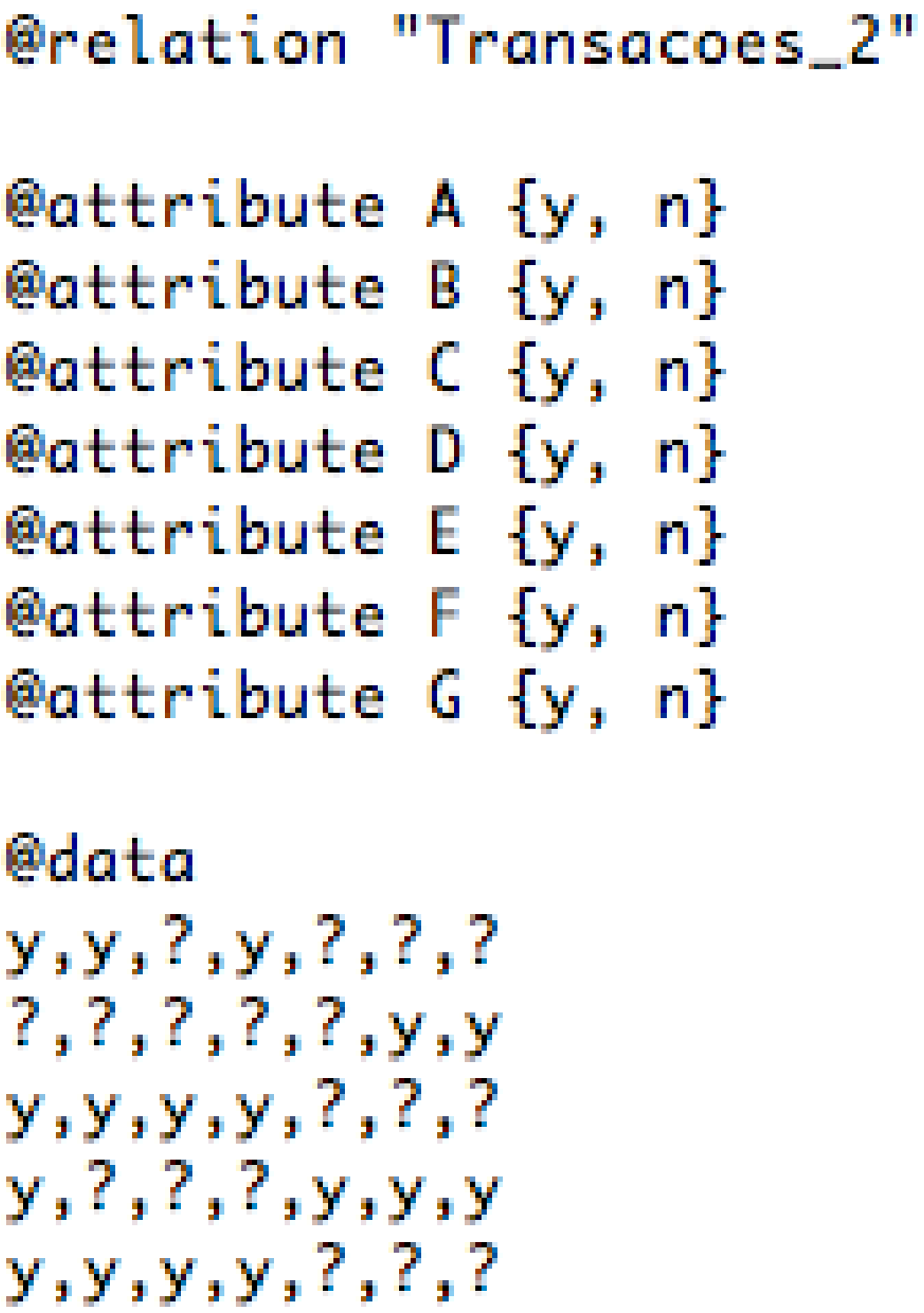


Figura 2.12: Arquivo “Transacoes_2.arff” com Itens Ausentes Representados por “?”.

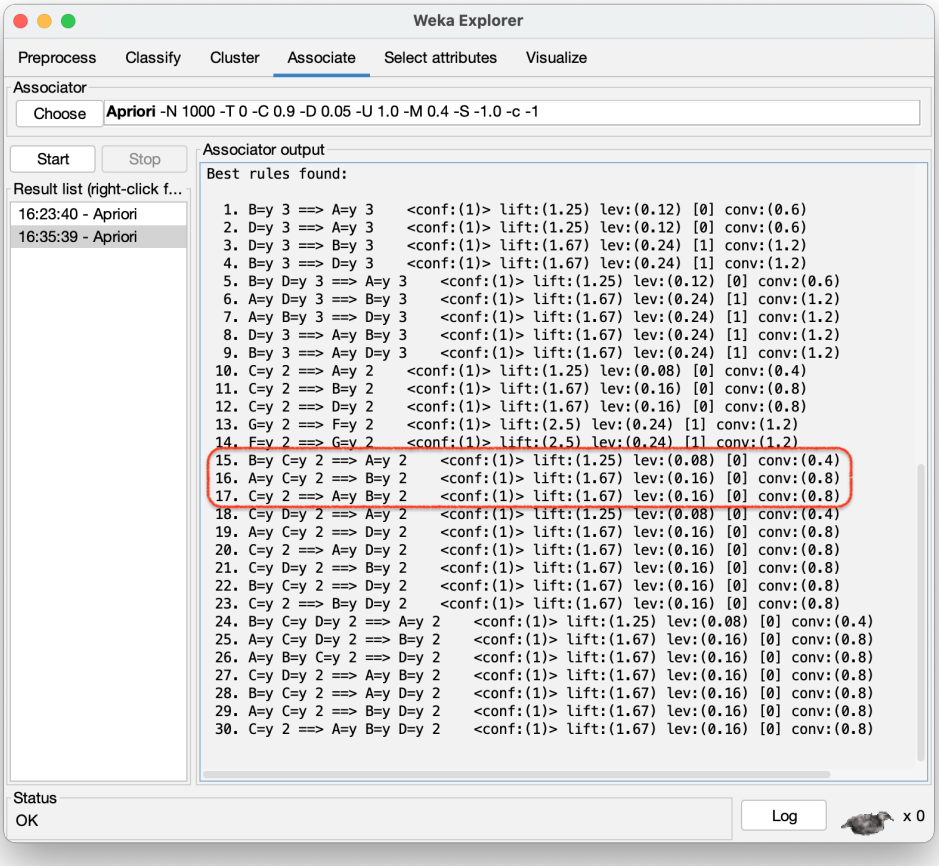


Figura 2.13: As 30 Regras de Associação Geradas com o Arquivo “Transacoes_2.arff”.

```
import re

import numpy as np
import pandas as pd
import networkx as nx
from itertools import combinations
import matplotlib
import matplotlib.pyplot as plt
from IPython.display import Markdown
from dataclasses import dataclass, field

print(f"Python    : {sys.version.split()[0]}")
print(f"Pandas    : {pd.__version__}")
print(f"NumPy     : {np.__version__}")
print(f"Networkx  : {nx.__version__}")
print(f"Matplotlib: {matplotlib.__version__}")
print("  Ambiente pronto!")
```

```
Python    : 3.9.7
Pandas    : 2.3.3
NumPy     : 2.0.2
Networkx  : 3.2.1
Matplotlib: 3.9.4
  Ambiente pronto!
```

2.7.3 🔹 Prática 1 — Itemsets de Tamanho 1 (L_1)

O primeiro passo da mineração é identificar a relevância individual de cada item. Para isso, transformamos a contagem bruta em **Suporte** (**Sup$_0$**), que representa a frequência relativa de um item no conjunto total de transações (N).

Nesta etapa, aplicamos o conceito de **Poda (*Pruning*)**:

- Se definirmos um suporte mínimo de **40%**, qualquer item que apareça menos que isso (como a **Água**, com apenas 20%) é descartado.
- **Por que podar?** Pela propriedade do Apriori, se a "Água" é não frequente sozinha, qualquer combinação que a inclua (como "Água e Pão") também será infrequente. Isso economiza processamento ao ignorar caminhos sem potencial.

2.7.3.1 A Métrica de Cobertura

O cálculo do suporte para um item (ou conjunto) X é dado por:

$$\textbf{Sup}_\mathbf{0}(X) = \frac{\text{frequência de } X}{N}$$

- **Suporte:** Mede a **cobertura** (popularidade). Indica quão representativo aquele item é na base de dados.

O resultado da análise de frequência e a aplicação do critério de poda para itens individuais podem ser observados na Tabela 2.14, gerada pelo código a seguir, que detalha o suporte de cada item e seu status de aceitação.

```
# 1. Dados e Representação
transacoes_raw = [
    {"Arroz", "Feijão", "Óleo"},
    {"Queijo", "Vinho"},
    {"Arroz", "Feijão", "Batata", "Óleo"},
    {"Arroz", "Água", "Queijo", "Vinho"},
    {"Arroz", "Feijão", "Batata", "Óleo"}
]
transacoes = [frozenset(t) for t in transacoes_raw]
```

```python
# Todos os itens na ordem utilizada neste capítulo
todos_itens = ["Arroz", "Feijão", "Batata", "Óleo", "Água", "Queijo", "Vinho"]

# 2. Função de Suporte (Base para as próximas práticas)
def calcular_suporte(itemset, transacoes):
    if not isinstance(itemset, (frozenset, set)):
        itemset = frozenset([itemset]) if isinstance(itemset, str) else frozenset(itemset)
    # Contagem de transações que contêm o itemset
    contagem = sum(1 for t in transacoes if itemset.issubset(t))
    return contagem / len(transacoes)

# 3. Análise do L1 e Poda
SUP_MIN = 0.4
l1_frequentes = []
resultados_l1 = []

for item in todos_itens:
    sup = calcular_suporte(item, transacoes)
    status = "ACEITO" if sup >= SUP_MIN else "PODADO"

    if sup >= SUP_MIN:
        l1_frequentes.append(item)

    # Armazena os dados para o DataFrame
    resultados_l1.append({
        "Item": item,
        "Suporte": f"{sup:.1%}",
        "Status": status
    })

# 4. Exibição Tabular
df_l1 = pd.DataFrame(resultados_l1)
print(f"Configuração: Suporte Mínimo = {SUP_MIN:.0%}\n")

# 5. Ordem descendente pelo Suporte
df_l1 = df_l1.sort_values(by="Suporte", ascending=False)

Markdown(df_l1.to_markdown(index=False, colalign=("center",) * len(df_l1.columns)))
```

```
Configuração: Suporte Mínimo = 40%
```

Tabela 2.14: Análise de Suporte e Poda dos Itens Individuais (L1).

Item	Suporte	Status
Arroz	80.0%	ACEITO
Feijão	60.0%	ACEITO
Óleo	60.0%	ACEITO
Batata	40.0%	ACEITO
Queijo	40.0%	ACEITO
Vinho	40.0%	ACEITO
Água	20.0%	PODADO

2.7.4 🔹 Prática 2 — Itemsets de Tamanho 2 (L_2) e Regras Direcionadas

> **Atenção:** Esta prática é incremental e depende da execução da **Prática 1** (função `calcular_suporte`, lista `l1_frequentes` e `SUP_MIN`).

Após identificarmos os itens isolados que possuem relevância (L_1), avançamos para a formação de pares (L_2). O objetivo aqui é descobrir não apenas o que se vende, mas o que se vende **junto**.

Nesta etapa, a eficiência da **poda** torna-se evidente: utilizamos apenas os itens aprovados na etapa anterior para gerar combinações. Itens descartados (como a **Água**) não entram nos novos testes, reduzindo o esforço computacional. Para validar se um par constitui uma regra forte, acrescentamos a **Confiança**:

$$\textbf{Sup}(X \Rightarrow Y) = \frac{|X \cup Y|}{N}$$

$$\textbf{Conf}(X \Rightarrow Y) = \frac{\textbf{Sup}(X \cup Y)}{\textbf{Sup}(X)}$$

- **Suporte (Sup):** Indica a **cobertura** (popularidade do par no dataset).
- **Confiança (Conf):** Indica a **precisão** ou assertividade. É a probabilidade de o item Y estar presente, dado que o item X já foi adquirido.

Após a filtragem dos itens individuais, geramos as combinações de pares e calculamos sua precisão, conforme apresentado na Tabela 2.15, gerada com a execução do código a seguir, que lista as regras candidatas e seu respectivo suporte e confiança.

```
# 1. Função de Confiança
def calcular_confianca(antecedente, consequente, transacoes):
    sup_total = calcular_suporte(frozenset(antecedente) | frozenset(consequente), transacoes)
    sup_ant = calcular_suporte(antecedente, transacoes)
    return sup_total / sup_ant if sup_ant > 0 else 0.0

# 2. Configuração de Limiares
CONF_MIN = 0.9
resultados_l2 = []

# 3. Geração de Regras e Poda
# Geramos pares apenas com itens que sobreviveram à poda no L1
for combo in combinations(l1_frequentes, 2):
    itemset_par = frozenset(combo)
    sup_par = calcular_suporte(itemset_par, transacoes)

    # Filtro 1: O par deve ser frequente (Suporte)
    if sup_par >= SUP_MIN:
        itens = list(itemset_par)
        # Testamos as duas direções da regra: A -> B e B -> A
        for ant, cons in [(itens[0], itens[1]), (itens[1], itens[0])]:
            conf = calcular_confianca([ant], [cons], transacoes)

            # Filtro 2: A regra deve ser precisa (Confiança)
            status = "ACEITO" if conf >= CONF_MIN else "PODADO"
            resultados_l2.append({
                "Regra": f"{ant} → {cons}",
                "Suporte": f"{sup_par:.0%}",
                "Confiança": f"{conf:.2f}",
                "Status": status
            })

# Criando a lista de pares que realmente são frequentes
l2_frequentes = [frozenset(re.split(r' → ', r['Regra'])) 
```

```
                for r in resultados_l2 if r['Status'] == "ACEITO"]
# Removemos duplicatas (pois A->B e B->A geram o mesmo frozenset)
l2_frequentes = list(set(l2_frequentes))

# 4. Exibição Tabular
df_l2 = pd.DataFrame(resultados_l2)
print(f"Configuração: Confiança Mínima = {CONF_MIN:.0%}\n")

# 5. Ordem descendente pela Confiança
df_l2 = df_l2.sort_values(by="Confiança", ascending=False)

Markdown(df_l2.to_markdown(index=False, colalign=("center",) * len(df_l2.columns)))
```

```
Configuração: Confiança Mínima = 90%
```

Tabela 2.15: Análise de Regras de Associação entre Pares (L2) e Filtro de Confiança.

Regra	Suporte	Confiança	Status
Feijão → Arroz	60%	1	ACEITO
Batata → Arroz	40%	1	ACEITO
Óleo → Arroz	60%	1	ACEITO
Batata → Feijão	40%	1	ACEITO
Óleo → Feijão	60%	1	ACEITO
Feijão → Óleo	60%	1	ACEITO
Batata → Óleo	40%	1	ACEITO
Vinho → Queijo	40%	1	ACEITO
Queijo → Vinho	40%	1	ACEITO
Arroz → Feijão	60%	0.75	PODADO
Arroz → Óleo	60%	0.75	PODADO
Feijão → Batata	40%	0.67	PODADO
Óleo → Batata	40%	0.67	PODADO
Arroz → Batata	40%	0.5	PODADO

2.7.5 🔹 Prática 3 — Itemsets de Tamanho 3 (L_3) e Visualização

Atenção: Esta prática depende das funções e das listas de itens frequentes geradas nas **Práticas 1 e 2**.

À medida que aumentamos o tamanho dos conjuntos para trios (L_3), a análise torna-se mais rica: buscamos regras onde a presença de dois itens "puxa" um terceiro. Introduzimos aqui o **Grafo de Associação**, uma ferramenta visual indispensável quando as tabelas começam a ficar extensas, permitindo identificar o fluxo lógico das decisões de compra.

A evolução para conjuntos de três itens e a validação de regras compostas são apresentadas na Tabela 2.16, gerada com a execução do código a seguir.

```
# 1. Configuração de Candidatos
# trios_candidatos = [frozenset(c) for c in combinations(l1_frequentes, 3)]
trios_candidatos = []

# Comparamos cada par com todos os outros
for i in range(len(l2_frequentes)):
    for j in range(i + 1, len(l2_frequentes)):
        # Une dois pares (ex: {Arroz, Feijão} + {Arroz, Óleo} = {Arroz, Feijão, Óleo})
        candidato = l2_frequentes[i] | l2_frequentes[j]
```

```
        # Só nos interessam uniões que resultem em exatamente 3 itens
        if len(candidato) == 3 and candidato not in trios_candidatos:
            # PODA: Verificamos se TODOS os subconjuntos de 2 itens deste trio são frequentes
            subconjuntos = [frozenset(c) for c in combinations(candidato, 2)]
            if all(s in l2_frequentes for s in subconjuntos):
                trios_candidatos.append(candidato)

print(f"Trios de candidatos gerados via L1: {len(list(combinations(l1_frequentes, 3)))}")
print(f"Trios de candidatos gerados via L2 (com poda): {len(trios_candidatos)}")
print([", ".join(sorted(list(c))) for c in trios_candidatos])

resultados_l3 = []
G_L3 = nx.DiGraph() # Grafo auxiliar para o próximo bloco

# 2. Mineração de Trios e Regras
for trio in trios_candidatos:
    sup_trio = calcular_suporte(trio, transacoes)

    if sup_trio >= SUP_MIN:
        itens = sorted(list(trio))

        for combo in combinations(itens, 2):
            ant = frozenset(combo)
            cons = trio - ant
            conf = calcular_confianca(ant, cons, transacoes)

            ant_str = ", ".join(sorted(list(ant)))
            cons_str = ", ".join(sorted(list(cons)))

            if conf >= CONF_MIN:
                status = "ACEITO"
                G_L3.add_edge(ant_str, cons_str, weight=conf)
            else:
                status = "PODADO"

            resultados_l3.append({
                # .join() transforma a lista em string, removendo [] e ''
                "Itemset (L3)": ", ".join(sorted(list(trio))),
                "Regra": f"{{{ant_str}}} → {{{cons_str}}}",
                "Suporte": f"{sup_trio:.0%}",
                "Confiança": f"{conf:.2f}",
                "Status": status
            })

# 3. Exibição Tabular
df_l3 = pd.DataFrame(resultados_l3)
print(f"Configuração: SUP_MIN {SUP_MIN:.0%} | CONF_MIN: {CONF_MIN:.0%}\n")

# 4. Ordem descendente pela Confiança
df_l3 = df_l3.sort_values(by="Confiança", ascending=False)

Markdown(df_l3.to_markdown(index=False, colalign=("center",) * len(df_l3.columns)))
```

```
Trios de candidatos gerados via L1: 20
Trios de candidatos gerados via L2 (com poda): 4
['Arroz, Batata, Óleo', 'Batata, Feijão, Óleo', 'Arroz, Feijão, Óleo', 'Arroz, Batata, Feijão']
Configuração: SUP_MIN 40% | CONF_MIN: 90%
```

Tabela 2.16: Mineração de Trios Frequentes (L3) e Regras com Múltiplos Antecedentes.

Itemset (L3)	Regra	Suporte	Confiança	Status
Arroz, Batata, Óleo	{Arroz, Batata} → {Óleo}	40%	1	ACEITO
Arroz, Batata, Óleo	{Batata, Óleo} → {Arroz}	40%	1	ACEITO
Batata, Feijão, Óleo	{Batata, Feijão} → {Óleo}	40%	1	ACEITO
Batata, Feijão, Óleo	{Batata, Óleo} → {Feijão}	40%	1	ACEITO
Arroz, Feijão, Óleo	{Arroz, Feijão} → {Óleo}	60%	1	ACEITO
Arroz, Feijão, Óleo	{Arroz, Óleo} → {Feijão}	60%	1	ACEITO
Arroz, Feijão, Óleo	{Feijão, Óleo} → {Arroz}	60%	1	ACEITO
Arroz, Batata, Feijão	{Arroz, Batata} → {Feijão}	40%	1	ACEITO
Arroz, Batata, Feijão	{Batata, Feijão} → {Arroz}	40%	1	ACEITO
Arroz, Batata, Óleo	{Arroz, Óleo} → {Batata}	40%	0.67	PODADO
Batata, Feijão, Óleo	{Feijão, Óleo} → {Batata}	40%	0.67	PODADO
Arroz, Batata, Feijão	{Arroz, Feijão} → {Batata}	40%	0.67	PODADO

Para facilitar a compreensão das interdependências entre esses trios, as regras aceitas são mapeadas visualmente no grafo da Figura 2.14, gerado com a execução do código a seguir.

```
# 4. Visualização do Grafo (Apenas para as Regras ACEITAS)
if not G_L3.edges():
    print("\nNenhuma regra L3 atingiu os critérios para exibição no grafo.")
else:
    plt.figure(figsize=(10, 5))
    # Seed fixo para garantir que o layout seja reproduzível
    pos = nx.spring_layout(G_L3, k=1.5, seed=41)

    # Desenho dos nós e labels
    nx.draw_networkx_labels(G_L3, pos, font_size=10, font_weight="bold")

    # Desenho das arestas com curvatura para evitar sobreposição de regras bidirecionais
    nx.draw_networkx_edges(G_L3, pos, arrowstyle='->', arrowsize=30, edge_color='skyblue',
                           connectionstyle="arc3,rad=0.1", width=2)

    # Labels de Confiança (em vermelho) sobre as arestas
    edge_labels = {k: f"{v:.2f}" for k, v in nx.get_edge_attributes(G_L3, 'weight').items()}
    nx.draw_networkx_edge_labels(G_L3, pos,
                                 edge_labels=edge_labels, font_color='red', font_size=9)

    plt.title("Mapa de Fluxo L3 (Regras de Trios ACEITAS)")
    plt.axis('off')
    plt.show()
```

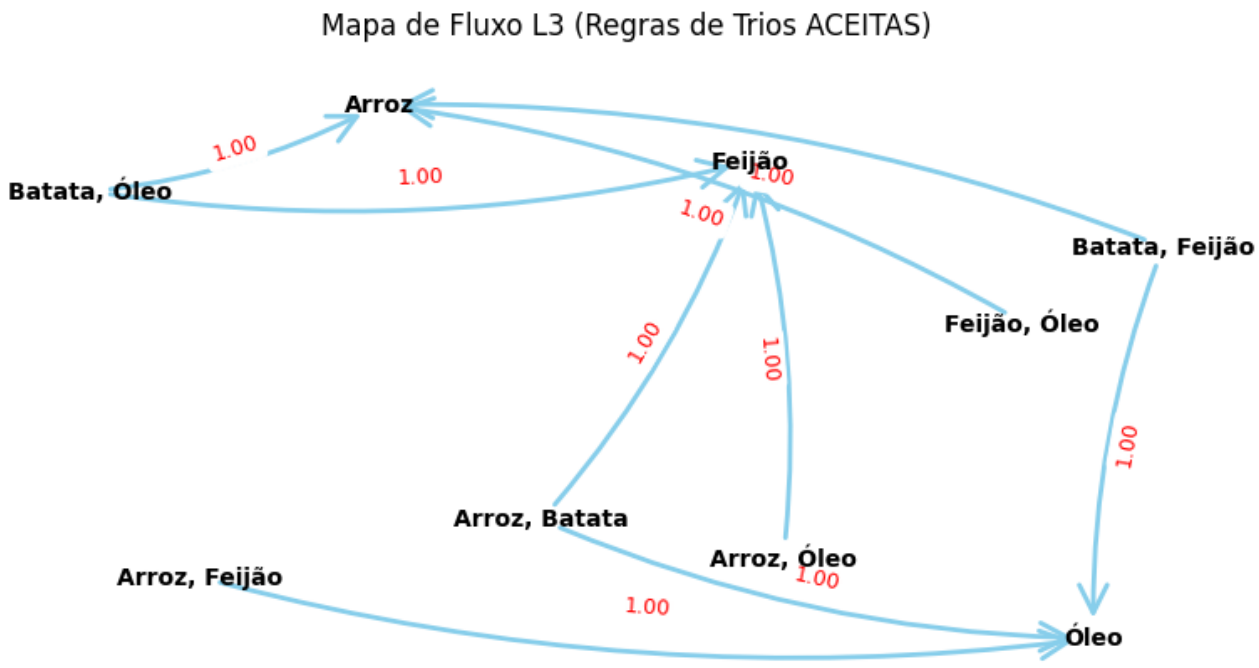


Figura 2.14: Representação visual (Grafo) das regras de associação de nível L3 aceitas.

2.7.5.1 💡 Destaques Didáticos desta Etapa:

1. **Assimetria e Comportamento de Compra:** A tabela revela que a associação não é bidirecional. A **Batata** atua como um forte "item de entrada" (100% de confiança no antecedente), mas falha como item de impulso (podada no consequente), indicando que é um produto planejado pelo consumidor.
2. **Poda Ativa (Propriedade Apriori):** Diferente da busca exaustiva, aqui aplicamos a poda real: um trio só é testado se **todos** os seus pares internos forem frequentes em L_2. Isso demonstra como o algoritmo reduz o espaço de busca, ignorando trios que matematicamente não poderiam ser frequentes.
3. **Eficiência Computacional:** A comparação entre "Candidatos via L1" vs "Candidatos via L2" evidencia a economia de recursos. Ao filtrar os candidatos antes de calcular o suporte na base de dados, mostramos como o Apriori viabiliza a mineração em grandes volumes de dados.
4. **Escalabilidade e Generalização:** O processo de construção de L_4 a partir de L_3, ou de L_5 a partir de L_4, segue exatamente a mesma lógica recursiva de união e poda apresentada até aqui. À medida que o tamanho dos conjuntos (k) aumenta, o "funil" estatístico torna-se mais estreito, restando apenas as associações mais robustas da base de dados.

Para consolidar esse aprendizado e evitar a repetição manual de blocos de código para cada nível, apresentamos a seguir uma **implementação generalizada**. Este código automatiza a descoberta de todos os conjuntos frequentes (L_i) de forma iterativa, prosseguindo até que não restem mais candidatos que satisfaçam os limiares de suporte e confiança definidos.

2.7.6 🔹 Prática 4 — Algoritmo Apriori Completo e a Métrica Lift

> **Atenção:** Esta prática depende das funções `calcular_suporte`, `calcular_confianca` e da variável `transacoes` definidas nas **Práticas 1 e 2**.

O algoritmo **Apriori** fundamenta-se na **propriedade anti-monotônica do suporte**: se um conjunto de itens é infrequente, todos os seus superconjuntos (combinações maiores) também o serão. Isso permite "podar" o espaço de busca, evitando que o computador teste bilhões de combinações inúteis em grandes bases de dados.

Interpretação do Lift (Elevação)

O **Lift** é o "filtro de realidade" da análise de associação. Ele indica se a relação entre dois itens é uma **conexão real** ou apenas coincidência estatística devido à popularidade de um deles.

2.7.6.1 A Lógica Matemática

O cálculo compara a probabilidade de Y ocorrer quando X está presente, contra a probabilidade de Y ocorrer sozinho:

- **Conf**$(X \Rightarrow Y)$ ou $P(Y|X)$: É a chance de levar Y **dado que** X já está no carrinho.
- **Sup**(Y) ou $P(Y)$: É a chance **geral** de alguém levar Y (a popularidade basal).

$$\textbf{Lift}(X \Rightarrow Y) = \frac{\textbf{Conf}(X \Rightarrow Y)}{\textbf{Sup}(Y)} = \frac{P(Y|X)}{P(Y)}$$

- **Lift** > 1**:** O item X realmente impulsiona a venda de Y. Há uma relação de valor (associação positiva).
- **Lift** ≈ 1**:** A relação é aleatória. Y é comprado com ou sem X na mesma proporção.
- **Lift** < 1**:** Efeito de substituição. Quem compra X tende a **não** comprar Y (associação negativa).

2.7.6.2 Implementação: Automação Total e Relatório de Inteligência

O código abaixo integra a descoberta iterativa de itemsets (*loop* $L_1, L_2, L_3, \ldots, L_n$), a validação de confiança e o cálculo de **Lift**, exibindo o status final de cada regra.

O relatório consolidado de todas as regras encontradas pelo algoritmo, priorizadas pela força de associação, é exibido Tabela 2.17.

```
@dataclass
class RegraAssociacao:
    antecedente: frozenset
    consequente: frozenset
    suporte: float
    confianca: float
    lift: float = 0.0

def apriori(transacoes, sup_min, conf_min):
    # Dicionário para armazenar suportes já calculados: {itemset: valor}
    suportes_cache = {}
    n_transacoes = len(transacoes)

    def get_suporte(itemset):
        if itemset not in suportes_cache:
            count = sum(1 for t in transacoes if itemset.issubset(t))
            suportes_cache[itemset] = count / n_transacoes
        return suportes_cache[itemset]

    itens = sorted(set().union(*transacoes))
    frequentes = []
    # L1: Itens individuais
    candidatos_k = [frozenset([i]) for i in itens]

    # Etapa 1: Itemsets Frequentes (Geração e Poda)
    while candidatos_k:
        # Filtro de Suporte: Identifica quem é frequente no nível atual
        freq_k = [c for c in candidatos_k if get_suporte(c) >= sup_min]
        frequentes.extend(freq_k)

        if not freq_k: break
```

```
        # Gerar Candidatos para o próximo nível (k + 1)
        k_proximo = len(freq_k[0]) + 1
        freq_set = set(freq_k) # Set para busca rápida O(1)

        # união (Join)
        candidatos_brutos = {a | b for a, b in combinations(freq_k, 2)
                             if len(a | b) == k_proximo}

        # PODA (Pruning): Só mantém o candidato se
        # TODOS os seus subconjuntos de tamanho k forem frequentes
        candidatos_k = []
        for cand in candidatos_brutos:
            subconjuntos = [frozenset(c) for c in combinations(cand, k_proximo - 1)]
            if all(s in freq_set for s in subconjuntos):
                candidatos_k.append(cand)

    # Etapa 2: Regras de Associação
    regras_aceitas = []
    dados_relatorio = []

    for itemset in frequentes:
        if len(itemset) < 2: continue

        # Para cada subconjunto do itemset frequente
        # (Gera regras muitos-para-um e muitos-para-muitos)
        for tam_ant in range(1, len(itemset)):
            for ant_tuple in combinations(itemset, tam_ant):
                ant = frozenset(ant_tuple)
                con = itemset - ant

                sup_total = get_suporte(itemset)
                sup_ant = get_suporte(ant)
                sup_con = get_suporte(con)

                # Cálculos de métricas
                conf = sup_total / sup_ant
                # Lift: Confiança dividida pelo suporte basal do consequente
                lift = conf / sup_con if sup_con > 0 else 0.0

                status = "ACEITO" if conf >= conf_min else "PODADO"
                regra = RegraAssociacao(ant, con, sup_total, conf, lift)

                if conf >= conf_min:
                    regras_aceitas.append(regra)

                # Formatação para o relatório
                ant_str = ", ".join(sorted(list(ant)))
                con_str = ", ".join(sorted(list(con)))

                dados_relatorio.append({
                    "Regra": f"{{{ant_str}}} → {{{con_str}}}",
                    "Suporte": f"{regra.suporte:.0%}",
                    "Confiança": f"{regra.confianca:.2f}",
                    "Lift": f"{regra.lift:.2f}",
                    "Status": status
                })
```

```
    return regras_aceitas, dados_relatorio

# --- Execução ---
regras_finais, relatorio_lista = apriori(transacoes, SUP_MIN, CONF_MIN)

# Criar DataFrame e ordenar decrescentemente pelo Lift (força da regra)
df_final = pd.DataFrame(relatorio_lista)
if not df_final.empty:
    df_final['Lift_Num'] = df_final['Lift'].astype(float)
    df_final = df_final.sort_values(by='Lift_Num', ascending=False).drop(columns=['Lift_Num'])

print(f"RELATÓRIO APRIORI COMPLETO (N={len(transacoes)}, CONF_MIN={CONF_MIN})\n")
Markdown(df_final.to_markdown(index=False, colalign=("center",) * len(df_final.columns)))
```

```
RELATÓRIO APRIORI COMPLETO (N=5, CONF_MIN=0.9)
```

Tabela 2.17: Relatório Consolidado de Regras de Associação (Algoritmo Apriori Completo).

Regra	Suporte	Confiança	Lift	Status
{Queijo} → {Vinho}	40%	1	2.5	ACEITO
{Vinho} → {Queijo}	40%	1	2.5	ACEITO
{Óleo} → {Batata}	40%	0.67	1.67	PODADO
{Batata} → {Óleo}	40%	1	1.67	ACEITO
{Arroz, Batata} → {Feijão}	40%	1	1.67	ACEITO
{Feijão} → {Arroz, Batata}	40%	0.67	1.67	PODADO
{Batata} → {Feijão}	40%	1	1.67	ACEITO
{Feijão} → {Batata}	40%	0.67	1.67	PODADO
{Óleo} → {Feijão}	60%	1	1.67	ACEITO
{Feijão} → {Óleo}	60%	1	1.67	ACEITO
{Arroz, Feijão} → {Batata}	40%	0.67	1.67	PODADO
{Batata} → {Arroz, Feijão}	40%	1	1.67	ACEITO
{Feijão} → {Arroz, Óleo}	60%	1	1.67	ACEITO
{Óleo} → {Arroz, Feijão}	60%	1	1.67	ACEITO
{Batata, Feijão} → {Óleo}	40%	1	1.67	ACEITO
{Feijão, Óleo} → {Batata}	40%	0.67	1.67	PODADO
{Batata, Óleo} → {Feijão}	40%	1	1.67	ACEITO
{Feijão} → {Batata, Óleo}	40%	0.67	1.67	PODADO
{Batata} → {Feijão, Óleo}	40%	1	1.67	ACEITO
{Óleo} → {Batata, Feijão}	40%	0.67	1.67	PODADO
{Arroz, Feijão, Óleo} → {Batata}	40%	0.67	1.67	PODADO
{Arroz, Feijão} → {Batata, Óleo}	40%	0.67	1.67	PODADO
{Arroz, Batata} → {Feijão, Óleo}	40%	1	1.67	ACEITO
{Batata, Feijão} → {Arroz, Óleo}	40%	1	1.67	ACEITO
{Arroz, Óleo} → {Batata, Feijão}	40%	0.67	1.67	PODADO
{Batata} → {Arroz, Feijão, Óleo}	40%	1	1.67	ACEITO
{Feijão, Óleo} → {Arroz, Batata}	40%	0.67	1.67	PODADO
{Batata, Óleo} → {Arroz, Feijão}	40%	1	1.67	ACEITO
{Batata} → {Arroz, Óleo}	40%	1	1.67	ACEITO
{Arroz, Óleo} → {Feijão}	60%	1	1.67	ACEITO
{Arroz, Feijão} → {Óleo}	60%	1	1.67	ACEITO

Tabela 2.17: Relatório Consolidado de Regras de Associação (Algoritmo Apriori Completo).

Regra	Suporte	Confiança	Lift	Status
{Óleo} → {Arroz, Batata}	40%	0.67	1.67	PODADO
{Feijão} → {Arroz, Batata, Óleo}	40%	0.67	1.67	PODADO
{Óleo} → {Arroz, Batata, Feijão}	40%	0.67	1.67	PODADO
{Arroz, Batata} → {Óleo}	40%	1	1.67	ACEITO
{Arroz, Óleo} → {Batata}	40%	0.67	1.67	PODADO
{Arroz, Batata, Feijão} → {Óleo}	40%	1	1.67	ACEITO
{Arroz, Batata, Óleo} → {Feijão}	40%	1	1.67	ACEITO
{Arroz} → {Batata}	40%	0.5	1.25	PODADO
{Arroz} → {Batata, Feijão}	40%	0.5	1.25	PODADO
{Arroz} → {Feijão}	60%	0.75	1.25	PODADO
{Feijão} → {Arroz}	60%	1	1.25	ACEITO
{Óleo} → {Arroz}	60%	1	1.25	ACEITO
{Arroz} → {Óleo}	60%	0.75	1.25	PODADO
{Batata, Feijão} → {Arroz}	40%	1	1.25	ACEITO
{Batata} → {Arroz}	40%	1	1.25	ACEITO
{Feijão, Óleo} → {Arroz}	60%	1	1.25	ACEITO
{Arroz} → {Feijão, Óleo}	60%	0.75	1.25	PODADO
{Batata, Óleo} → {Arroz}	40%	1	1.25	ACEITO
{Arroz} → {Batata, Óleo}	40%	0.5	1.25	PODADO
{Arroz} → {Batata, Feijão, Óleo}	40%	0.5	1.25	PODADO
{Batata, Feijão, Óleo} → {Arroz}	40%	1	1.25	ACEITO

Por fim, a Figura 2.15 apresenta o grafo de influência final, onde a graduação de cores destaca o valor estratégico (**Lift**) de cada associação descoberta.

```
# --- Grafo de Lift (Apenas regras ACEITAS) ---
G_final = nx.DiGraph()
for r in regras_finais:
    G_final.add_edge(", ".join(sorted(r.antecedente)),
                     ", ".join(sorted(r.consequente)), weight=r.lift)

if not G_final.edges():
    print("\nNenhuma regra atingiu os critérios para exibição no grafo.")
else:
    plt.figure(figsize=(12, 6))
    pos = nx.spring_layout(G_final, k=2.5, seed=1)
    lifts = [G_final[u][v]['weight'] for u, v in G_final.edges()]

    nx.draw_networkx_labels(G_final, pos, font_size=10, font_weight='bold')

    # Mapeamento de cores baseado no Lift
    sm = plt.cm.ScalarMappable(cmap=plt.cm.Reds,
                               norm=plt.Normalize(vmin=min(lifts), vmax=max(lifts)))

    nx.draw_networkx_edges(G_final, pos, edge_color=lifts, edge_cmap=plt.cm.Reds, width=2,
                           arrowsize=25, connectionstyle="arc3,rad=0.2")

    plt.colorbar(sm, ax=plt.gca(), label='Lift (Força da Associação)')
    plt.axis('off')
```

```
    plt.show()
```

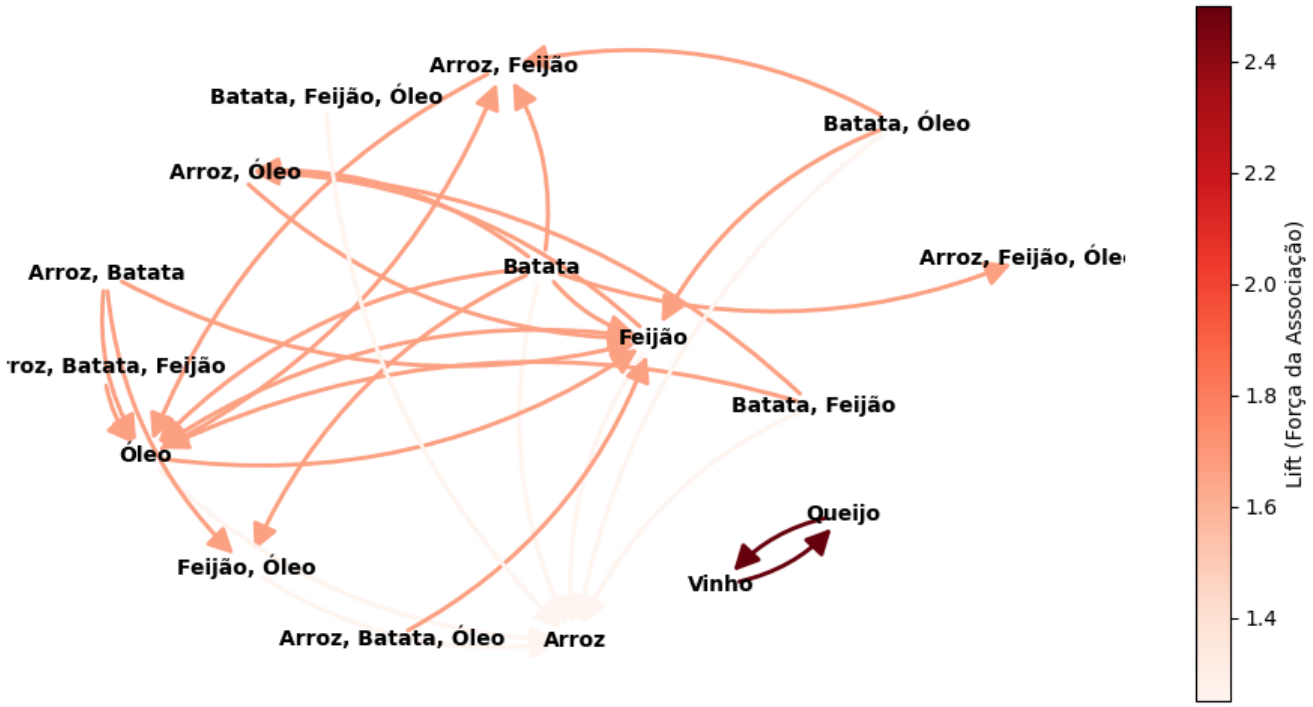


Figura 2.15: Grafo de Influência: A intensidade da cor (Reds) indica o valor estratégico do Lift.

Desafio: Nesta Prática 4, o código gera candidatos e testa o suporte na base de dados. Como poderíamos alterar a função para que ela descarte um candidato se um de seus subconjuntos não for frequente, sem precisar consultar a base de dados?

2.7.7 🔹 Prática 5 — Biblioteca MLxtend

A biblioteca **MLxtend** contém implementações otimizadas do **Apriori** e também do **FP-Growth** (que é ainda mais rápido).

Instalação:

```
!pip install mlxtend -q --no-cache-dir 2>/dev/null
```

2.7.7.1 Implementação Direta

O uso dessas bibliotecas requer um pré-processamento para converter a lista de transações em uma matriz booleana (*One-Hot Encoding*). A Tabela 2.18 apresenta o resultado gerado pelo código a seguir. Caso esteja utilizando o Colab ou Jupyter Notebook, execute o código abaixo e, em uma nova célula, imprima a variável `te_ary` para visualizar a estrutura da matriz lógica.

```
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

# 1. Dados de exemplo
dataset = [["Arroz", "Feijão", "Óleo"],
           ["Queijo", "Vinho"],
           ["Arroz", "Feijão", "Batata", "Óleo"],
           ["Arroz", "Água", "Vinho", "Queijo"],
           ["Arroz", "Feijão", "Batata", "Óleo"]]

# 2. Processamento MLxtend
```

```
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = apriori(df, min_support=0.4, use_colnames=True)
regras = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.9)

# --- RELATÓRIO ---

# 3. Preparação do DataFrame Final
df_final = regras[['antecedents', 'consequents', 'support', 'confidence', 'lift']].copy()

# ORDENAÇÃO: Sempre antes de formatar como string para garantir precisão matemática
df_final = df_final.sort_values(by='lift', ascending=False)

# LIMPEZA E JUNÇÃO: Criando a coluna única "Regra"
df_final['Regra'] = df_final.apply(
    lambda x: f"{{{', '.join(list(x['antecedents']))}}} → \
        {{{', '.join(list(x['consequents']))}}}",
    axis=1
)

# SELEÇÃO DE COLUNAS: Mantendo apenas a Regra e as métricas
df_final = df_final[['Regra', 'support', 'confidence', 'lift']]
df_final.columns = ['Regra', 'Suporte', 'Confiança', 'Lift']

# --- FORMATAÇÃO VISUAL ---
df_final['Suporte'] = df_final['Suporte'].map('{:.1%}'.format)
df_final['Confiança'] = df_final['Confiança'].map('{:.2f}'.format)
df_final['Lift'] = df_final['Lift'].map('{:.2f}'.format)

# 4. Exibição do Relatório
Markdown(df_final.to_markdown(index=False, colalign=("center",) * len(df_final.columns)))
```

Tabela 2.18: Regras de Associação Geradas com a Biblioteca MLxtend.

Regra	Suporte	Confiança	Lift
{Queijo} → {Vinho}	40.0%	1	2.5
{Vinho} → {Queijo}	40.0%	1	2.5
{Óleo} → {Feijão}	60.0%	1	1.67
{Feijão} → {Óleo}	60.0%	1	1.67
{Batata} → {Óleo}	40.0%	1	1.67
{Batata} → {Feijão}	40.0%	1	1.67
{Batata} → {Arroz, Feijão}	40.0%	1	1.67
{Batata, Arroz} → {Óleo}	40.0%	1	1.67
{Batata, Arroz} → {Feijão}	40.0%	1	1.67
{Óleo, Batata} → {Feijão}	40.0%	1	1.67
{Óleo} → {Arroz, Feijão}	60.0%	1	1.67
{Arroz, Feijão} → {Óleo}	60.0%	1	1.67
{Óleo, Arroz} → {Feijão}	60.0%	1	1.67
{Batata} → {Óleo, Arroz}	40.0%	1	1.67
{Batata, Feijão} → {Óleo, Arroz}	40.0%	1	1.67
{Óleo, Batata, Arroz} → {Feijão}	40.0%	1	1.67

Tabela 2.18: Regras de Associação Geradas com a Biblioteca MLxtend.

Regra	Suporte	Confiança	Lift
{Batata, Arroz, Feijão} → {Óleo}	40.0%	1	1.67
{Óleo, Batata} → {Arroz, Feijão}	40.0%	1	1.67
{Batata} → {Óleo, Arroz, Feijão}	40.0%	1	1.67
{Feijão} → {Óleo, Arroz}	60.0%	1	1.67
{Batata, Feijão} → {Óleo}	40.0%	1	1.67
{Batata} → {Óleo, Feijão}	40.0%	1	1.67
{Batata, Arroz} → {Óleo, Feijão}	40.0%	1	1.67
{Óleo, Batata} → {Arroz}	40.0%	1	1.25
{Feijão} → {Arroz}	60.0%	1	1.25
{Óleo} → {Arroz}	60.0%	1	1.25
{Batata} → {Arroz}	40.0%	1	1.25
{Batata, Feijão} → {Arroz}	40.0%	1	1.25
{Óleo, Feijão} → {Arroz}	60.0%	1	1.25
{Óleo, Batata, Feijão} → {Arroz}	40.0%	1	1.25

2.7.7.2 Outras Bibliotecas

Além da **MLxtend**, existem outras opções dependendo do seu volume de dados:

1. **PyCaret:** É uma biblioteca de *Low-Code*. Com apenas duas linhas (`setup` e `create_model`), ela faz todo o processo acima e ainda gera gráficos interativos. É excelente para quem quer rapidez.
2. **Apyori:** Uma biblioteca bem simples e leve, que não depende do Pandas ou Scikit-learn. É boa para scripts rápidos onde você não quer instalar dependências pesadas.
3. **Efficient-Apriori:** Focada em performance e memória, permite passar as transações como um gerador Python (bom para arquivos CSV tão grandes que não cabem na memória RAM).

2.7.7.3 Comparação com este capítulo

Ao usar a **MLxtend**, você ganha:

- **Performance:** Capaz de lidar com milhares de itens e transações em segundos.
- **Métricas Extras:** Além de Suporte, Confiança e Lift, ela calcula automaticamente o **Leverage** e a **Conviction**.
- **Integração com Pandas:** O resultado já é um DataFrame.

2.7.8 Conclusão do Ciclo de Aprendizado

Através destas cinco práticas, percorremos desde a organização de dados brutos até a mineração avançada e otimizada. Vimos que a Mineração de Regras de Associação não é uma busca cega, mas um processo científico baseado em:

1. **Poda (*Pruning*):** Redução drástica do esforço computacional ao ignorar candidatos improváveis.
2. **Métricas (Sup, Conf, Lift):** Filtros estatísticos que separam o ruído do conhecimento real e valioso.
3. **Visualização (Grafos):** Ferramentas que permitem ao analista ignorar o óbvio e focar em padrões de alto valor estratégico para o negócio.

2.8 🤖 Uso do NotebookLM como Tutor Complementar

Seguindo o padrão do capítulo anterior, além dos notebooks interativos no Google Colab, incentiva-se o uso do **NotebookLM** como ferramenta complementar de aprendizagem. Essa ferramenta de IA utiliza exclusivamente os documentos fornecidos pelos autores como base de conhecimento, garantindo respostas alinhadas ao conteúdo do livro.

Para cada capítulo, foi preparado um projeto específico na plataforma. Para uma experiência de estudo ampliada, utilize o acesso abaixo:

🎓 Estude com o Tutor Inteligente

Para interagir com o conteúdo deste capítulo, acesse o link abaixo. O ambiente contém materiais didáticos em diferentes formatos, gerados a partir do **PDF** do capítulo. Na plataforma, explore especialmente as opções **Guia de Estudo** e **Conversa** para aprofundar sua compreensão.
🚀 ACESSAR NOTEBOOKLM: CAPÍTULO 02

2.9 Considerações Finais

Um conjunto de Regras de Associação constitui uma forma de conhecimento extraído de uma Base de Dados, sendo esta representação do conhecimento geralmente um tipo de aprendizado muito útil para aplicações práticas, como o aumento de vendas de uma rede de supermercados, o projeto de catálogos de novos produtos ou o lançamento de campanhas promocionais baseadas em vendas casadas. Geralmente quando fazemos busca na Web, ao digitarmos uma palavra de busca é comum que outras palavras sejam sugeridas. Isso ocorre porque a ferramenta de busca está usando Regras de Associação e tem em sua Base de Dados registros de que pessoas que buscam a Palavra_1 geralmente buscam também a `Palavra_2`, a `Palavra_3`, e assim por diante.

Nesta primeira abordagem da extração de conhecimento a partir de uma Base de Dados foi suficiente apenas um procedimento algoritmo, sem necessidade de inferências. Nos próximos capítulos vamos mostrar que na prática é comum nos depararmos com situações para as quais não se conhece um algoritmo que produza o conhecimento necessário para uma tomada de decisão. Para estes casos, será necessário pensar num mecanismo de inferência que nos permita chegar à conclusão mais plausível para determinada situação. Esse é o caso de sistemas conhecidos como Sistemas Especialistas, que auxiliam por exemplo um médico a fazer diagnóstico a partir dos sintomas do paciente. Como nem sempre os sintomas declarados pelo paciente são compatíveis com determinada doença, ou então porque o paciente omite determinados sintomas importantes para o diagnóstico correto, o sistema precisa fazer inferências comparando sua Base [permanente] de Conhecimento com os sintomas declarados.

Regras de Associação frequentemente usam atributos nominais (por exemplo, temperatura elevada, amena, baixa) e mais raramente atributos numéricos (por exemplo 40°C, 23°C, 4°C), porque algoritmos para extração de Regras de Associação com atributos numéricos não costumam apresentar bom desempenho em grandes Bases de Dados. Além disso, ao não levar em conta por exemplo o preço de um artigo ou a quantidade de itens vendidos em cada transação, as Regras de Associação geralmente se transformam numa forma simplista de representação do conhecimento extraído da Base de Dados.

No exemplo da Cesta de Artigos mostramos como gerar Regras de Associação que indiquem venda casada dos artigos mais comum. Mas, frequentemente, os especialistas em vendas não estão muito interessados nestes itens porque a associação entre eles já é conhecida. Na realidade, estes especialistas buscam pares de itens dos quais um deles é um produto barato e o outro tem alta taxa de lucro. Nestes casos, lançar uma superpromoção do produto barato faz com que as vendas do produto com alta taxa de lucro aumente.

Em nossa Cesta de Artigos está implícito o padrão de associação entre Queijo e Vinho. Talvez aí, numa campanha de inverno, cadeias de supermercados possam fazer promoções de queijos com o único propósito de vender mais vinhos. Mas como as vendas de ambos eram relativamente baixas, esta regra não satisfez os critérios estabelecidos de **SupMin** e **ConfMin**. E, no entanto, é possivelmente este tipo de informação a mais procurada. O que fazer para conseguir minerar as pérolas de informação?

2.10 Lista de Exercícios

1. (20%) Explique com suas próprias palavras a importância do **Suporte Mínimo (SupMin)** e **Confiança Mínima (ConfMin)** para a geração de **Regras de Associação**.
2. (30%) Explique com suas próprias palavras o que é **Conjunto Frequente** no contexto das **Regras de Associação**.
3. (50%) Crie uma pequena **Cesta de Compras** ($\pm$ 5 Exemplos) com itens relacionados ao seu ambiente de trabalho, ou à área de seu TCC, ou a qualquer outra área de seu interesse, e gere as **Regras de Associação**

no Weka. Anexe o respectivo arquivo "`.arff`", e um pequeno relatório sobre a simulação.

2.10.1 🔧 Desafios de Programação (extras)

4. **(Médio)** Adicione a métrica **Lift** à função `apriori()` da Prática 4 como critério de filtragem (parâmetro `lift_min`).
5. **(Avançado)** Pesquise o algoritmo **FP-Growth** como alternativa mais eficiente ao Apriori. Quais são as principais vantagens em bases de dados muito grandes?

2.11 Referências do Capítulo

Este capítulo baseou-se nas obras fundamentais que consolidaram as bases da mineração de dados: **agrawal1993mining**, para a introdução histórica e formal das regras de associação; **frank2016weka**, como documentação técnica e referência do sistema WEKA; **rocha2008analise**, **tan2009** e **witten2005**, para os conceitos estruturais do processo de descoberta de conhecimento (KDD) e algoritmos de mineração; **padhy2010**, para a integração com sistemas inteligentes; e o trabalho clássico de **quinlan1986**, sobre a indução de árvores de decisão.

Classificação e Árvores de Decisão

3.1 Introdução

É de grande interesse, em muitas situações, conseguir classificar antecipadamente o tipo de problema apresentado por um paciente com base nos sintomas relatados e tomar medidas para combater determinada doença em seu estágio inicial. Em muitos casos reais isso tem sido possível graças à análise minuciosa de Bases de Dados contendo anotações médicas de outros pacientes com soluções bem sucedidas previamente documentadas.

Em instituições financeiras, para um gerente de banco nem sempre é algo simples fazer uma avaliação de risco sobre a concessão de empréstimos de alto valor. Com base em dados de transações anteriores e nas características específicas de cada cliente, quase sempre é possível extrair automaticamente informações não óbvias que ajudam a classificar um correntista como bom ou mau pagador.

Estes são apenas alguns casos em que se verifica que há sempre informações úteis e não evidentes em grandes Bases de Dados. Estas informações podem ser automaticamente extraídas com Mineração de Dados e interpretadas de modo a constituir conhecimento especializado e útil para a tomada de decisão. A representação do conhecimento através de Árvores de Decisão vai ser o tema deste capítulo.

3.2 Classificação

Classificação é uma forma de **modelagem preditiva**, isto é, com base nos **atributos de entrada** de um objeto é possível predizer o **atributo de saída** desse objeto. Na prática, os Exemplos de uma Base de Dados estão previamente rotulados em duas ou mais classes para serem utilizados num processo de treinamento, cujo fim é criar uma estrutura de representação do conhecimento contido nessa Base de Dados.

Quando se fala em Exemplos previamente rotulados geralmente se subentende que eles serão usados em Aprendizado Supervisionado de Máquina. Os rótulos ou classes dos Exemplos orientam o processo de treinamento, ao final do qual se obtém um Modelo que sintetiza todo o conhecimento contido nas variáveis ou nos atributos. Este Modelo pode então ser usado para prever o valor da variável alvo ou do atributo de saída de novos Exemplos desconhecidos.

O objetivo então da Classificação é predizer em que classe um novo Exemplo, não pertencente ao Conjunto de Treinamento, deve ser colocado. Para que esta tarefa possa ser adequadamente desempenhada é necessário extrair da Base de Dados uma estrutura de conhecimento, tal como Árvores de Decisão ou Regras de Classificação.

Em outras palavras, o Modelo induzido por inferência é equivalente a uma função que mapeia valores de entrada, geralmente denominados variáveis independentes ou explicativas, a um único valor de saída, geralmente denominado variável dependente ou alvo. Na **Classificação**, a variável de saída, via de regra, é discreta (ou categórica), enquanto que na **Regressão** a variável de saída é contínua.

A Figura 3.1 ilustra simplificadamente um estudo clássico introduzido por (**fisher1936**), cujo artigo original apresenta três conjuntos com 50 amostras (ou Exemplos), totalizando 150 medidas do comprimento e da largura de uma pequena flor conhecida como Flor de Lis ou Íris. De acordo com os atributos de entrada “Comprimento” e “Largura” da pétala, cada Exemplo dessa flor pode ser classificado em uma das três classes: Setosa, Versicolor ou Virgínica.

As linhas tracejadas no gráfico ajudam a entender por que a classificação da Íris do tipo Setosa pode ser mais simples que a dos tipos Versicolor e Virgínica. Como a Íris do tipo Setosa apresenta largura e comprimento da pétala bem menor que as outras duas, basta considerar apenas um dos atributos, digamos Comprimento $< 2,5$ cm, para poder classificá-la corretamente. No caso dos tipos Versicolor e Virgínica, tanto o atributo Comprimento quanto Largura se sobrepõem em algumas regiões do gráfico e, portanto, poderá haver erro associado à classificação destes tipos de Íris.

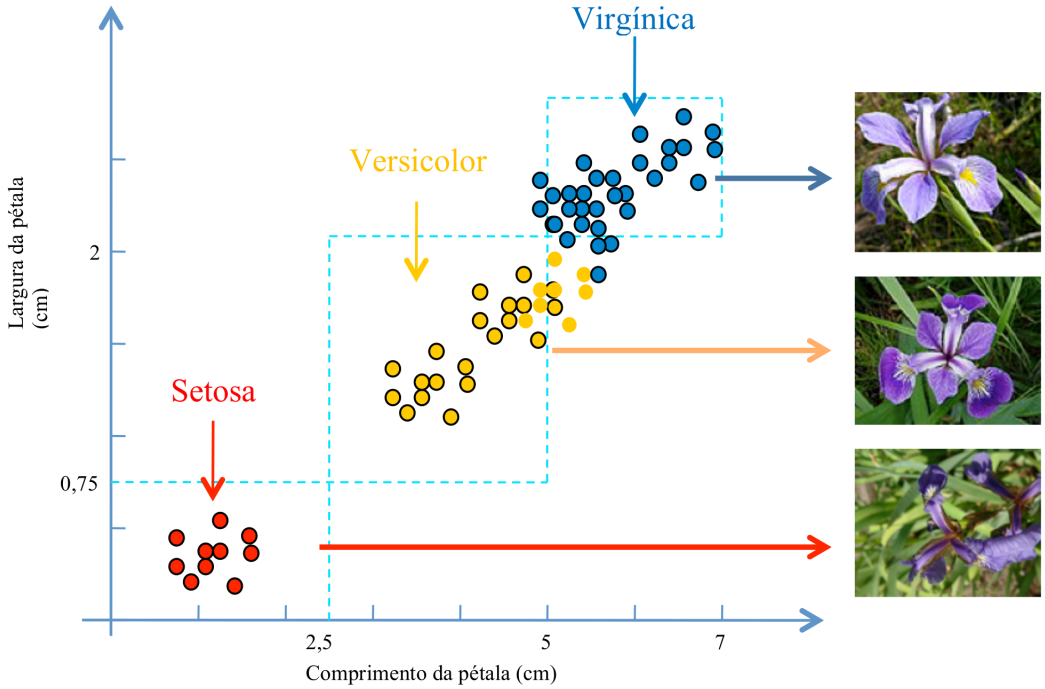


Figura 3.1: Representação do Estudo da Flor Íris com Dois Atributos [1] [2] [3].

Em muitas aplicações práticas é perfeitamente aceitável a utilização de algoritmos relativamente simples que produzam um modelo claro de representação do conhecimento, mesmo que isso implique uma certa taxa de erro na classificação. Modelos facilmente compreensíveis de representação de conhecimento, como Árvores de Decisão e Regras de Classificação, permitem que um especialista avalie o modelo e detecte problemas em sua estrutura. Tanto para o médico quanto para o gerente de banco que se utilizam de um modelo de classificação para auxiliar em sua decisão, é importante que eles consigam interpretar todos os passos lógicos utilizados pelo sistema para chegar àquela classificação e avaliá-los à luz da respectiva experiência profissional.

Por outro lado, em muitas aplicações o mais importante é otimizar a taxa de acerto ou a precisão do modelo, mesmo que isso implique certa perda de clareza, de simplicidade ou de desempenho do modelo. Redes Neurais e Máquinas de Vetor de Suporte são duas ilustrações de técnicas que podem oferecer alta precisão, mas que utilizam modelos de classificação difíceis de entender. É comum o uso dessas duas técnicas na área de aplicações financeiras, porque investidores geralmente estão mais interessados no ganho obtido diariamente na aplicação mais bem classificada do que no modelo matemático explicativo. Por esta razão, a decisão de usar um **modelo orientado ao conhecimento** ou um **modelo tipo caixa-preta** deve ser feita caso a caso.

Para a tarefa de Classificação, os dois modelos orientados ao conhecimento mais comuns de representação são Árvores de Decisão e Regras de Classificação. Ambos são logicamente equivalentes e permitem que a partir de uma Árvore de Decisão seja possível obter as correspondentes Regras de Classificação, e vice-versa, embora a obtenção de Árvores a partir de Regras seja um processo mais complexo. Há, porém, vantagens e desvantagens observadas durante a geração desses modelos, que serão discutidas mais a frente.

A Figura 3.2 apresenta modelos simplificados de uma **Árvore de Decisão** e das correspondentes **Regras de Classificação** para o caso da flor Íris. Com estes modelos simplificados, os erros de classificação ilustrados na Figura 3.1 novamente se repetiriam aqui. Para reduzir a taxa de erros, veremos que será necessário usar métodos de

[1] Fonte: Wikimedia Commons - Iris virginica (Acessado em 03.03.26).
[2] Fonte: Wikimedia Commons - Iris versicolor (Acessado em 03.03.26).
[3] Fonte: Wikimedia Commons - Iris setosa (Acessado em 03.03.26).

aprendizado mais refinados ou complexos.

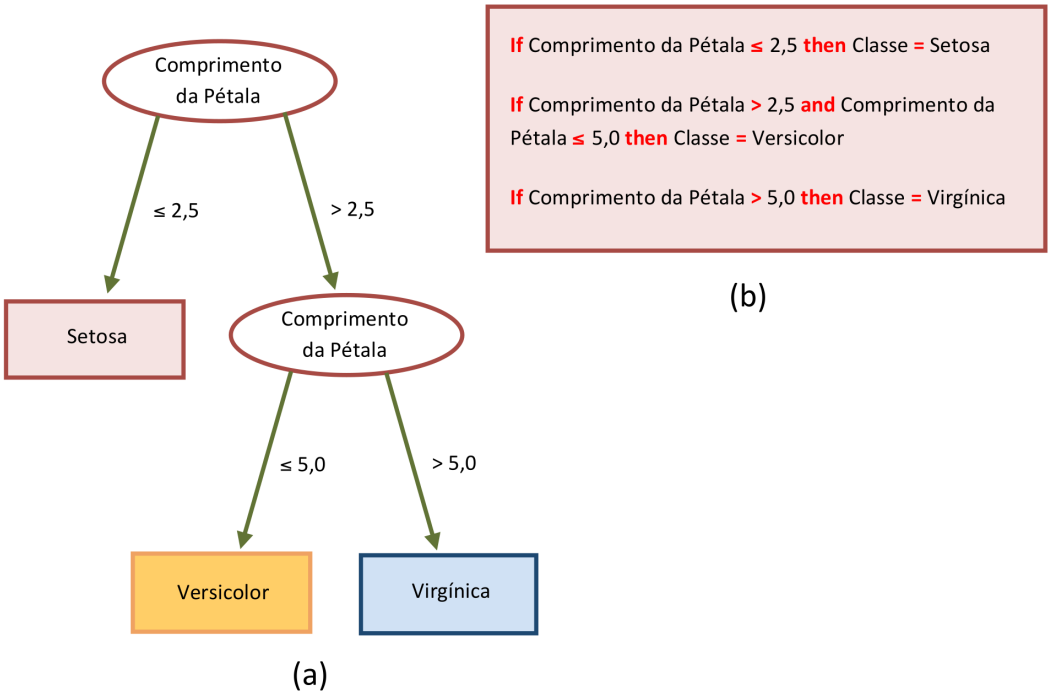


Figura 3.2: Modelos Equivalentes: (a) Árvore de Decisão e (b) Regras de Classificação.

Dependendo da representação desejada, diferentes métodos de inferência serão usados sobre os dados. Mesmo que um modelo faça classificação com erros, é importante observar que cada Exemplo sempre pertencerá a uma única classe.

3.3 Árvores de Decisão

Numa Árvore de Decisão cada **atributo** é representado por um **nó de decisão**, cuja função é testar o valor desse atributo. Uma **classe** é representada por um **nó folha**, que reúne todos os Exemplos que chegarem a ele depois de satisfazerem os testes dos nós de decisão intermediários. Portanto, numa Árvore de Decisão, a classificação de um Exemplo desconhecido implica percorrer toda a árvore a partir de um **nó raiz**, testando atributos em sucessivos **nós internos** até chegar a um **nó folha**, que lhe atribuirá uma **classe**. O objetivo de uma Árvore de Decisão é retornar uma classe para um Exemplo desconhecido.

3.4 Indução de Árvores de Decisão

Uma Árvore de Decisão pode ser construída de forma recursiva, dividindo sucessivamente o conjunto de atributos em subconjuntos. Primeiramente escolhemos um elemento do conjunto de atributos para ser o nó raiz e adicionamos uma aresta para cada um dos possíveis valores que este atributo pode assumir. A seguir, repetimos o processo em cada uma das arestas com os atributos restantes até que todos os Exemplos considerados pertençam à mesma classe.

Gerar uma Árvore de Decisão com escolha aleatória dos atributos não é difícil, porém dependendo da ordem dos atributos, diferentes árvores serão geradas. Isso significa que uma mesma Base de Dados pode produzir muitas Árvores de Decisão funcionalmente equivalentes, mas com tamanhos distintos. Como estamos interessados nos modelos mais compactos, é interessante encontrar critérios que ajudem a decidir sobre a ordem em que os atributos devem aparecer na Árvore de Decisão.

Para ilustrar esta questão, vamos utilizar como caso de estudo novamente a clássica Tabela do Tempo (Tabela 3.1), introduzida por (Quinlan, 1986), tendo como atributos de entrada “Dia”, “Temperatura”, “Umidade” e ”Vento”, e como atributo de saída (ou classe) “Partida”. Para ser mais preciso, na realidade temos duas classes “Sim” e “Não”, e queremos construir uma Árvore de Decisão que represente de forma compacta os Exemplos contidos nessa

tabela. A seguir, com a Árvore de Decisão obtida, dado um novo Exemplo, vamos tentar predizer se vai ou não haver "Partida" num determinado dia, sendo a resposta a combinação linear dos atributos de entrada.

Tabela 3.1: Tabela do Tempo.

Dia	Temperatura	Umidade	Vento	Partida
Ensolarado	Elevada	Alta	Falso	**Não**
Ensolarado	Elevada	Alta	Verdadeiro	**Não**
Nublado	Elevada	Alta	Falso	**Sim**
Chuvoso	Amena	Alta	Falso	**Sim**
Chuvoso	Baixa	Normal	Falso	**Sim**
Chuvoso	Baixa	Normal	Verdadeiro	**Não**
Nublado	Baixa	Normal	Verdadeiro	**Sim**
Ensolarado	Amena	Alta	Falso	**Não**
Ensolarado	Baixa	Normal	Falso	**Sim**
Chuvoso	Amena	Normal	Falso	**Sim**
Ensolarado	Amena	Normal	Verdadeiro	**Sim**
Nublado	Amena	Alta	Verdadeiro	**Sim**
Nublado	Elevada	Normal	Falso	**Sim**
Chuvoso	Amena	Alta	Verdadeiro	**Não**

Inicialmente vamos considerar separadamente para o nó raiz cada um dos quatro atributos possíveis e ver como o atributo de saída "Partida" se divide em "Sim" e "Não". Na Tabela 3.2 é ressaltado o atributo "Dia", na Tabela 3.3, "Temperatura", na Tabela 3.4, "Umidade", e na Tabela 3.5, "Vento".

Como estamos interessados em construir uma árvore compacta, dentre os quatro atributos candidatos para nó raiz, o atributo "Dia" parece o mais promissor porque dentre as três arestas que teremos de colocar neste nó ("Ensolarado", "Nublado" e "Chuvoso"), a aresta para "Nublado" tem **todos** seus elementos pertencentes à mesma classe "Sim" e, portanto, esta aresta da Árvore de Decisão termina aqui com um nó folha "Sim".

Tabela 3.2: Dia

Dia	Partida
Ensolarado	**Sim**
Ensolarado	**Sim**
Ensolarado	**Não**
Ensolarado	**Não**
Ensolarado	**Não**
Nublado	**Sim**
Nublado	**Sim**
Nublado	**Sim**
Nublado	**Sim**
Chuvoso	**Sim**
Chuvoso	**Sim**
Chuvoso	**Sim**
Chuvoso	**Não**
Chuvoso	**Não**

Tabela 3.3: Temperatura

Temperatura	Partida
Elevada	**Sim**
Elevada	**Sim**
Elevada	**Não**
Elevada	**Não**
Amena	**Sim**
Amena	**Sim**
Amena	**Sim**
Amena	**Não**
Amena	**Não**
Baixa	**Sim**
Baixa	**Sim**
Baixa	**Sim**
Baixa	**Não**
Baixa	**Não**

Tabela 3.4: Umidade

Umidade	Partida
Alta	**Sim**
Alta	**Sim**
Alta	**Sim**
Alta	**Não**
Alta	**Não**
Alta	**Não**
Alta	**Não**
Normal	**Sim**
Normal	**Sim**
Normal	**Sim**
Normal	**Sim**
Normal	**Sim**
Normal	**Sim**
Normal	**Não**

Tabela 3.5: Vento

Vento	Partida
Falso	**Sim**
Falso	**Sim**
Falso	**Sim**
Falso	**Sim**
Falso	**Sim**
Falso	**Sim**
Falso	**Não**
Falso	**Não**
Verdadeiro	**Sim**
Verdadeiro	**Sim**
Verdadeiro	**Sim**
Verdadeiro	**Não**
Verdadeiro	**Não**
Verdadeiro	**Não**

A Figura 3.3 ilustra esta primeira iteração na construção de uma Árvore de Decisão compacta.

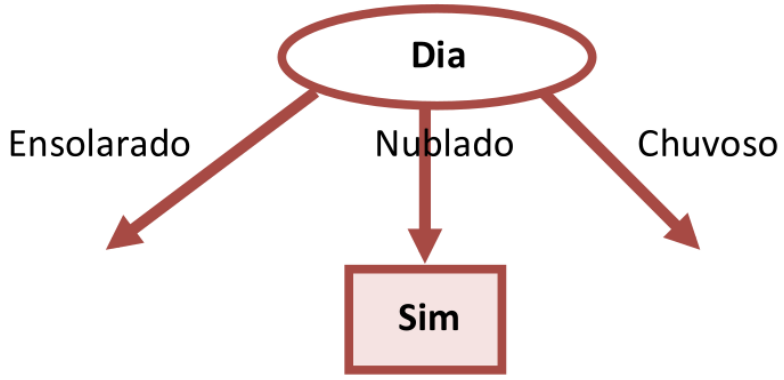


Figura 3.3: Nó raiz para os dados do Tempo.

Como nas arestas "Ensolarado" e "Chuvoso" há elementos tanto da classe "Sim" como da classe "Não" (veja Tabela 3.2), outro atributo deve ser escolhido para cada aresta, e assim sucessivamente até que todos os elementos de um ramo pertençam a uma mesma classe. Como restam os atributos "Temperatura", "Umidade" e "Vento", analisando a Tabela 3.1, vamos testar cada um deles em combinação com a aresta "Ensolarado".

As Tabelas 3.6, 3.7 e 3.8 mostram as combinações possíveis de "Dia=Ensolarado" com "Temperatura", "Umidade" e "Vento". Aqui também percebemos que "Umidade" parece ser a escolha mais promissora porque todos os elementos de "Umidade=Alta" correspondem à classe "Não" e todos os elementos com "Umidade=Normal" pertencem à classe "Sim". Portanto, temos mais dois nós folhas aqui, favorecendo a construção de uma árvore mais compacta.

Tabela 3.6: Temperatura

Dia	Temp.	Partida
Ensolarado	Elevada	**Não**
Ensolarado	Elevada	**Não**
Ensolarado	Amena	**Sim**
Ensolarado	Amena	**Não**
Ensolarado	Baixa	**Sim**

Tabela 3.7: Umidade

Dia	Umidade	Partida
Ensolarado	**Alta**	**Não**
Ensolarado	**Alta**	**Não**
Ensolarado	**Alta**	**Não**
Ensolarado	**Normal**	**Sim**
Ensolarado	**Normal**	**Sim**

Tabela 3.8: Vento

Dia	Vento	Partida
Ensolarado	Falso	**Sim**
Ensolarado	Falso	**Não**
Ensolarado	Falso	**Não**
Ensolarado	Verdadeiro	**Sim**
Ensolarado	Verdadeiro	**Não**

A Figura 3.4 ilustra a segunda iteração do algoritmo com mais dois nós folhas, dando por completa esta região da Árvore de Decisão.

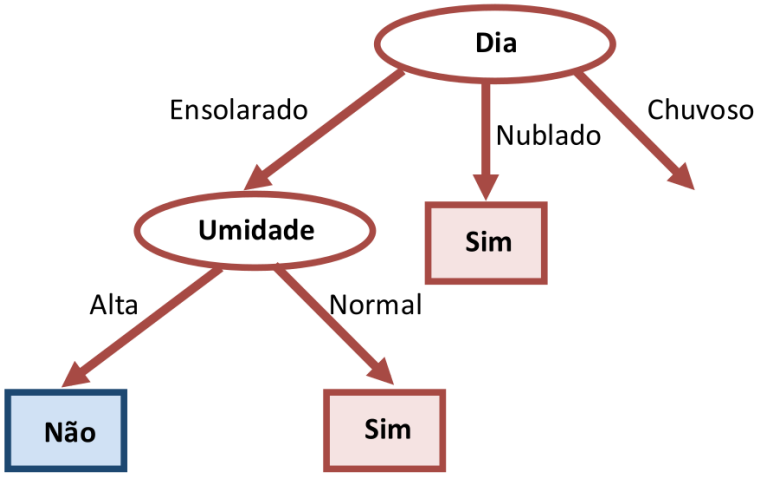


Figura 3.4: O atributo "Umidade" combinado com "Dia".

Para a terceira aresta, ou seja, "Dia=Chuvoso", restam duas alternativas agora: "Temperatura" e "Vento". Vamos construir as tabelas de combinação para descobrir a mais interessante. As Tabelas 3.9 e 3.10 ilustram as possíveis combinações.

Tabela 3.9: Dia e Temperatura

Dia	Temperatura	Partida
Chuvoso	Baixa	**Não**
Chuvoso	Baixa	**Sim**
Chuvoso	Amena	**Sim**
Chuvoso	Amena	**Sim**
Chuvoso	Amena	**Não**

Tabela 3.10: Dia e Vento

Dia	Vento	Partida
Chuvoso	**Falso**	**Sim**
Chuvoso	**Falso**	**Sim**
Chuvoso	**Falso**	**Sim**
Chuvoso	**Verdadeiro**	**Não**
Chuvoso	**Verdadeiro**	**Não**

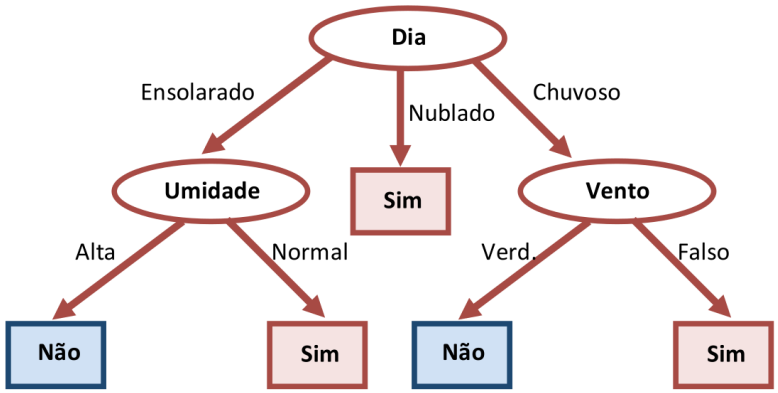


Figura 3.5: Árvore de Decisão para os Dados da Tabela do Tempo.

O critério de escolha do melhor atributo para cada iteração no algoritmo ID3, criado por (Quinlan, 1986), é medido pela significância estatística, que em nosso caso se expressa pela proporção de "Sim"s e "Não"s no atributo de saída "Partida". Como foi ilustrado anteriormente, é mais promissor escolher um atributo que tenha associado a ele respostas compostas unicamente por "Sim"s ou "Não"s porque neste caso podemos colocar um nó folha correspondente e terminar com as subdivisões. Em outras palavras, quanto mais compacta uma árvore, menos testes serão necessários para classificar um Exemplo. Por outro lado, se o conjunto de respostas é composto por uma mescla de "Sim"s e "Não"s, então faz-se necessário colocar mais um nó interno, com um novo atributo sendo testado, implicando um crescimento da Árvore de Decisão.

De acordo com a fórmula de Shannon, em uma tabela como a Tabela 3.1 a quantidade de informação presente é,

$$Info(Tabela) = -\sum_{i=1}^{N} p_i \log_2 p_i$$

sendo p_i a proporção de "Sim"s e "Não"s associados a um atributo (a quantidade de informação ou entropia é medida em bits, ou frações de bits!). Por ex., na Tabela 3.1 temos apenas duas classes ("Sim" e "Não"), sendo que dos 14 Exemplos, 9 pertencem à classe "Sim" e 5 à classe "Não". Portanto, a quantidade de informação associada a esta tabela pode ser calculada da seguinte forma,

$$\text{Info}\,(Tab.3.1) = (-9/14 \log_2 9/14) + (-5/14 \log_2 5/14) = 0,94 bits$$

Uma forma alternativa de interpretar estes números, é pensar que estamos interessados em medir o grau de "impureza" de um conjunto de respostas. Se todas as respostas forem apenas "Sim" ou apenas "Não", então o grau de impureza do conjunto é 0. Por outro lado, se tivermos 10% de "Sim"s e 90% de "Não"s, então o grau de impureza seria,

$$Info(Tab\_(10/90)) = (-1/10 \log_2 1/10) + (-9/10 \log_2 9/10) = 0,47 bits$$

De acordo com este raciocínio, o grau de impureza máxima é representado pela proporção 50% de "Sim"s e 50% de "Não"s, sendo $Info(Tab\_(50/50) = 1 bit)$.

Voltando ao nosso problema original de escolha do atributo mais promissor em cada iteração do algoritmo, vamos calcular o grau de impureza da Tabela 3.2, que se refere ao atributo "Dia". Esse atributo se subdivide em três alternativa possíveis, com as seguintes proporções de "Sim"s e "Não"s: "Ensolarado" (2"Sim"/3"Não)","Nublado" (4"Sim"/0"Não) e "Chuvoso" (3"Sim"/2"Não"). Portanto, seu grau de impureza é,

$$Info(Ensolarado) = (-2/5 \log_2 2/5) + (-3/5 \log_2 3/5) = 0,97bits$$

$$Info(Nublado) = (-4/4 \log_2 4/4) + (-0/4 \log_2 0/4) = 0,00bits$$

$$Info(Chuvoso) = (-3/5 \log_2 3/5) + (-2/5 \log_2 2/5) = 0,97bits$$

Fazendo a soma ponderada de cada uma dessas alternativas sobre os 14 Exemplos, resulta,

$$Info\,(Dia) = 0,97 * \frac{5}{14} + 0 * \frac{4}{14} + 0,97 * \frac{5}{14} = \ 0,69bits$$

Aplicando-se raciocínio semelhante para os atributos "Temperatura", "Umidade" e "Vento" obtêm-se os seguintes valores,

$$Info\,(Temperatura) = 0.91bits$$

$$Info\,(Umidade) = 0.79bits$$

$$Info\,(Vento) = 0.89bits$$

Portanto, dos quatro atributos possíveis na primeira iteração, o atributo "Dia" é o que tem o grau mais baixo de impureza, e , portanto, é o mais promissor para construir uma Árvore de Decisão Compacta. Continuando este procedimento recursivamente, chega-se a Árvore de Decisão apresentada na Figura 3.5.

Há mais sutilezas matemáticas envolvidas que não foram mencionadas, e outros detalhes importantes do algoritmo ID3 precisariam ser abordados se nossa intenção fosse explicar seu funcionamento. Porém, o que pretendemos aqui é apenas dar uma ideia de seu embasamento teórico para que ao nos depararmos com uma ferramenta que implemente este algoritmo seja possível entender o resultado de seus cálculos.

3.5 Árvore de Decisão Não Compacta

Para efeito comparativo, vamos supor que algum critério arbitrário de escolha da ordem dos atributos tenha sido utilizado e que o nó raiz contenha o atributo "Umidade". A Tabela 3.11 mostra que este atributo possui Exemplos misturados pertencentes a classes distintas, portanto é necessário um novo teste, i.e., escolher um novo atributo para teste. A Figura 3.6 mostra o resultado dessa escolha arbitrária.

O atributo escolhido arbitrariamente agora foi "Dia" e as combinações possíveis com "Umidade" são mostradas na Tabela 3.12. A segunda iteração do algoritmo para a construção da Árvore de Decisão Alternativa é mostrada na Figura 3.7.

Embora as arestas de "Dia=Ensolarado" e "Dia=Nublado" terminem em nó folha, a aresta para "Dia=Chuvoso" exige um novo teste já que há duas respostas distintas possíveis. O próximo atributo escolhido arbitrariamente foi "Vento", como mostra a Tabela 3.13. A ilustração da Figura 3.8 ajuda a entender como a falta de uma rotina de otimização produz árvores desnecessariamente grandes.

Tabela 3.11: Umidade (arbitrária).

Umidade	Partida
Alta	**Sim**
Alta	**Sim**
Alta	**Sim**
Alta	**Não**
Alta	**Não**
Alta	**Não**
Alta	**Não**
Normal	**Sim**
Normal	**Sim**
Normal	**Sim**
Normal	**Sim**
Normal	**Sim**
Normal	**Sim**
Normal	**Não**

Figura 3.6: Árvore de Decisão com Nó Raiz Arbitrário.

Tabela 3.12: Dia e Umidade (arbitrários).

Dia	Umidade	Partida
Ensolarado	Alta	**Não**
Ensolarado	Alta	**Não**
Ensolarado	Alta	**Não**
Nublado	Alta	**Sim**
Nublado	Alta	**Sim**
Chuvoso	Alta	**Sim**
Chuvoso	Alta	**Não**

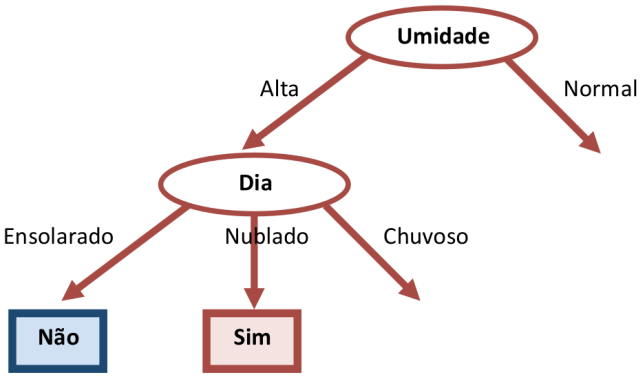


Figura 3.7: Segunda Iteração da Árvore de Decisão Alternativa.

Tabela 3.13: Dia, Umidade e Vento (arbitrários).

Dia	Umidade	Vento	Partida
Chuvoso	Alta	Falso	**Sim**
Chuvoso	Alta	Verdadeiro	**Não**

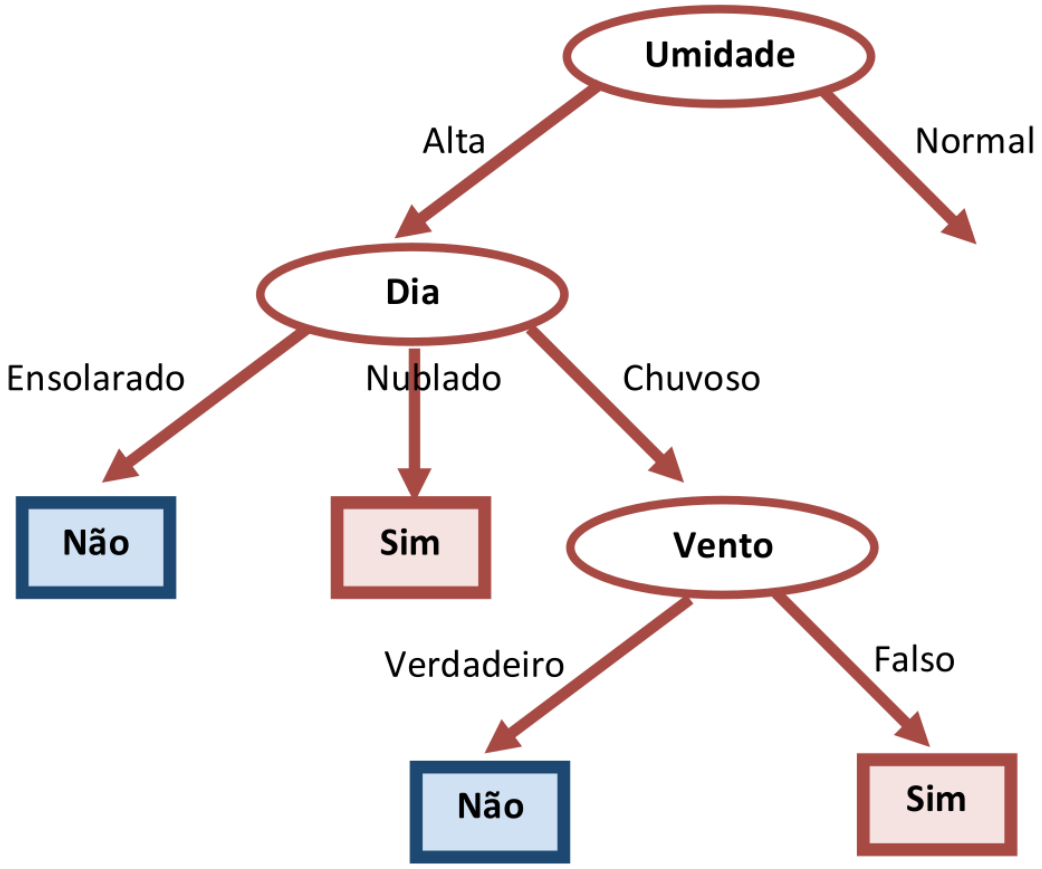


Figura 3.8: Árvore de Decisão sem Otimização.

A Tabela 3.13 mostra que esta região da Árvore de Decisão está encerrada, com dois novos nós folhas. Vamos agora considerar a aresta correspondente a "Umidade=Normal" (Tabela 3.14) e supor que o novo teste escolhido será o atributo "Dia". O resultado da escolha do atributo "Dia" para a aresta de "Umidade=Normal" é mostrado na Figura 3.9.

Tabela 3.14: Dia e Umidade (arbitrários).

Dia	Umidade	Partida
Ensolarado	Normal	**Sim**
Ensolarado	Normal	**Sim**
Nublado	Normal	**Sim**
Nublado	Normal	**Sim**
Chuvoso	Normal	**Sim**
Chuvoso	Normal	**Sim**
Chuvoso	Normal	**Não**

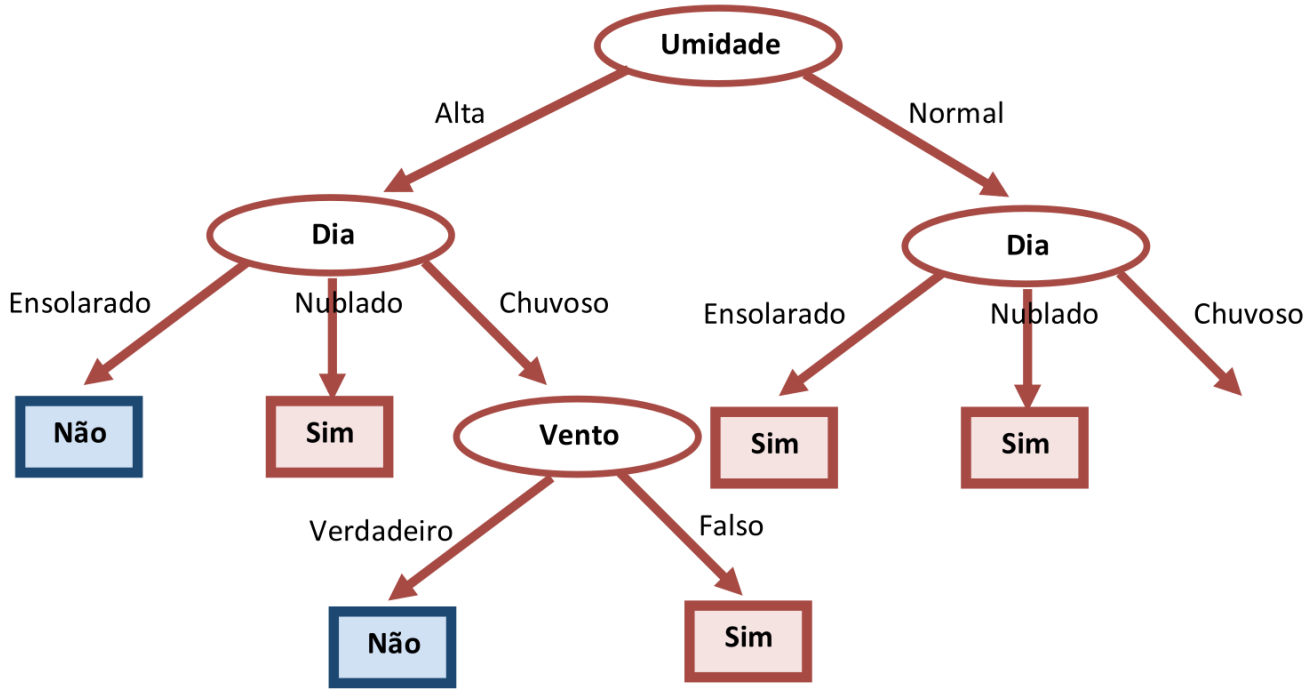


Figura 3.9: Atributo “Dia” Usados em Duas Posições Diferentes.

Como a opção de “Dia=Chuvoso” exige um novo teste, vamos supor que o atributo escolhido tenha sido “Vento”, produzindo o resultado mostrado na Tabela 3.15. O resultado final da Árvore de Decisão Não-Compacta é mostrado na Figura 3.10.

Tabela 3.15: Dia, Umidade e Vento (arbitrários).

Dia	Umidade	Vento	Partida
Chuvoso	Normal	Falso	**Sim**
Chuvoso	Normal	Falso	**Sim**
Chuvoso	Normal	Verdadeiro	**Não**

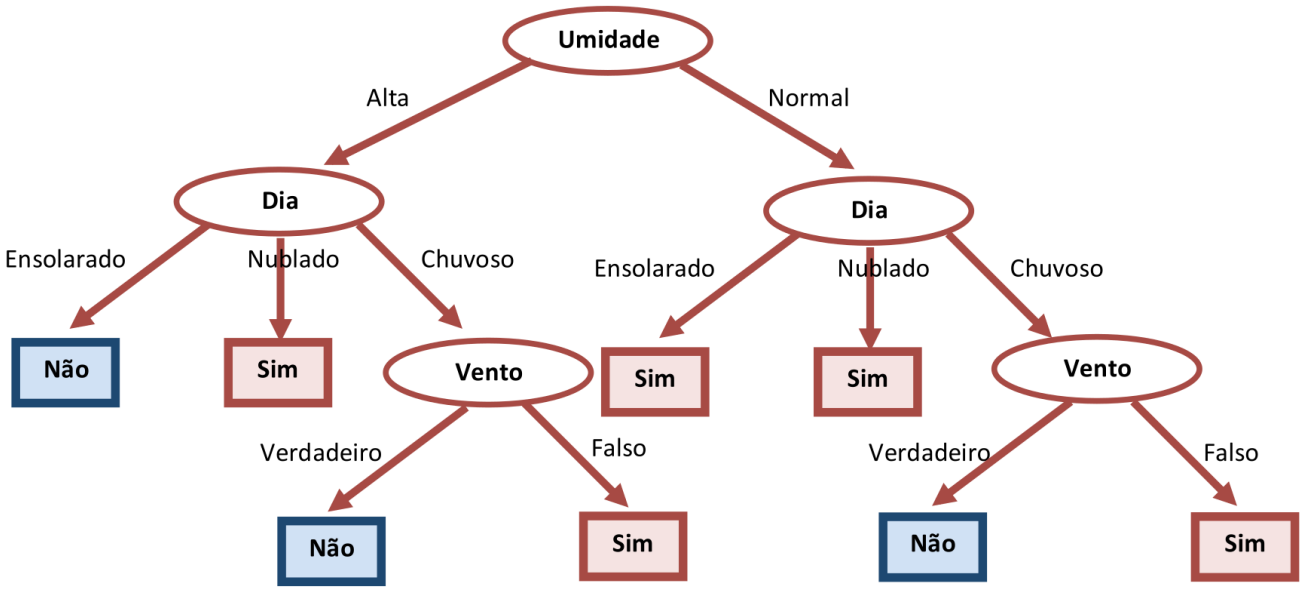


Figura 3.10: Árvore de Decisão Não-Compacta para a Tabela do Tempo.

Com este teste, o algoritmo se encerra já que todos os Exemplos foram devidamente considerados e se encaixaram num dos caminhos possíveis da Árvore de Decisão.

são irrelevantes, é da maior importância para a pesquisa médica. Em outras áreas, a modelagem descritiva pode ser igualmente interessante. Pense, por exemplo, na importância em entender o comportamento dos frequentadores de determinado estabelecimento comercial, ou no aumento de lucro que alguém pode obter ao traçar o perfil de consumo de um segmento social.

3.7 Regras de Classificação a partir de uma Árvore de Decisão

A geração de Regras de Classificação a partir de uma Árvore de Decisão é feita percorrendo desde o nó raiz até um nó folha, anotando a conjunção de condições representadas pelos nós internos. A cada classe da Árvore de Decisão corresponde uma Regra de Classificação, sendo ambas logicamente equivalentes.

Vamos reproduzir a Árvore de Decisão da Figura 3.5 para ilustrar esse processo de geração de Regras de Classificação a partir de uma Árvore de Decisão.

Partindo-se do nó raiz "Dia", seguindo pela aresta correspondente à condição "Ensolarado", passando pelo nó interno "Umidade" e, finalmente, tomando a aresta "Alta", chega-se ao nó folha "Não". Dessa forma, podemos gerar a primeira regra relativa à classe "Não", como ilustra a Regra 3.1:

$$\textbf{If } (\text{Dia} = \text{Ensolarado}) \textbf{ and } (\text{Umidade} = \text{Alta}) \textbf{ then } (\text{Partida} = \text{Não}) \tag{3.1}$$

Esta regra foi construída como uma conjunção lógica ("*AND*") de duas condições lógicas. Por inspeção na Árvore de Decisão da Figura 3.5 verifica-se que há outro caminho que pode levar à classe "Não", que pode ser representada pela Regra 3.2:

$$\textbf{If } (\text{Dia} = \text{Chuvoso}) \textbf{ and } (\text{Vento} = \text{Verdadeiro}) \textbf{ then } (\text{Partida} = \text{Não}) \tag{3.2}$$

As duas Regras [-@-eq3-1] e 3.2, formadas por conjunções ("*AND*"), podem ser fundidas numa única regra utilizando-se uma disjunção lógica ("*OR*"), como mostra a Regra 3.3:

$$\begin{aligned} &\textbf{If } [(\text{Dia} = \text{Ensolarado}) \textbf{ and } (\text{Umidade} = \text{Alta})] \\ &\textbf{or } [(\text{Dia} = \text{Chuvoso}) \textbf{ and } (\text{Vento} = \text{Verdadeiro})] \\ &\textbf{then } (\text{Partida} = \text{Não}) \end{aligned} \tag{3.3}$$

Regras com estrutura lógica semelhante à Regra 3.3 são conhecidas como regras na forma disjunção de conjunções. Aplicando-se o mesmo procedimento descrito para os casos restantes, obtém-se a Regra de Classificação correspondente à classe "Sim", representada pela Regra 3.4:

$$\begin{aligned} &\textbf{If } [(\text{Dia} = \text{Ensolarado}) \textbf{ and } (\text{Umidade} = \text{Normal})] \textbf{ or } [(\text{Dia} = \text{Nublado})] \\ &\textbf{or } [(\text{Dia} = \text{Chuvoso}) \textbf{ and } (\text{Vento} = \text{Falso})] \\ &\textbf{then } (\text{Partida} = \text{Sim}) \end{aligned} \tag{3.4}$$

3.8 Treinamento, Aprendizado e Classificação

Recapitulando o que foi feito até aqui, inicialmente consideramos a existência de uma Base de Dados codificada na forma de uma tabela com vários atributos. A seguir, apresentamos um algoritmo capaz de construir uma Árvore de Decisão a partir desses dados estruturados. Como cada Exemplo da Base de Dados era composto por alguns atributos e um rótulo de classe (ou atributo de saída), nós podemos entender todo este processo de construção de Árvores de Decisão como sendo um processo de **Aprendizado Supervisionado** por meio de **Treinamento**.

Diz-se que o **Aprendizado** é **Supervisionado** quando cada **Exemplo** usado no **Treinamento** possui um **rótulo de classe** que orienta (ou otimiza) um mecanismo de reforço ou penalização, ou seja, quando já se sabe antecipadamente a qual classe determinado elemento pertence. Muitas vezes os Exemplos não trazem o rótulo de classe e ainda assim é possível ocorrer aprendizado, porém, nestes casos diz-se que o Aprendizado é Não-Supervisionado. Por ex., com um conjunto de Exemplos sem rótulos de classe é possível agrupá-los de acordo com algum critério, como o critério de afinidade.

A Árvore de Decisão resultante representa o Modelo gerado ou o Conceito aprendido. Note que para cada Base de Dados será construída uma Árvore de Decisão que lhe corresponde. Ou seja, nós temos um algoritmo que em princípio gera árvores genéricas, que refletem a estrutura dos dados utilizados no treinamento. Como a Árvore de Decisão gerada poderá ser usada para diagnosticar doenças, classificar produtos comerciais, ou explicar a correlação de fatores de um fenômeno, podemos dizer que o Sistema Inteligente que emprega este tipo de algoritmo de aprendizado melhora seu desempenho a partir de sua própria experiência (ou treinamento). A Figura 3.11 ilustra as três fases de um Sistema Inteligente Classificatório.

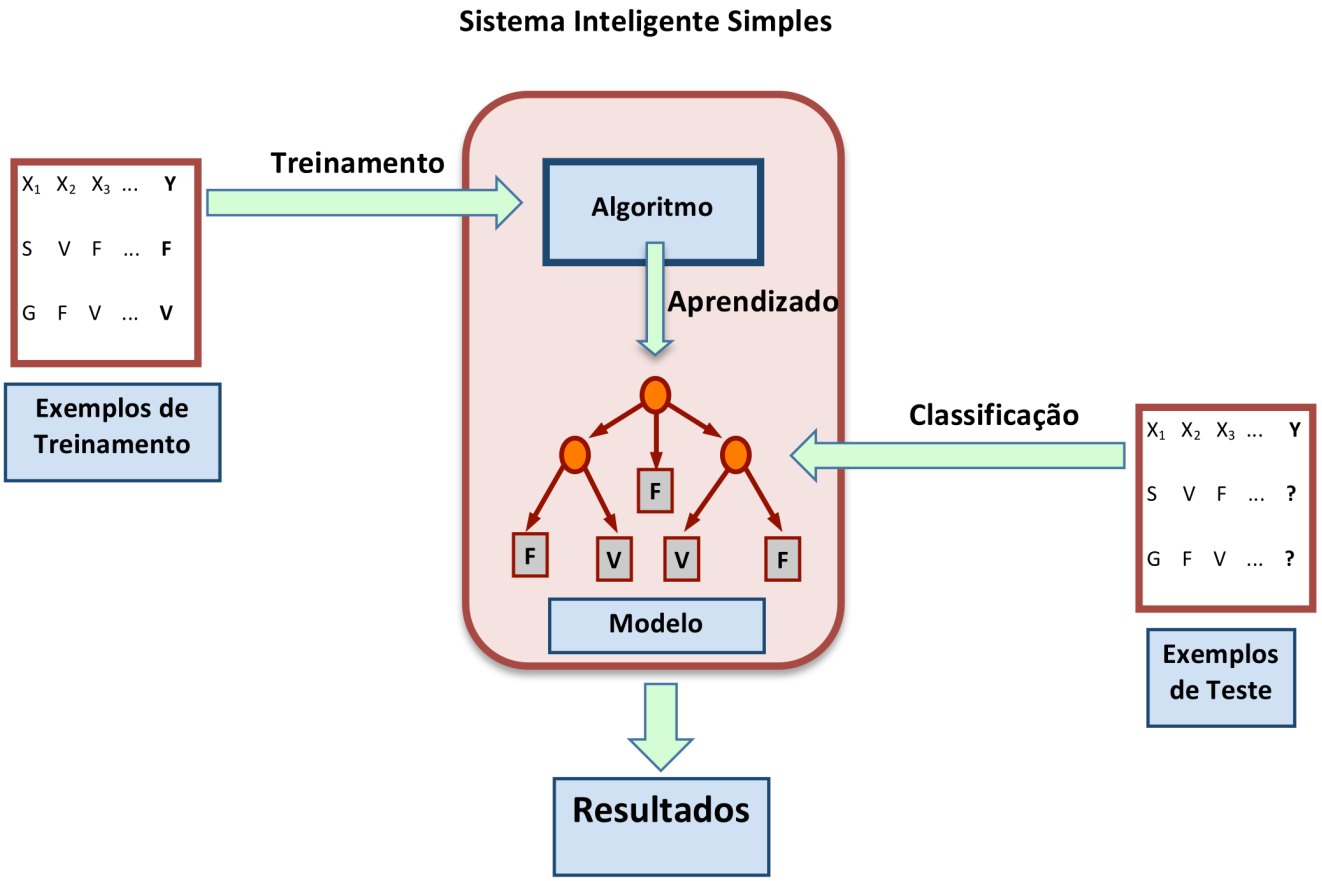


Figura 3.11: Treinamento, Aprendizado e Classificação em um Sistema Inteligente Simples.

O Treinamento viabiliza o Aprendizado de um Conceito pelo Sistema Inteligente, e a aquisição de um Conceito pelo sistema elimina a necessidade de um aprendizado constante. O processo de Aprendizado pode ser demorado, mas uma vez aprendido um Conceito, o processo de Classificação é bem mais rápido. A Figura 3.11 não mostra outros caminhos ou fluxos de trabalho do sistema para a obtenção do Modelo desejado de Árvore de Decisão. É comum que após a geração do primeiro modelo de árvore, os resultados obtidos não sejam satisfatórios. Nesse caso, nova rodada de treinamento se inicia, com novos parâmetros e ajustes adicionais, até a geração de um segundo modelo. Este processo se repete de forma iterativa e interativa até que se chegue a uma Árvore de Decisão com as características buscadas.

Tendo descrito o processo de criação ou indução de uma Árvore de Decisão, vamos ver agora que resultados de classificação podemos obter com esta árvore. A Tabela 3.16 apresenta alguns novos Exemplos que usaremos para teste.

Tabela 3.16: Tabela do Tempo com Exemplos de Teste.

Dia	Temperatura	Umidade	Vento	Partida
Ensolarado	Elevada	Alta	Falso	**Não**
Ensolarado	Elevada	Alta	Verdadeiro	**Não**

Dia	Temperatura	Umidade	Vento	Partida
Nublado	Elevada	Alta	Falso	**Sim**
Chuvoso	Amena	Alta	Falso	**Sim**
Chuvoso	Baixa	Normal	Falso	**Sim**
Chuvoso	Baixa	Normal	Verdadeiro	**Não**
Nublado	Baixa	Normal	Verdadeiro	**Sim**
Ensolarado	Amena	Alta	Falso	**Não**
Ensolarado	Baixa	Normal	Falso	**Sim**
Chuvoso	Amena	Normal	Falso	**Sim**
Ensolarado	Amena	Normal	Verdadeiro	**Sim**
Nublado	Amena	Alta	Verdadeiro	**Sim**
Nublado	Elevada	Normal	Falso	**Sim**
Chuvoso	Amena	Alta	Verdadeiro	**Não**
Ensolarado	Amena	Normal	Falso	**????**
Ensolarado	Baixa	Alta	Verdadeiro	**????**
Nublado	Baixa	Alta	Verdadeiro	**????**
Chuvoso	Elevada	Normal	Falso	**????**

Usando tanto a Árvore de Decisão da Figura 3.5 quanto a da Figura 3.10, o resultado para o primeiro Exemplo de Teste, aquele que se inicia com "Dia=Ensolarado" e "Temperatura=Amena", é "Sim", para o segundo Exemplo, com "Dia=Ensolarado" e "Temperatura=Baixa" , a resposta é "Não", para o terceiro Exemplo "Dia=Nublado", a resposta é "Sim" e finalmente para o quarto Exemplo, com "Dia=Chuvoso", a resposta é "Sim".

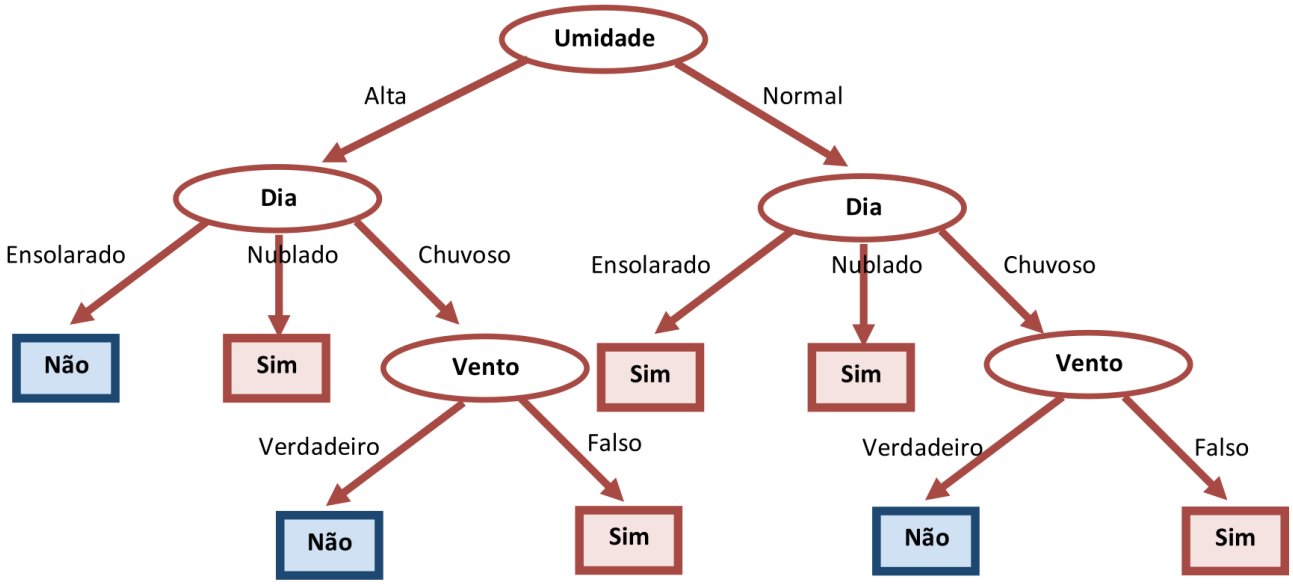
3.9 *Overfitting* e Poda

É possível que para alguns Exemplos de Teste os resultados produzidos por Árvores de Decisão compactas, como a da Figura 3.5, não sejam os mesmos de Árvores de Decisão não-compactas, como a da Figura 3.10, muito embora os resultados para todos os Exemplos de Treinamento tenham sido os mesmos. O que pode ocorrer com árvores não compactas é que algumas das arestas refletem um superajuste (*overfitting*) aos Exemplos de Treinamento. Se neste Conjunto de Treinamento houver ruído ou *outliers*, a estrutura resultante da Árvore de Decisão pode não refletir às relações essenciais entre os atributos da Base de Dados.

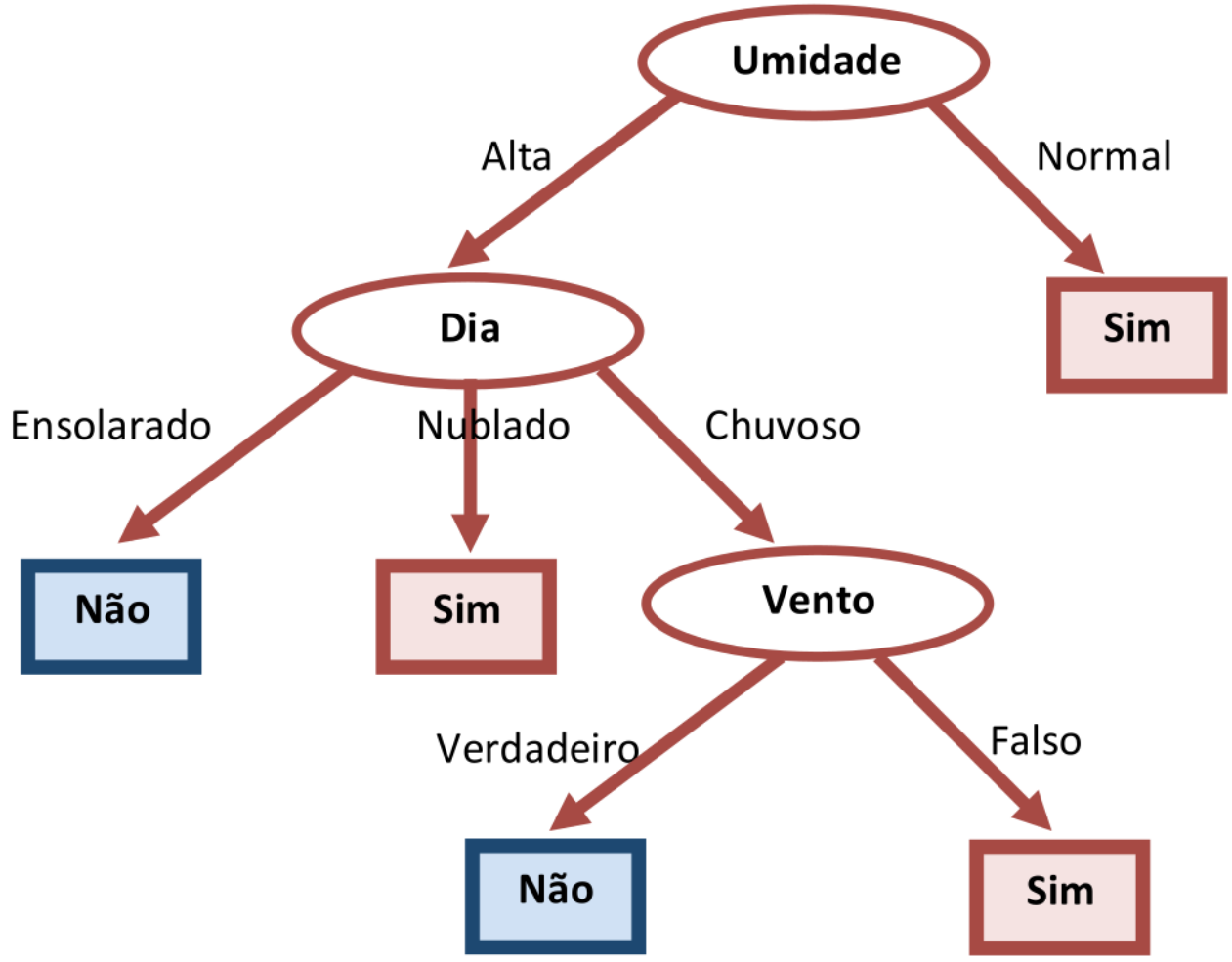
Para evitar o *overfitting*, muitos algoritmos se valem de uma técnica conhecida como "Poda", que consiste em eliminar algumas arestas da Árvore de Decisão com base em medidas estatísticas dos Exemplos. A Poda pode ocorrer sobre uma Árvore de Decisão concluída, com a eliminação de algumas arestas consideradas não necessárias, ou durante a construção da Árvore de Decisão, com a introdução precoce de um nó folha em arestas com baixa importância estatística, por exemplo.

A Árvore de Decisão da Figura 3.10 poderia ter algumas de suas arestas removidas sem comprometer seriamente a taxa de erros na classificação. De fato, observando-se a Tabela 3.4, verifica-se que dos sete Exemplos que apresentam "Umidade=Normal", seis deles têm rótulo de classe "Sim". Por esta razão, o nó interno à direita da Figura 3.10, (reproduzida na Figura 3.12a), representando o atributo "Vento", poderia ser eliminado e substituído por um nó folha "Sim" já que a maioria dos Exemplos que chegam a este nó apresentam o rótulo de classe "Sim". A seguir, o nó interno "Dia" também pode ser eliminado pois todas seus nós terminais passaram a ser do tipo "Sim". A Figura 3.12b mostra o resultado desse processo de Poda.

Para avaliar quantitativamente o desempenho de um modelo gerado, seja ele uma Árvore de Decisão ou um conjunto de Regras de Classificação, veremos que há técnicas e medidas de desempenho especialmente desenvolvidas para este fim.



(a) Antes da poda.



(b) Depois da poda.

Figura 3.12: Árvore de Decisão Não-Compacta: (a) antes da poda e (b) depois da poda.

3.10 Matriz de Confusão e Avaliação dos Resultados

Ao se gerar uma Árvore de Decisão o que se espera é que ela classifique corretamente Exemplos desconhecidos, mas na prática às vezes verifica-se a ocorrência de classificações equivocadas. Isso também ocorre com o diagnóstico de profissionais especializados. Quando um especialista deseja detectar a presença ou não de uma doença, ele solicita exames laboratoriais para auxiliá-lo a formular um diagnóstico positivo ou negativo sobre esta provável doença.

Se as respostas possíveis para um diagnóstico forem "Positivo" e "Negativo", quatro combinações de resultados previstos e resultados reais podem ocorrer:

1. Se o paciente for portador da doença e o médico acertar no diagnóstico, dizemos que este caso é um **Verdadeiro Positivo (VP)**;
2. Se o paciente não for portador da doença e o médico acertar no diagnóstico, dizemos que este caso é um **Verdadeiro Negativo (VN)**;
3. Se o paciente for portador da doença, e o médico errar no diagnóstico afirmando que ele está são, dizemos que este caso é um **Falso Negativo (FN)**;
4. Se o paciente não for portador da doença, e o médico errar no diagnóstico dizendo que ele está doente, dizemos que este caso é um **Falso Positivo (FP)**.

Essas quatro combinações de resultados costumam ser representadas por uma matriz que recebe o nome de "Matriz de Confusão", como mostra a Tabela 3.17.

Tabela 3.17: Matriz de Confusão.

	Positivo Previsto	Negativo Previsto
Positivo Real	**Verdadeiro Positivo (VP)**	**Falso Negativo (FN)**
Negativo Real	**Falso Positivo (FP)**	**Verdadeiro Negativo (VN)**

Os valores contidos numa Matriz de Confusão podem ser utilizados para avaliar o desempenho de uma Árvore de Decisão. O que se espera nos resultados é que os casos positivos sejam classificados como positivos e os negativos como negativos, ou seja, o desejável é que as taxas de sucesso para Verdadeiro Positivo e Verdadeiro Negativo sejam altas, e que as taxas de Falso Positivo e Falso Negativo sejam baixas.

Fazendo uma relação entre os Exemplos corretamente classificados, i.e., Verdadeiro Positivo (VP) mais Verdadeiro Negativo (VN), com o número total de classificações (VP+VN+FP+FN), podemos definir uma métrica de desempenho para a taxa de acertos ou sucesso, conhecida como Precisão ou Acurácia de uma Árvore de Decisão. Portanto,

$$Acurcia = \frac{VP + VN}{VP + VN + FP + FN} \times 100\%$$

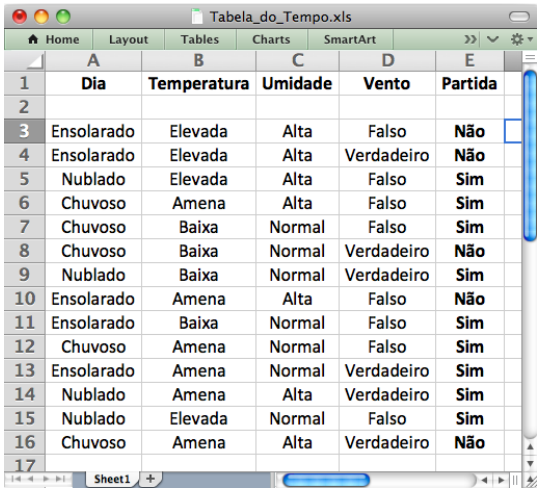
3.11 Como Gerar uma Árvore de Decisão Usando o Weka

A ferramenta Weka (Weka, 2013) permite gerar Árvores de Decisão de forma automática ou interativa. Vamos mostrar a geração automática a partir de um arquivo de entrada.

Passo 1 - Uma forma rápida de criar o arquivo de entrada tipo ".arff" a partir de uma planilha de dados ".xls" (Figura 3.13a) é salvá-la no formato ".csv" (Figura 3.13b), depois abrir este arquivo num editor de texto, acrescentar algumas palavras-chave e salvar novamente (Figura 3.14a). Saia do editor de texto e mude manualmente a extensão do arquivo de ".csv" para ".arff" (Figura 3.14b). Note que as Figuras 3.14a e 3.14b são idênticas, mas a extensão dos arquivos é diferente.

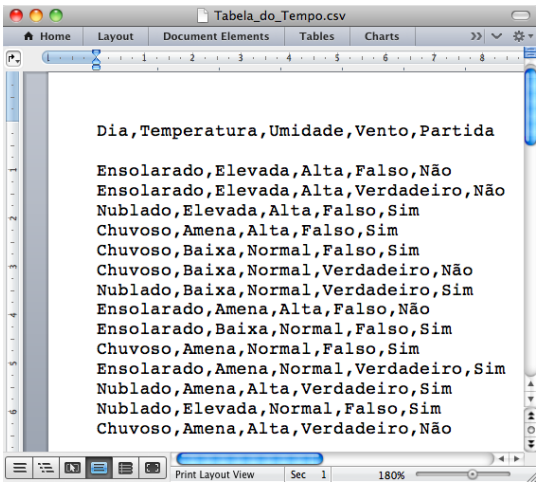
Na unidade anterior, sobre como gerar Regras de Associação no Weka, foi explicado que o Weka aceita trabalhar diretamente o formato ".csv", sem a necessidade de acrescentar palavras-chaves. Mas, em alguns casos a criação do arquivo ".arff" se justifica. Em caso de dúvida sobre esta questão, consulte o material da unidade anterior.

Passo 2 - Dando dois cliques sobre o ícone do arquivo "Tabela_do_Tempo.arff" ele se abrirá dentro do Weka, mostrando uma figura semelhante à Figura 3.15. Alternativamente, é possível primeiramente abrir o Weka Explorer, depois "Open file..." e localizar o arquivo "Tabela_do_Tempo.arff".



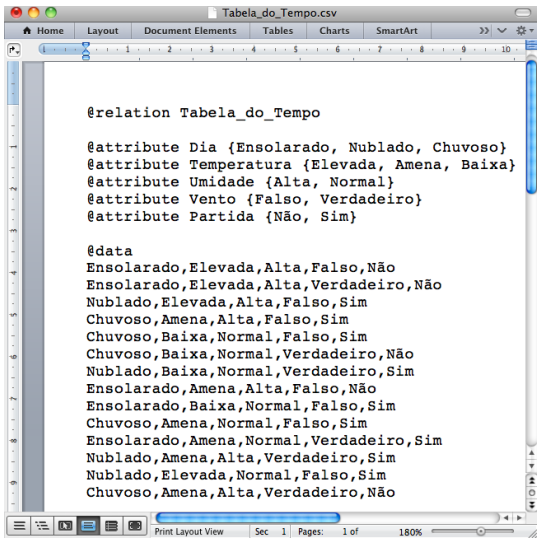
	A	B	C	D	E
1	Dia	Temperatura	Umidade	Vento	Partida
2					
3	Ensolarado	Elevada	Alta	Falso	Não
4	Ensolarado	Elevada	Alta	Verdadeiro	Não
5	Nublado	Elevada	Alta	Falso	Sim
6	Chuvoso	Amena	Alta	Falso	Sim
7	Chuvoso	Baixa	Normal	Falso	Sim
8	Chuvoso	Baixa	Normal	Verdadeiro	Não
9	Nublado	Baixa	Normal	Verdadeiro	Sim
10	Ensolarado	Amena	Alta	Falso	Não
11	Ensolarado	Baixa	Normal	Falso	Sim
12	Chuvoso	Amena	Normal	Falso	Sim
13	Ensolarado	Amena	Normal	Verdadeiro	Sim
14	Nublado	Amena	Alta	Verdadeiro	Sim
15	Nublado	Elevada	Normal	Falso	Sim
16	Chuvoso	Amena	Alta	Verdadeiro	Não
17					

(a) Formato ".xls".

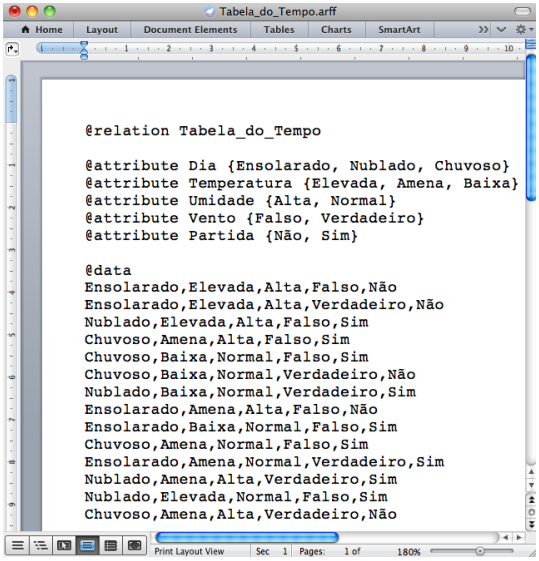


(b) Formato ".csv".

Figura 3.13: Tabela do Tempo: (a) formato ".xls" e (b) formato ".csv".



(a) Formato ".csv" com Palavras-Chave



(b) Arquivo ".arff"

Figura 3.14: Arquivo Tabela_do_Tempo: (a) formato ".csv" com Palavras-Chave e (b) arquivo ".arff".

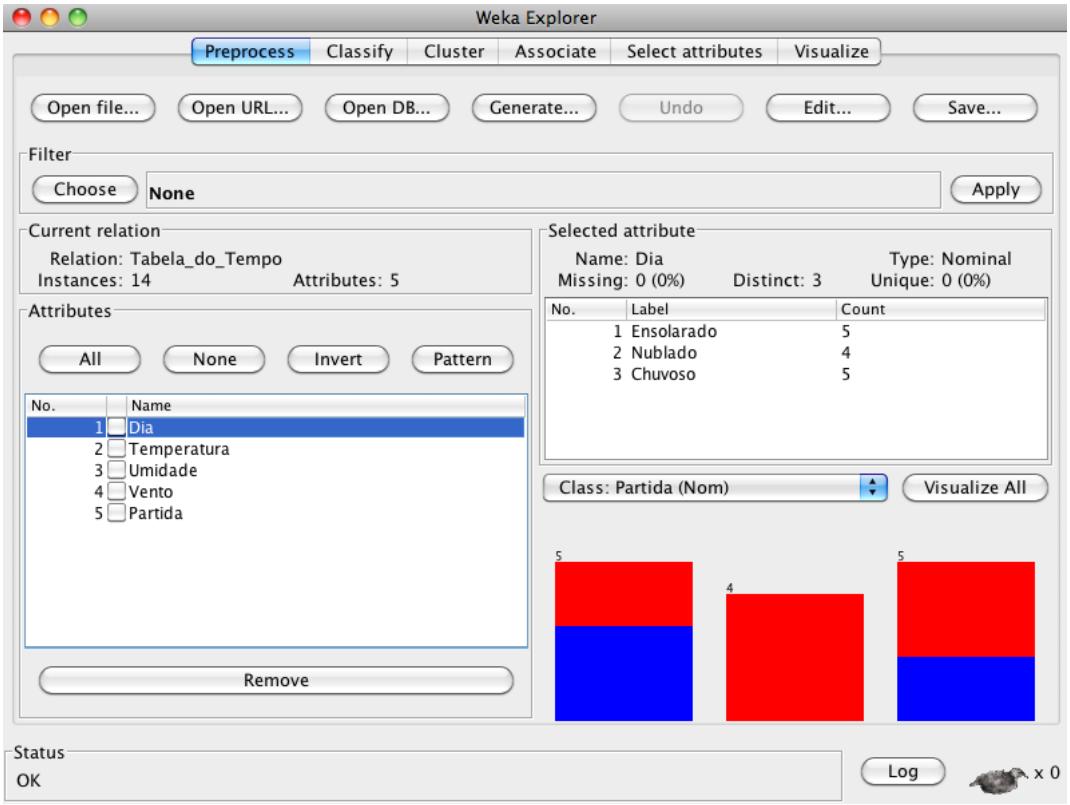


Figura 3.15: Arquivo "Tabela_do_Tempo.arff" aberto no Weka.

Note que na seção "Attributes" aparecem os cinco atributos da Tabela do Tempo na ordem em que foram declarados. Na Figura 3.15 aparecem ainda os detalhes da composição de um dos atributos selecionados, neste caso "Dia", mas qualquer um dos quatro atributos restantes pode ser selecionado.

Passo 3 – Com o arquivo "Tabela_do_Tempo.arff" aberto, clique na aba "Classify", localizada da parte superior esquerda da janela do Weka Explorer. A seguir, clique em "Choose" para escolher o algoritmo de classificação. Primeiro clique em "trees", depois em "J48" (Figura 3.16a e Figura 3.16b). O algoritmo "J48" é uma implementação mais recente do "ID3" e, além disso, tem também a vantagem de permitir, dentro do Weka, visualizar a Árvore de Decisão construída.

Uma vez escolhido o algoritmo "J48" é possível conferir e alterar alguns parâmetros. Clicando com o botão esquerdo do mouse sobre "J48" abre-se um menu com todos os valores *default*. Inicialmente vamos trabalhar com estes valores.

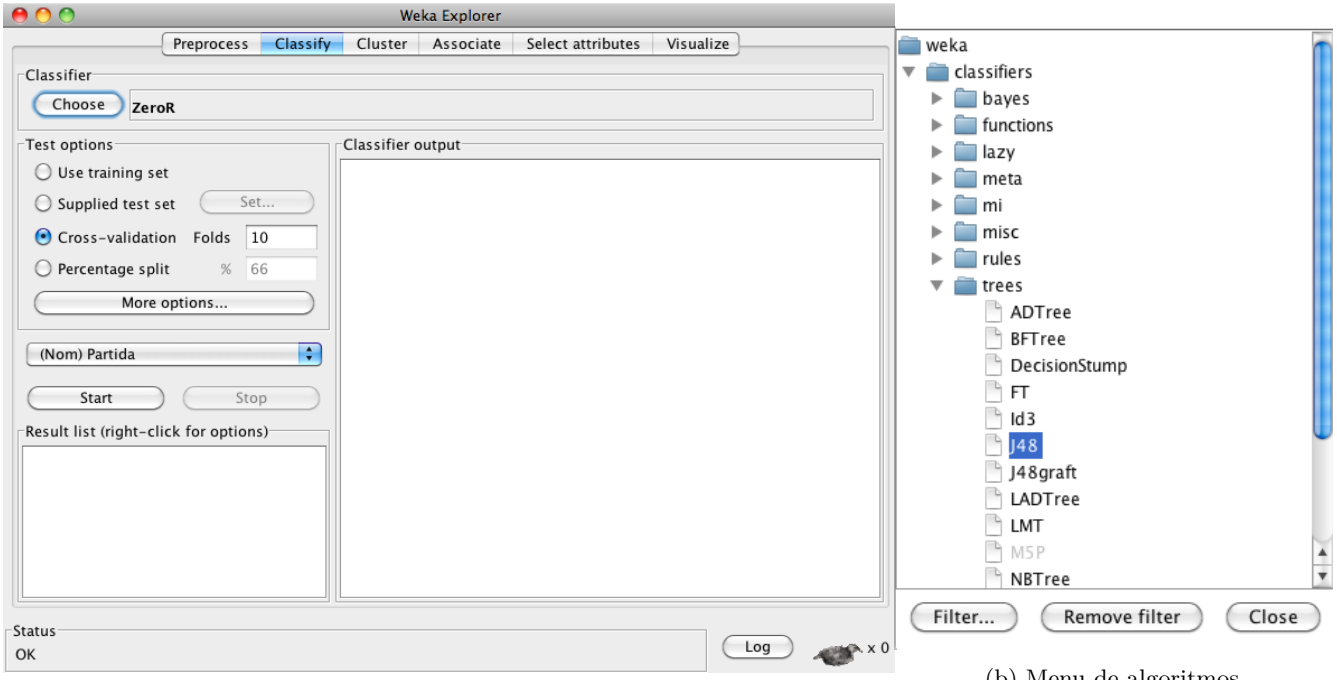
Passo 4 – Ainda dentro da aba "Classify", em "Test options" escolha "Use training set" e clique no botão "Start", um pouco abaixo. A opção "Use training set" significa que o mesmo **Conjunto de Treinamento** usado para gerar a Árvore de Decisão será usado para testar os resultados (veja Figura 3.17). Se o Conjunto de Exemplos da Base de Dados não for inconsistente, geralmente a taxa de acerto com esta opção deve ser de 100%. Mais adiante, na próxima unidade de estudo, veremos que há outras formas mais interessantes de testar a robustez e a qualidade do Modelo gerado.

Passo 5 – Ao disparar o processo de treinamento com o algoritmo "J48" aparecem na região direita da tela ("Classifier output") os resultados desejados (Figura 3.18). A Árvore de Decisão aparece na forma textual, mas pode ser vista na forma gráfica. Na parte inferior da tela, aparecem o número e a porcentagem de exemplos classificados corretamente, a Acurácia (ou Precisão) por Classe e a Matriz de Confusão.

Se você tiver interesse em saber como cada uma dos exemplos foi classificado, clique na aba "More options..." e depois habilite "Output predictions". Clique em "Start" novamente.

Passo 6 – Clicando com o botão direito do mouse na região inferior esquerda, onde se lê "Result list (right-click for options)", mais precisamente sobre a faixa azul "trees.J48", é possível visualizar graficamente o resultado, escolhendo "Visualize tree" (Figura 3.19).

Os números entre parênteses em cada nó folha da Figura 3.19 indicam quantos exemplos chegaram até esta folha.



(a) Opção "Choose" para escolher

(b) Menu de algoritmos

Figura 3.16: Aba "Classify" com a opção (a) "Choose" para escolher e (b) o menu de algoritmos.

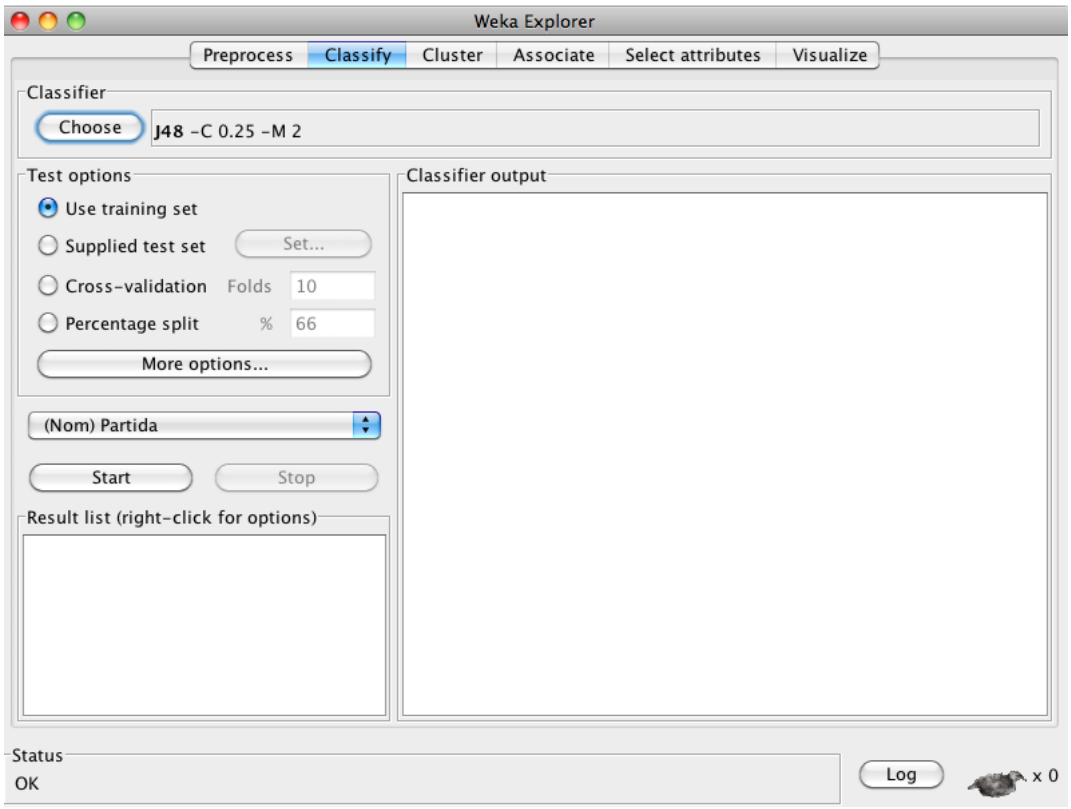


Figura 3.17: A Escolha da Opção de Teste "Use training set".

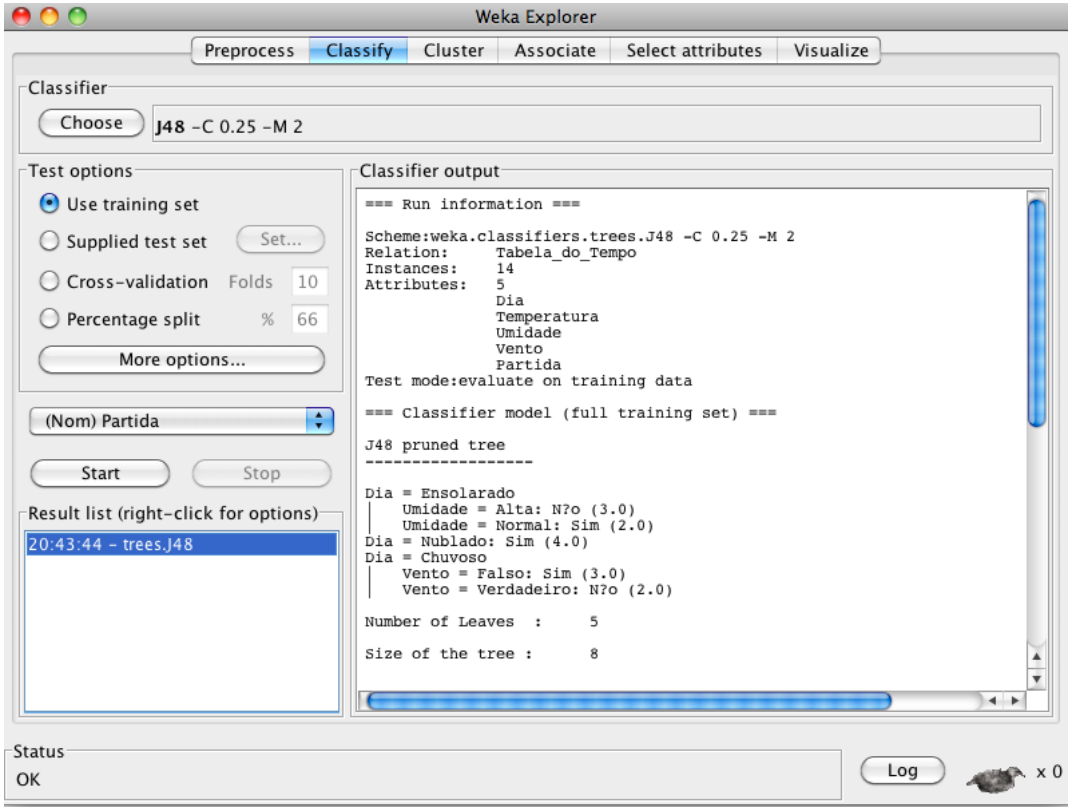


Figura 3.18: Resultado do Processo de Treinamento e Indução da Árvore de Decisão.

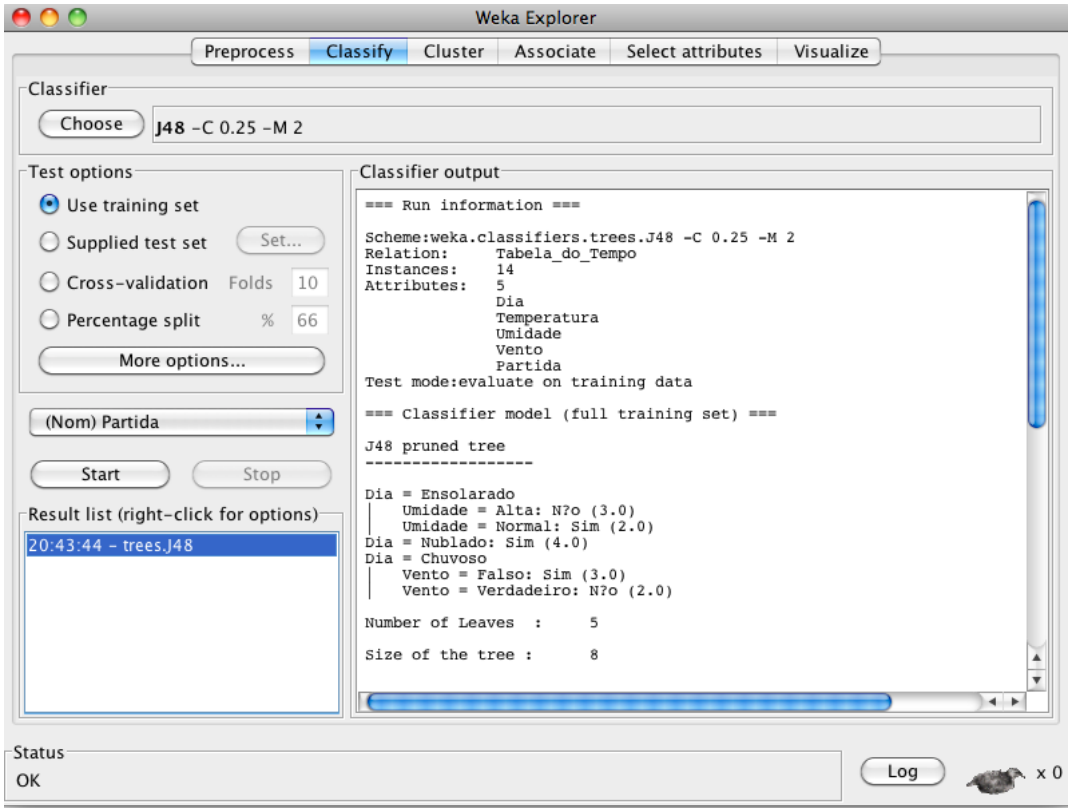


Figura 3.19: Representação Gráfica da Árvore de Decisão.

Somando-se estes números, verifica-se que 14 exemplos foram testados nesta simulação.

Passo 7 – Se você quiser saber exatamente quais Exemplos foram classificados em quais classes, primeiramente abra o arquivo "Tabela_do_Tempo.arff" no editor do Weka, que pode ser acessado pelo seguinte caminho: interface "Weka GUI Chooser", depois "Tools", e depois "ArffViewer" (Figura 3.20).

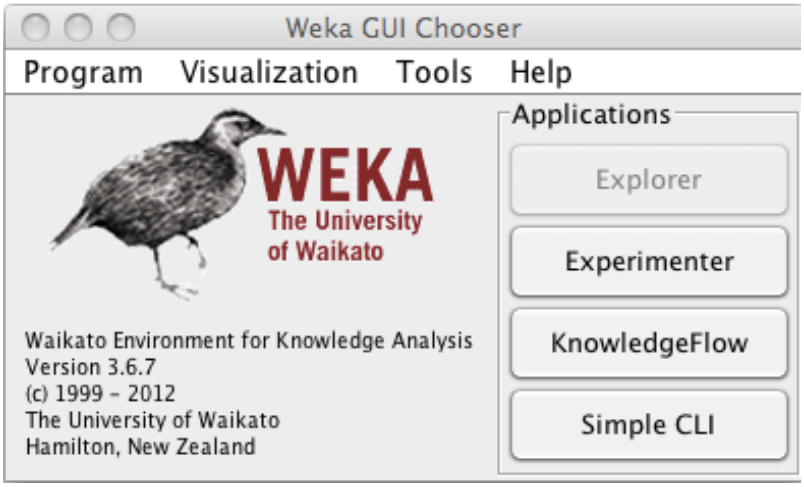


Figura 3.20: Interface "Weka GUI Chooser".

Quando a janela "ARFF-Viewer" se abrir, localize o arquivo "Tabela_do_Tempo.arff" clicando primeiramente em "File" e depois em "Open...".

Com o editor do Weka aberto, clique no atributo "Umidade", ou qualquer outro atributo, para ordenar os Exemplos de acordo com os valores que este atributo pode assumir.

Note que com este editor, é possível alterar os valores dos atributos, clicando sobre o campo a ser alterado.

Obs.: O editor do Weka também pode ser acessado no "Weka Explorer", escolhendo a aba "Preprocess" e depois "Edit...".

Passo 8 – Para testar novos Exemplos no Classificador sem sair da interface "Weka Explorer", há a opção "Supplied test set". Vamos supor que os quatro novos Exemplos da Tabela 3.16 de nosso material de Teoria devam ser testados no "Weka Explorer".

Primeiramente crie um arquivo ".arff" com os quatro Exemplos extras da Tabela 3.16 (basta abrir o arquivo "Tabela_do_Tempo.arff" no editor do Weka, fazer as modificações e salvar com um novo nome, digamos "Teste.arff").

Carregue no "Weka Explorer" o **Conjunto de Treinamento** "Tabela_do_Tempo.arff" da forma usual, i.e., clicando no "Preprocess" da barra superior e depois em "Open file...". A seguir clique em "Classify", escolha em "Choose" o algoritmo desejado, digamos "J48", e em "Test options" escolha "Supplied test set". Pressionando a tecla "Set", aparece a opção "Open file..." com a qual é possível carregar o **Conjunto de Teste** "Teste.arff".

Em "More options" habilite a opção "Output predictions" e dispare o programa com a opção "Start". Na seção "Classifier output", devem aparecer as quatro predições buscadas.

3.12 Considerações Finais

A geração de Árvores de Decisão normalmente é comparativamente mais rápida que outros métodos de classificação. Árvores de Decisão pequenas são fáceis de entender e Árvores grandes podem ser convertidas em Regras de Classificação. Geralmente a taxa de acerto de classificação de Exemplos de Teste, ou seja, a Acurácia das Árvores de Decisão, é compatível com outros métodos equivalentes, ou um pouco abaixo de métodos mais complexos. Porém, em Aprendizado de Máquina raramente se encontra um método com desempenho superior que seus pares para qualquer conjunto de dados.

Não foram estudados aqui casos reais de inconsistências, ausência de dados ou exceções na Base de Dados, para citar apenas alguns dos possíveis problemas. Suponha que durante a indução da Árvore de Decisão dois Exemplos ligeiramente distintos, mas que percorrem o mesmo caminho, pertençam a classes distintas! Neste caso, verifica-se

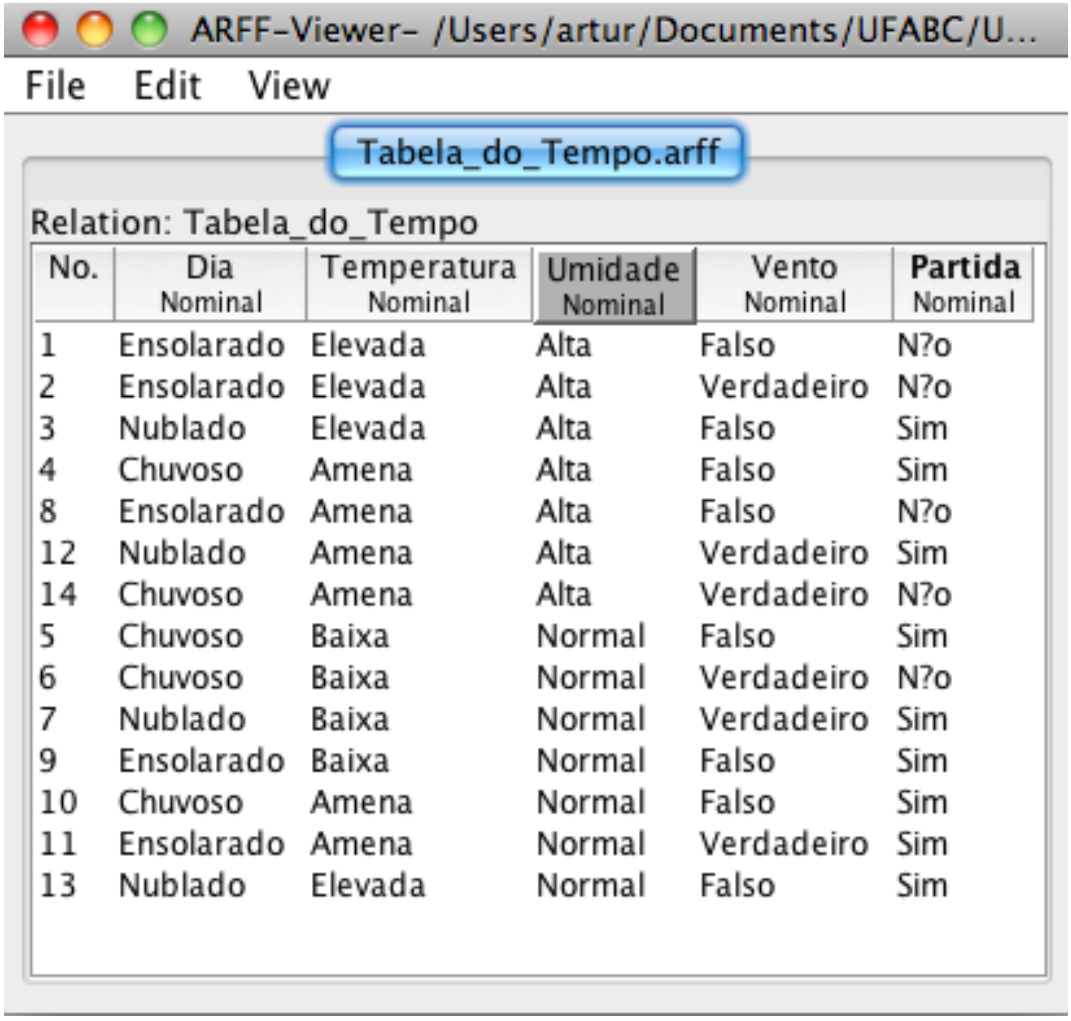


Figura 3.21: Tabela do Tempo Ordenada pelos Valores de “Umidade”.

que há inconsistência nos dados e uma análise pontual deverá determinar e eliminar o problema. Considere casos reais de Exemplos ausentes representados por caminhos logicamente possíveis, como um dos valores possíveis que determinado atributo pode assumir, mas que não estão presentes na Base de Dados. Que classe atribuir a estes casos? Algum critério deverá ser adotado para estes casos, como o de atribuir o valor da classe estatisticamente predominante no conjunto de Exemplos.

Além dessas situações, há os casos de dados espúrios causados por coleta equivocada, mas com valores lógicos perfeitamente dentro das possibilidades aceitas para cada atributo, e que não terão sido detectados na fase inicial de limpeza de dados porque a natureza desses problemas é de incompatibilidade com o modelo gerado ou o conceito aprendido. O tratamento de problemas desta natureza foge ao escopo desta primeira abordagem à geração ou indução de Árvores de Decisão, mas tais problemas podem ser estudados na referência bibliográfica fornecida na parte final deste material.

3.13 Lista de Exercícios

1. (20%) Explique **com suas próprias palavras** a seguinte afirmação: numa **Árvore de Decisão**, os nós testam **Atributos**.

2. (30%) Explique **com suas próprias palavras**, o que é "superajuste" ou "*overfitting*", seus efeitos e dê um exemplo.

3. Carregue o arquivo "iris.arff" (anexo) no Weka e elimine os atributos "sepalwidth" (largura da sépala) e "sepallength" (comprimento da sépala). Para fazer esta operação, basta selecionar estes dois atributos no "Weka Explorer" e depois clicar em "Remove". Devem sobrar apenas três atributos: "petallength" (comprimento da pétala), "petalwidth" (largura da pétala) e "class" (classe), com 150 Exemplos, divididos entre "Iris-setosa", "Iris-versicolor" e "Iris-virginica".

(a) (30%) – Gere a Árvore de Decisão com o algoritmo "J4.8", analise a representação gráfica da árvore e explique por que esta Árvore de Decisão deve cometer menos erros de classificação que a Árvore de Decisão da Figura 3.2, de nosso material didático.

(b) (20%) – No "Weka Explorer", vá em "Visualize", ajuste "PlotSize" e "PointSize", clique em "Update" e escolha a representação com os atributos "petalwidth" e "petallength". Analise esta figura (que deve se parecer à Figura 3.1 de nosso material didático).

3.14 Referências do Capítulo

Este capítulo baseou-se em obras clássicas e manuais técnicos que definem o estado da arte em ciência de dados. A fundamentação teórica sobre a indução de árvores de decisão seguiu o trabalho seminal de **quinlan1986**, complementado pelas metodologias de mineração e descoberta de conhecimento exploradas em **han2008** e **tan2009**. As implementações práticas e o uso da ferramenta Weka fundamentam-se em **witten2005** e **frank2016weka**. Para a contextualização no cenário de sistemas inteligentes e análise de dados, foram consultadas as obras de **rezende2005**, **pinheiro2008** e **rocha2008analise**. Por fim, o conjunto de dados histórico utilizado em diversos exemplos práticos remete ao estudo de **fisher1936**.

Classificação e Regras de Classificação

Executar Colab Abrir si-md2 GitHub

4.1 Introdução

No contexto de Aprendizado de Máquina, a Classificação pressupõe que um Modelo tenha sido gerado ou induzido a partir de Exemplos de Treinamento. Com este Modelo, novos Exemplos de Teste podem ser classificados. A Figura 4.1 ilustra as três fases desse processo: **Treinamento**, **Aprendizado** e **Classificação**.

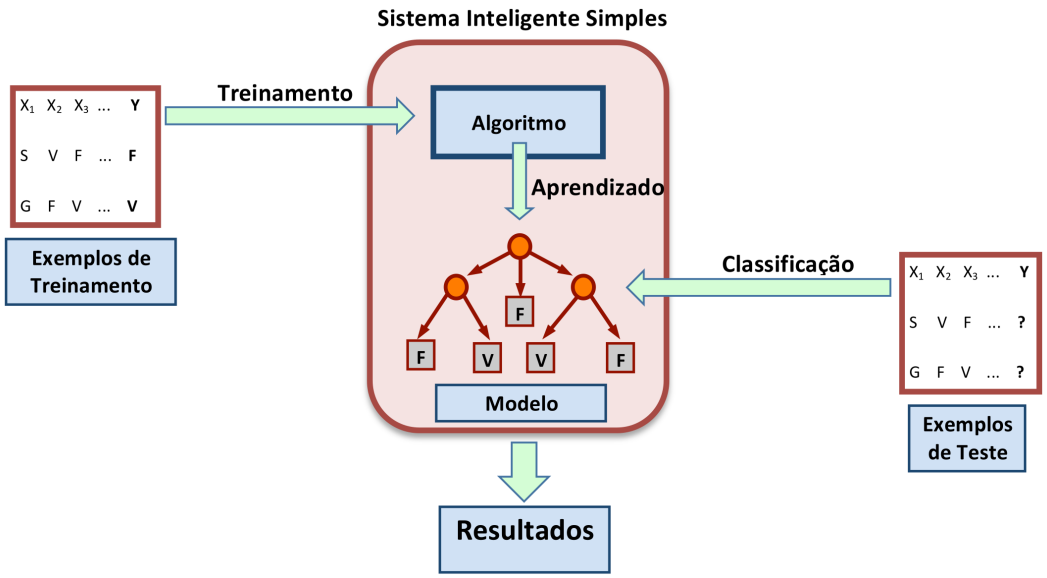


Figura 4.1: Treinamento, Aprendizado e Classificação em um Sistema Inteligente Simples.

Há várias maneiras de representar o conhecimento embutido num Modelo, sendo as mais comuns Árvores de Decisão e **Regras de Classificação**. As Regras de Classificação assumem a forma genérica:

IF Condição **THEN** Valor

Sendo "Valor" um resultado discreto, como sim/não, baixo/médio/alto verdadeiro/falso etc. Quando os resultados esperados não pertencem a classes discretas, ou seja, quando "Valor" for uma variável real, a classificação recebe o nome de **Regressão**.

A Classificação através de Regras de Classificação e o processo de geração automática de Regras de Classificação serão os temas desta unidade.

4.2 Classificação

O **Aprendizado Supervisionado** se dá através de um **Algoritmo de Aprendizado**, cuja função é criar uma representação do conhecimento extraído de um conjunto de **Exemplos de Treinamento**. Há várias formas possíveis de representação, mas a que nos interessa aqui são as **Regras de Classificação**.

Os Exemplos de Treinamento, por sua vez, são uma forma conveniente de estruturar os dados de uma empresa ou de um certo domínio do saber. Todos os Exemplos de Treinamento têm o mesmo número de atributos, sendo a diferença entre eles representada pelos valores que cada atributo assume.

O algoritmo de aprendizado recebe este nome porque ele cria um modelo cuja aplicação específica não foi pensada pelo seu projetista. Ou seja, ele demonstra certa capacidade de adaptação que melhora seu desempenho depois de uma fase de treinamento. Alguns algoritmos de aprendizado demonstram a capacidade de aprendizado incremental e podem ir melhorando seu desempenho com o acúmulo de experiência. Esta capacidade é muito apreciada porque concede certa **autonomia** aos **Sistemas Inteligentes**.

Uma equipe de físicos, engenheiros e cientistas da computação, responsáveis pelo lançamento de sondas espaciais, conseguiu recentemente desenvolver um algoritmo de aprendizado que permite a uma sonda espacial sair de sua trajetória para desviar-se de uma possível colisão com um meteorito ou cometa inesperado e retornar à rota original. Alguns robôs atualmente conseguem, sem ajuda externa, voltar à posição vertical depois de uma queda acidental causada por irregularidade no terreno e continuar normalmente sua tarefa de rotina.

4.3 Algoritmos de Aprendizado

Um algoritmo de aprendizado pode variar desde aqueles que simplesmente escolhem um dos atributos do Conjunto de Treinamento como a resposta possível a um teste, caso do algoritmo ***oneR*** (uma Regra), passando por aqueles cuja resposta a um novo teste é uma combinação linear dos valores dos atributos, até a utilização de complicados modelos não-lineares, como as ***Redes Neurais***, ou o aprendizado estatístico das ***Máquinas de Vetor de Suporte***, ou ***Support Vector Machines***, ***SVM***.

A finalidade desta unidade não é estudar detalhadamente algoritmos de aprendizado. Como há um grande número de ferramentas de acesso livre que implementam estes algoritmos, nosso objetivo é passar uma ideia geral do funcionamento desses algoritmos, cujo entendimento é indispensável para o ajuste adequado de seus parâmetros e para a correta interpretação de seus resultados.

4.4 Algoritmo *oneR* ou 1R (uma Regra)

É possivelmente o Algoritmo de Aprendizado para classificação mais simples e, no entanto, seu desempenho pode ser surpreendentemente bom, dependendo da Base de Dados (**witten2005**). Esse algoritmo aposta na hipótese de que basta consultar apenas um dos atributos para classificar corretamente os Exemplos de Teste. A tarefa então do algoritmo é encontrar durante o treinamento o atributo que apresenta a menor taxa de erros de classificação.

O algoritmo *oneR* tem bom desempenho para Bases de Dados em que um dos atributos é claramente mais importante que o restante, porque seus valores quase sempre darão uma pista de qual deve ser a classificação correta. Para Bases de Dados em que todos os atributos contribuem com igual peso na resposta, a taxa de acerto do algoritmo tende a ser inversamente proporcional ao número de atributos.

Para compreendermos como o *oneR* calcula a taxa de erros de cada atributo, vamos utilizar a clássica Tabela do Tempo, criada por (**quinlan1986**), e reproduzida na Tabela 4.1.

Tabela 4.1: Tabela do Tempo.

Dia	Temperatura	Umidade	Vento	Partida
Ensolarado	Elevada	Alta	Falso	**Não**
Ensolarado	Elevada	Alta	Verdadeiro	**Não**
Nublado	Elevada	Alta	Falso	**Sim**
Chuvoso	Amena	Alta	Falso	**Sim**

Dia	Temperatura	Umidade	Vento	Partida
Chuvoso	Baixa	Normal	Falso	**Sim**
Chuvoso	Baixa	Normal	Verdadeiro	**Não**
Nublado	Baixa	Normal	Verdadeiro	**Sim**
Ensolarado	Amena	Alta	Falso	**Não**
Ensolarado	Baixa	Normal	Falso	**Sim**
Chuvoso	Amena	Normal	Falso	**Sim**
Ensolarado	Amena	Normal	Verdadeiro	**Sim**
Nublado	Amena	Alta	Verdadeiro	**Sim**
Nublado	Elevada	Normal	Falso	**Sim**
Chuvoso	Amena	Alta	Verdadeiro	**Não**

Primeiramente isolamos um dos atributos, digamos “Dia” e verificamos qual a distribuição das classes “Sim” e “Não” no atributo de saída “Partida”.

Tabela 4.2: Relação entre “Dia” e “Partida”.

Dia	Partida
Ensolarado	**Sim**
Ensolarado	**Sim**
Ensolarado	**Não**
Ensolarado	**Não**
Ensolarado	**Não**
Nublado	**Sim**
Nublado	**Sim**
Nublado	**Sim**
Nublado	**Sim**
Chuvoso	**Sim**
Chuvoso	**Sim**
Chuvoso	**Sim**
Chuvoso	**Não**
Chuvoso	**Não**

Vamos considerar como “sucesso” a classe (“Sim” ou “Não”) que aparecer com maior frequência, i.e., a maioria, para cada uma das opções possíveis (“Ensolarado”, “Nublado”, “Chuvoso”) do atributo “Dia” e como “erro” a menos frequente. Então uma inspeção na Tabela 4.2 mostra a seguinte distribuição:

Tabela 4.3: Taxa de Erros do Atributo “Dia”.

Valor do Atributo	Exemplos com Partida=Não	Exemplos com Partida=Sim	Maioria	Erros
Ensolarado	**3** $\longrightarrow$	2	**Não**	2/5
Nublado	0	**4** $\longrightarrow$	**Sim**	0/4
Chuvoso	2	**3** $\longrightarrow$	**Sim**	2/5
Total de Erros				**4/14**

Com base na Tabela 4.3 já podemos gerar algumas regras de classificação iniciais:

$$\text{Dia(Ensolarado)} \rightarrow \text{Partida=Não} \quad \text{ou} \quad \textbf{IF} \quad \text{Dia=Ensolarado} \quad \textbf{THEN} \quad \text{Partida=Não} \tag{4.1}$$

$$\text{Dia(Nublado)} \rightarrow \text{Partida=Sim} \quad \text{ou} \quad \textbf{IF} \quad \text{Dia=Nublado} \quad \textbf{THEN} \quad \text{Partida=Sim} \tag{4.2}$$

$$\text{Dia(Chuvoso)} \rightarrow \text{Partida=Sim} \quad \text{ou} \quad \textbf{IF} \quad \text{Dia=Chuvoso} \quad \textbf{THEN} \quad \text{Partida=Sim} \tag{4.3}$$

Os resultados mostram que em dia ensolarado não há partidas, possivelmente por se tratar de um esporte em quadra coberta e talvez porque os participantes prefiram neste tipo de dia outra atividade a céu aberto. Convém ressaltar que este conjunto de regras baseadas unicamente no atributo "Dia" apresenta uma taxa de erros de 4 em 14, ou 4/14. Vamos reproduzir a tabela de **witten2005** que aplica o mesmo procedimento para os outros atributos e ver se algum deles apresenta uma taxa de erros menor.

Tabela 4.4: Taxa de Erros dos Atributos.

Atributo	Regras	Erros	Total de Erros
Dia	Ensolarado → **Não**	2/5	**4/14**
	Nublado → **Sim**	0/4	
	Chuvoso → **Sim**	2/5	
Temperatura	Elevada → **Não**	2/4	**5/14**
	Amena → **Sim**	2/6	
	Baixa → **Sim**	1/4	
Umidade	Alta → **Não**	3/7	**4/14**
	Normal → **Sim**	1/7	
Vento	Falso → **Sim**	2/8	**5/14**
	Verdadeiro → **Não**	3/6	

A Tabela 4.4 revela que os atributos "Dia" e "Umidade" apresentam as menores taxas de erros. Adotando qualquer critério arbitrário de desempate, vamos ficar com o conjunto de erros gerados pelo atributo "Umidade" e gerar as seguintes Regras de Classificação:

$$\text{Umidade(Alta)} \rightarrow \text{Partida=Não} \quad \text{ou} \quad \textbf{IF} \quad \text{Umidade=Alta} \quad \textbf{THEN} \quad \text{Partida=Não} \tag{4.4}$$

$$\text{Umidade(Normal)} \rightarrow \text{Partida=Sim} \quad \text{ou} \quad \textbf{IF} \quad \text{Umidade=Normal} \quad \textbf{THEN} \quad \text{Partida=Sim} \tag{4.5}$$

Portanto, quando o algoritmo **oneR** tiver que classificar um novo exemplo, somente o atributo "Umidade" será considerado e o resultado será baseado nas Regras de Classificação 4.4 e 4.5. Isso significa que se os Exemplos de Treinamento forem usados como Exemplos de Teste e supondo que entre os 14 Exemplos de Treinamento não haja contradição entre si, o algoritmo **oneR** deve acertar 10 vezes e errar 4.

Mas, e se o Conjunto de Teste for diferente do Conjunto de Treinamento? Não será a estimativa de 10 acertos e 4 erros demasiadamente otimista ou ela se confirmará com os novos dados? E se o número de Exemplos de Treinamento tivesse sido 140, em vez de 14, que implicações isso teria nas estimativas de acertos?

Há outros algoritmos de classificação bem mais refinados que o **oneR** e que, na maioria dos casos, produzem resultados com taxa de sucesso mais elevada. Um dos algoritmos de classificação mais famosos é o ***PRISM*** (**cendrowska1987**), que utiliza o princípio de "cobertura", i.e., ele vai criando regras que se aplicam ao maior número possível de exemplos do Conjunto de Treinamento, até que toda a tabela esteja "coberta" pelas regras produzidas. Seu desenvolvimento foi inspirado nos "pontos fracos" do algoritmo de indução de Árvores de Decisão ID3 (**quinlan1986**), como a dificuldade de entender as árvores muito grandes e complexas geradas pelo algoritmo ID3.

Não vamos aqui nos deter em particularidades do **PRISM** porque os resultados gerados pelo oneR são suficientemente representativos para os nossos propósitos, de abordar os métodos de avaliação dos resultados produzidos pelo modelo induzido. E ao avaliarmos os resultados, estamos de certa forma avaliando a capacidade de predição de um modelo para determinada Base de Dados.

Abordar um modelo pelo seu desempenho é interessante porque há evidências empíricas de que nenhum algoritmo tem desempenho superior aos demais para qualquer Base de Dados. A estrutura interna do conjunto de dados desempenha um papel decisivo no desempenho do algoritmo e na qualidade dos resultados.

O algoritmo oneR, com toda sua simplicidade, pode ser imbatível para uma Base de Dados que tenha um atributo que se destaque sobre os demais, cujos valores são redundantes ou irrelevantes. E pode ser uma catástrofe para uma Base de Dados constituída por centenas ou milhares de atributos, todos igualmente importantes. Da mesma forma, qualquer algoritmo pode ter uma taxa de sucesso baixa se o Conjunto de Treinamento não constituir uma amostra representativa do universo de teste.

4.5 Avaliação dos Resultados

Após o treinamento para gerar um Modelo através do Algoritmo de Aprendizado, é de grande importância fazer uma avaliação do desempenho do modelo. Em outras palavras, interessa saber quão preditivo é o Modelo Aprendido. Há várias metodologia consagradas para este fim. Vamos iniciar nossa abordagem ao tema relembrando a **Acurácia** de um Classificador Binário.

Considere que nosso Classificador Binário esteja sendo usado para fazer um diagnóstico médico. Se as respostas possíveis para este diagnóstico forem "Positivo" e "Negativo", quatro possibilidades de predição podem ocorrer:

1. Se o paciente for portador de uma doença e o Classificador acertar > no diagnóstico, dizemos que este caso é um **Verdadeiro Positivo** > ou **VP**;
2. Se o paciente não for portador da doença e o Classificador acertar > no diagnóstico, dizemos que este caso é um **Verdadeiro Negativo** > ou **VN**;
3. Se o paciente for portador da doença, mas o Classificador errar no > diagnóstico indicando que ele está são, dizemos que este caso é um > **Falso Negativo** ou **FN**;
4. Se o paciente não for portador da doença, mas o Classificador errar > no diagnóstico indicando que ele está doente, dizemos que este > caso é um **Falso Positivo** ou **FP**.

Essas quatro combinações de resultados estão representadas na **Matriz de Confusão** mostrada na Tabela 4.5.

Tabela 4.5: Matriz de Confusão.

	Positivo Previsto	Negativo Previsto
Positivo Real	**Verdadeiro Positivo (VP)**	**Falso Negativo (FN)**
Negativo Real	**Falso Positivo (FP)**	**Verdadeiro Negativo (VN)**

A Precisão ou Acurácia do Classificador se expressa pelo número de classificações corretas (VP+VN) divididas pelo número total de classificações (VP+VN+FP+FN),

$$Acurcia = \frac{VP + VN}{VP + VN + FP + FN} \times 100\%$$

Ocorre que em situações reais o custo de um **Falso Positivo** pode não ser igual ao de um **Falso Negativo** e a Acurácia não consegue captar adequadamente essa situação de interesse.

Suponha que um classificador usado para Detecção de Anomalia tenha que atribuir a cada um dos 100 testes um rótulo de "Situação=Normal" ou "Situação=Anormal". Suponha ainda que a relação entre donos honestos de cartão de crédito e golpistas seja de 96 para 4 e o classificador tenha colocado 99 portadores de cartão na classe "Normal" e apenas um dos golpistas na classe "Anormal". Neste caso a Acurácia do Classificador será de,

$$Acurcia_{\text{Classif}} = \frac{96+1}{96+1+3+0} \times 100\% = 97\%$$

A interpretação baseada apenas na Acurácia indicaria um excelente desempenho, mas na realidade este é um péssimo classificador para uma operadora de cartões de crédito, porque seu interesse em detectar os golpistas é bem maior do que os donos honestos de cartão! Há um sem número de situações semelhantes a esta. Pense nos danos diferenciados, do ponto de vista de saúde pública, entre fornecer um falso diagnóstico positivo para um paciente são e um falso diagnóstico negativo para um paciente com uma doença contagiosa.

Para detectar estes casos de conjuntos não-balanceados de Falso Positivo e Falso Negativo, podemos definir a **Taxa de Verdadeiro Negativo**, também conhecida por **Especificidade**, como sendo o número de Verdadeiro Negativo (VN) dividido pelo número total de negativos, que é a soma de Verdadeiro Negativo (VN) mais Falso Positivo (FP), ou seja,

$$Taxa\_VN = \frac{\text{VN}}{VN+FP} \times 100\%$$

Se o indicador Taxa de Verdadeiro Negativo for utilizado para o caso citado dos cartões de crédito, teremos uma Taxa de Verdadeiro Negativo de,

$$Taxa\_VN = \frac{1}{1+3} \times 100\% = 25\%$$

Para outras situações, pode ser mais conveniente utilizar a **Taxa de Verdadeiro Positivo** , também conhecida por **Sensibilidade**, como sendo o número de Verdadeiro Positivo (VP) dividido pelo número total de positivos, que é a soma de Verdadeiro Positivo (VP) mais Falso Negativo (FN), ou seja,

$$Taxa\_VP = \frac{\text{VP}}{VP+FN} \times 100\%$$

Com estes indicadores em mente, suponha que se queira avaliar qual será o desempenho do Modelo gerado pelo Algoritmo de Aprendizado para determinada Base de Dados. Se utilizarmos o mesmo Conjunto de Treinamento como Conjunto de Teste, muito possivelmente a estimativa de desempenho resultará excessivamente otimista para testes reais, com novos Conjuntos de Teste.

Outra alternativa é reservar parte do Conjunto de Treinamento para ser usada como Conjunto de Teste. Mas qual o tamanho ideal da partição do conjunto de Exemplos de Treinamento? E como escolher os elementos deste subconjunto do Conjunto de Treinamento que serão usados para teste? E se o Conjunto de Teste for muito pequeno? Felizmente a Estatística tem estudos bastante robustos sobre questões dessa natureza que podem nos auxiliar.

4.6 Avaliação de Desempenho do Classificador

As duas técnicas citadas, conhecidas como **Técnica de Ressubstituição** e **Técnica de Reamostragem**, apresentam pontos fortes e fracos, e justificam a criação de vários **métodos de avaliação de desempenho** de classificadores. Vamos analisar apenas alguns dos principais métodos de avaliação de desempenho.

Método da Ressubstituição ou *"Use training set"*

Neste método o Conjunto de Treinamento é também utilizado como Conjunto de Teste, conforme mostra a Figura 4.2. Se o Conjunto de Treinamento for uma amostra representativa do universo do problema, suas estimativas de desempenho para um Conjunto de Teste composto por Exemplos não vistos anteriormente podem ser muito boas. Caso contrário, o modelo poderá apresentar muitos erros de generalização durante os testes, seja por problemas de excesso de complexidade do Modelo, que costuma causar *overfitting*, ou por Poda inadequada.

Por outro lado, o fato de o conjunto completo de treinamento ser usado para gerar o Modelo constitui uma vantagem sobre os métodos de reamostragem, principalmente se o número de Exemplos de Treinamento for pequeno.

De fato, na simulação Weka da Tabela 4.1 com o algoritmo oneR usando o método *"Use training set"*, o número de instâncias ou exemplos classificados corretamente foi 10 (71%) e 4 (29%) classificados incorretamente, conforme esperado. A Matriz de Confusão para os 14 Exemplos está mostrada na Tabela 4.6.

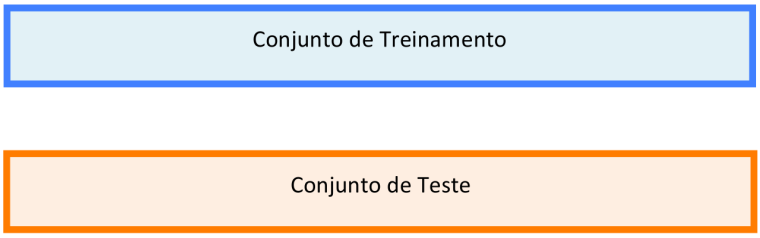


Figura 4.2: Na Ressubstituição, o Conjunto de Treinamento também é o Conjunto de Teste.

Tabela 4.6: Matriz de Confusão para a Tabela do Tempo com o oneR e o Método *"Use training set"*.

	Não Previsto	Sim Previsto
Não Real	**3**	**2**
Sim Real	**2**	**7**

Quando repetimos os mesmos procedimentos, porém usando o algoritmo *PRISM*, os resultados foram os seguintes: simulação Weka da Tabela 4.1 com o algoritmo *PRISM* usando o método *"Use training set"*, o número de instâncias ou exemplos classificados corretamente foi 14 (100%) e 0 (0%) classificados incorretamente. A Matriz de Confusão para os 14 Exemplos está mostrada na Tabela 4.7.

Tabela 4.7: Matriz de Confusão para a Tabela do Tempo com o *PRISM* e o Método *"Use training set"*.

	Não Previsto	Sim Previsto
Não Real	**5**	**0**
Sim Real	**0**	**9**

Método da Divisão da Amostra ou *Holdout* ou *Percentage Split*

Consiste na divisão dos Exemplos de Treinamento em dois conjuntos disjuntos, um para Treinamento, outro para Teste, conforme mostra a Figura 4.3. O valor das porcentagens de cada um dos conjuntos é geralmente expresso numa única porcentagem maior que 50%, estando subentendido que o conjunto menor é o complemento para 100%. O valor de divisão mais comum é 66% para Treinamento e 34% para Teste, embora não haja evidências empíricas que justifiquem essa escolha de 2/3 e 1/3.

Sua vantagem é a simplicidade, mas dependendo da composição obtida, as classes dos Exemplos podem não estar igualmente representadas nos dois conjuntos. Outra limitação desse método está no fato de que menos Exemplos são usados no Treinamento, podendo ter um impacto negativo no desempenho do Modelo induzido.

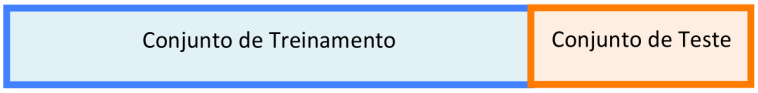


Figura 4.3: No *Holdout*, os Exemplos de Treinamento são Divididos em Dois Subconjuntos.

Para Conjuntos de Teste excessivamente pequenos, dividir o já escasso número de Exemplos de Teste pode ter um efeito desastroso ou na geração do Modelo ou na sua avaliação de desempenho. De fato, na simulação Weka da Tabela 4.1 com o algoritmo oneR usando o método da *"Percentage Split"*, com o Conjunto de Treinamento correspondendo a 66% dos Exemplos, o número de instâncias ou exemplos classificados corretamente foi 2 (40%) e 3 (60%) classificados incorretamente! A Matriz de Confusão para os 5 Exemplos considerados está mostrada na Tabela 4.8.

Tabela 4.8: Matriz de Confusão para a Tabela do Tempo com o *oneR* e o Método da *"Percentage Split"*.

	Não Previsto	Sim Previsto
Não Real	**0**	**2**
Sim Real	**1**	**2**

foi 4 (80%) e 1 (20%) classificado incorretamente. A Matriz de Confusão para os 5 Exemplos está mostrada na Tabela 4.9.

Tabela 4.9: Matriz de Confusão para a Tabela do Tempo com o *PRISM* e o Método *"Percentage Split"*.

	Não Previsto	Sim Previsto
Não Real	**1**	**1**
Sim Real	**0**	**3**

Método da Validação Cruzada ou *Cross-validation*

Neste método, os Exemplos de Treinamento são aleatoriamente divididos em k partições mutuamente exclusivas ou "*folds*", sendo k normalmente igual a 10. A Figura 4.4 mostra um exemplo para $k = 4$, ou seja, 3/4 dos Exemplos de Treinamento são usados para Treinamento e 1/4 dos Exemplos de Treinamento são reservados para a fase de Teste.

A cada iteração um desses *folds* será usado como Conjunto de Teste, enquanto que os outros serão usados para Treinamento. O nome de validação cruzada se justifica então pelo fato de que cada *fold* será usado *(k-1)* vezes para Treinamento e uma vez para Teste. O erro total será a soma dos erros de todas as k execuções, enquanto que o erro médio é o erro total dividido pelo número k de partições.

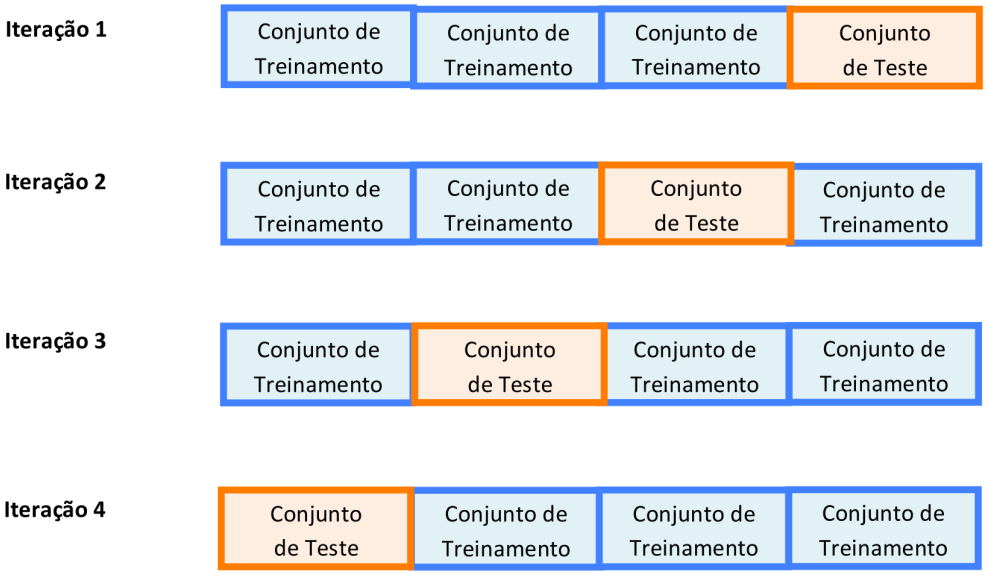


Figura 4.4: Na Validação Cruzada, o Conjunto de Treinamento é Dividido em k Partições.

De fato, na simulação Weka da Tabela 4.1 com o algoritmo oneR usando o método da *"Cross-validation"*, com $k = 10$ *folds*, o número de instâncias ou exemplos classificados corretamente foi 5 (36%), e 9 (64%) classificados incorretamente! A Matriz de Confusão para os 14 Exemplos considerados está mostrada na Tabela 4.10.

Tabela 4.10: Matriz de Confusão para a Tabela do Tempo com o *oneR* e o Método da *"Cross-validation"*.

	Não Previsto	Sim Previsto
Não Real	**3**	**2**
Sim Real	**7**	**2**

Na simulação Weka da Tabela 4.1 com o algoritmo *PRISM* usando o método da *"Cross-validation"*, com $k = 10$ *folds*, o número de instâncias ou Exemplos classificados corretamente foi 12 (86%), e 0 (0%) classificados incorretamente! (com 3 Exemplos não classificados). A Matriz de Confusão para os Exemplos considerados está mostrada na Tabela 4.11.

Tabela 4.11: Matriz de Confusão para a Tabela do Tempo com o *PRISM* e o Método da *"Cross-validation"*.

	Não Previsto	Sim Previsto
Não Real	**5**	**0**
Sim Real	**0**	**7**

Método Deixe-Um-De-Fora ou *Leave-One-Out*

É um caso especial do Método da Validação Cruzada em que o número de partições k é igual ao número de Exemplos N, isto é, $k = N$, e cada partição é composta por apenas um Exemplo, como mostra a Figura 4.5. A vantagem é que mais dados são usados para o Treinamento, mas a desvantagem é seu custo computacional para os casos em que N for muito grande.

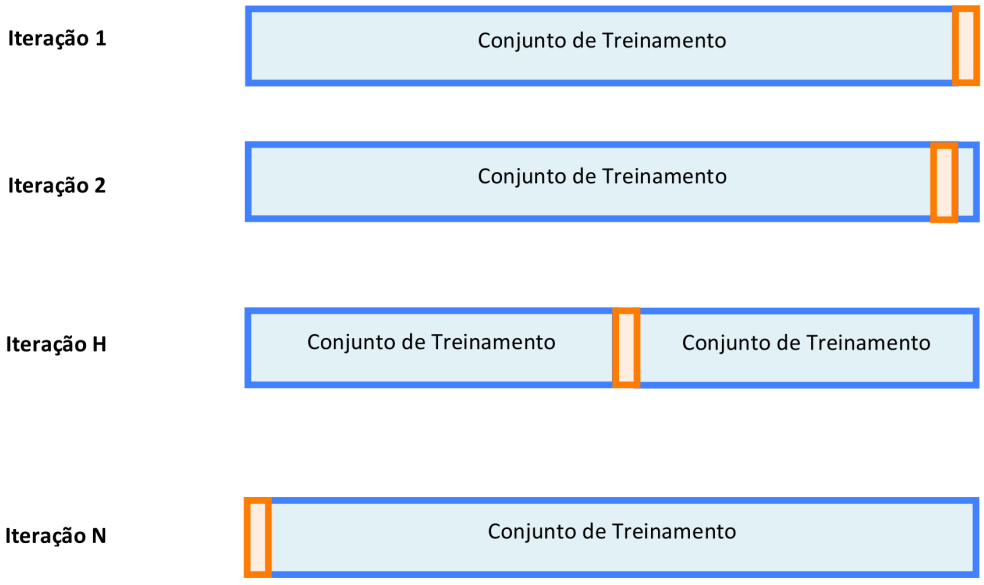


Figura 4.5: No *Leave-One-Out*, o Conjunto de Treinamento é Dividido em N Partições.

Na simulação Weka da Tabela 4.1 com o algoritmo oneR usando o método da *"Cross-validation"*, com $k = 14$ *folds*, número idêntico ao de Exemplos ou instâncias, portanto $k = N$, o resultado da classificação foi idêntico ao da *"Cross-validation"*, sendo o número de Exemplos classificados corretamente 5 (36%), e 9 (64%) classificados incorretamente! A Matriz de Confusão para os 14 Exemplos considerados está mostrada na Tabela 4.12.

Tabela 4.12: Matriz de Confusão para a Tabela do Tempo com o *oneR* e o Método da *"Leave-one-out"*.

	Não Previsto	Sim Previsto
Não Real	**3**	**2**
Sim Real	**7**	**2**

Repetindo o mesmo Conjunto de Teste, porém com o algoritmo *PRISM*, usando o método da *"Cross-validation"*, com $k = 14$ *folds*, o resultado da classificação foi idêntico ao da *"Cross-validation"*, sendo o número de Exemplos classificados corretamente 11 (79%), 0 (0%) classificados incorretamente e 3 Exemplos não classificados. A Matriz de Confusão para os 11 Exemplos considerados está mostrada na Tabela 4.13.

Tabela 4.13: Matriz de Confusão para a Tabela do Tempo com o *PRISM* e o Método da *"Leave-one-out"*.

	Não Previsto	Sim Previsto
Não Real	**5**	**0**
Sim Real	**0**	**6**

Nesta série de simulações, o algoritmo *PRISM* produziu um classificador com melhor desempenho que o classificador do algoritmo *oneR* para o mesmo Conjunto de Treinamento.

4.7 Considerações Finais

Nesta unidade, vimos um algoritmo simples, conhecido como oneR, usado para gerar Regras de Classificação. Outros algoritmos mais refinados usam princípios mais complexos, como o de cobertura, para produzir Regras de Classificação (caso do PRISM).

Foram apresentados alguns indicadores que auxiliam o usuário a decidir se os resultados obtidos são satisfatórios ou não. Também foram apresentados alguns métodos para estimar o desempenho futuro do classificador em situação de testes reais.

Duas simulações comparativas entre o desempenho do algoritmo oneR e PRISM para a Tabela do Tempo foram apresentadas para ajudar a fixar os conceitos aprendidos.

4.8 Lista Exercícios

1. (20%) Explique **com suas próprias palavras** as vantagens e desvantagens das **Árvores de Decisão** e **Regras de Classificação**.

2. (30%) Explique **com suas próprias palavras** os Métodos *"**Cross-validation**"* e *"**Holdout**"*.

3. Carregue o arquivo **"iris.arff"** (anexo) no Weka e faça a Classificação usando o algoritmo **"oneR"**, com o método ***"Cross-validation"*** (10 *Folds*).

(a) (30%) – Faça uma análise da **Matriz de Confusão** gerada, reproduzindo os valores obtidos na simulação.

(b) (20%) – Ainda com base nos resultados da simulação no Weka, explique **com suas próprias palavras** por que a classe **"Iris Setosa"** tem a mais alta taxa de **Verdadeiro Positivo** *("TP Rate")*, enquanto que a **"Iris Versicolor"**, a mais baixa (consulte a Figura 3.1 do Capítulo 3).

4.9 Referências do Capítulo

Este capítulo baseou-se nas metodologias de indução de regras de classificação exploradas em **cendrowska1987**, nas implementações práticas e no uso da ferramenta Weka fundamentadas em **witten2005**, e na contextualização no cenário de sistemas inteligentes e análise de dados consultadas em **rezende2005**, **rocha2008analise** e **tan2009**. A fundamentação teórica sobre a indução de árvores de decisão seguiu o trabalho seminal de **quinlan1986**.

Máquina de Vetores de Suporte ou *Support Vector Machine*

Executar Colab Abrir si-md2 GitHub

5.1 Introdução

Nas unidades anteriores, foi usado o recurso de representar um **Exemplo de Treinamento** por um ponto num plano cartesiano, como no caso da flor Íris (Figura 5.1). As Regras de Classificação, por sua vez, eram representadas por retas tracejadas e ilustravam como o **Modelo Aprendido** classificava os Exemplos.

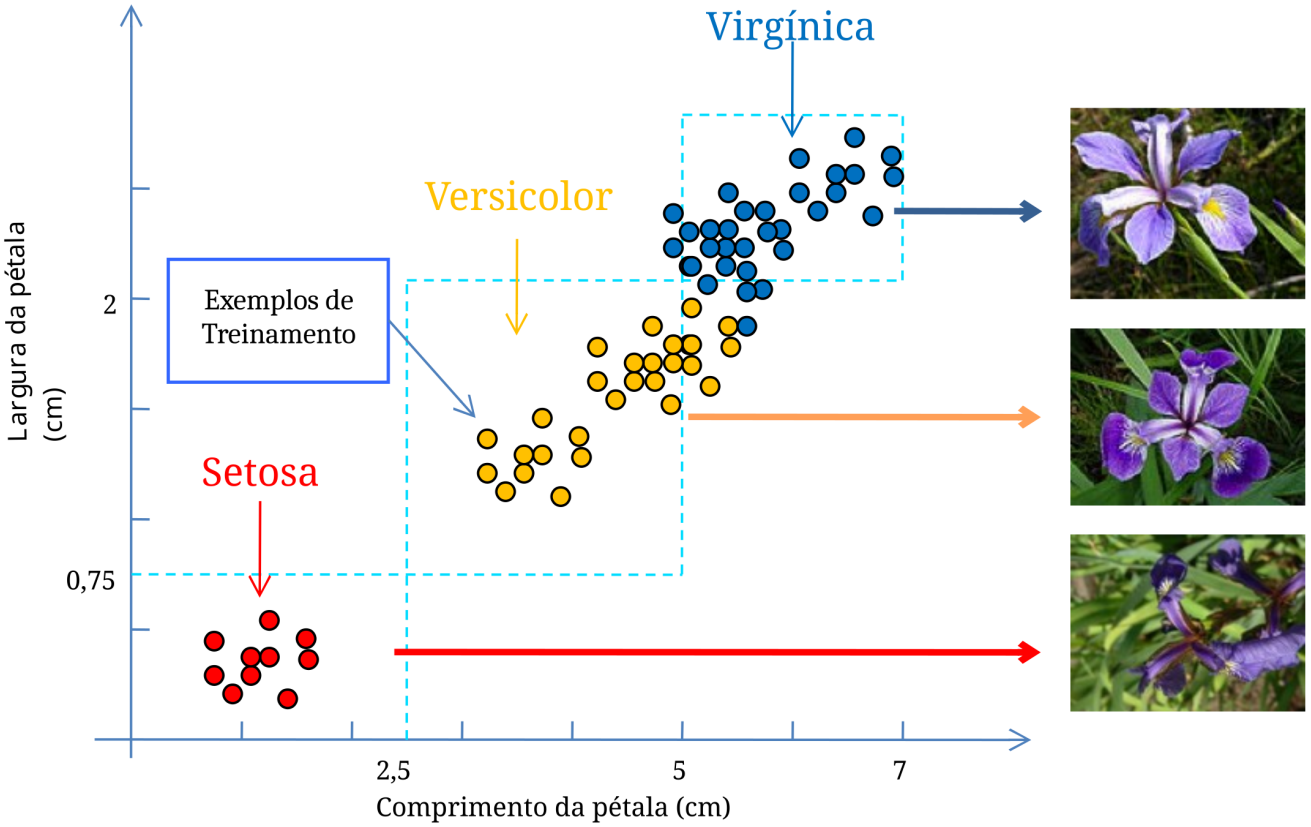


Figura 5.1: Representação de Exemplos de Treinamento no Plano Cartesiano [1] [2] [3].

[1] Fonte: Wikimedia Commons - Iris virginica (Acessado em 03.03.26).
[2] Fonte: Wikimedia Commons - Iris versicolor (Acessado em 03.03.26).
[3] Fonte: Wikimedia Commons - Iris setosa (Acessado em 03.03.26).

Tanto as **Regras de Associação** e **Classificação** quanto as **Árvores de Decisão** podem ser vistas como representantes de um paradigma de aprendizado supervisionado denominado **Aprendizado Orientado a Conhecimento**, porque são de fácil compreensão e utilização pelos seres humanos. Outro nome que se dá a este tipo de aprendizado é **Paradigma de Aprendizado Simbólico**, porque ele cria **Representações Simbólicas do Modelo** gerado, tais como Árvores de Decisão, Regras etc.

Vamos agora estudar um novo paradigma de aprendizado supervisionado, conhecido como **Paradigma de Modelos Funcionais**, que em geral apresenta resultados com melhor precisão que os representantes do paradigma Orientado a Conhecimento, mas que para entender como se chegou a determinado resultado há que ter um certo preparo em matemática. Este tipo de aprendizado também é conhecido como **Paradigma do Aprendizado Estatístico**, porque ele utiliza **Modelos Estatísticos** para guiar a geração do Modelo induzido durante o treinamento.

Como o escopo desta unidade não é a apresentação da teoria matemática dessa classe de algoritmos, vamos reduzir a um mínimo as deduções matemáticas e nos valer de ilustrações gráficas e interpretações geométricas das ideias fundamentais deste interessante tópico.

Dois dos algoritmos de aprendizado supervisionado mais populares dos Modelos Funcionais, são as **Redes Neurais**[4] ou *Neural Networks*, e as **Máquinas de Vetores de Suporte** ou *Support Vector Machine*. A teoria matemática sobre as Máquinas de Vetores de Suporte foi apresentada à comunidade científica por (**cortes1995**), em meados da década de 1990, e desde então suas implementações algorítmicas têm produzido resultados animadores para o problema do **reconhecimento de padrões** em Bases de Dados.

5.2 Representação dos Exemplos como Vetores

Nesta introdução a **Máquinas de Vetores de Suporte** (**MVS**), vamos passar a representar um **Exemplo de Treinamento** por um vetor, e o **Modelo Aprendido**, por outro vetor, chamado de **Vetor Peso w**. Qual a vantagem dessa forma de representação? A vantagem é que para classificar um Exemplo, bastará fazer o produto de dois vetores, algo computacionalmente simples. No contexto de classificação com MVS, mostraremos que o resultado dessa multiplicação será sempre ou "+1" ou "−1". Dessa forma, por meio da operação de **produto interno** entre dois vetores, os **Exemplos de Teste** serão classificados como pertencentes ou à classe "$y = +1$" ou à "$y = -1$".

Embora o algoritmo de aprendizado de um Modelo Funcional seja bem diferente daqueles algoritmos de aprendizado Orientado a Conhecimento, a formalização do problema da classificação continua a mesma. Em termos práticos, estamos simplesmente substituindo um módulo de classificação por outro, que desempenha a mesma função, possivelmente de forma mais complexa, porém com melhores resultados para muitos casos.

Um classificador binário é equivalente a uma função que faz o mapeamento de um conjunto de **atributos de entrada**, agora representados por um vetor $\mathbf{x}$, em uma das classes "$y = +1$" ou "$y = -1$". Em termos matemáticos,

$$y = f(\mathbf{x}), \qquad y \in \{-1,\ +1\} \qquad \text{e} \qquad \mathbf{x} \in \mathbb{R}^N \tag{5.1}$$

Por exemplo, num plano cartesiano, um ponto $\mathbf{x} \in \mathbb{R}^2$ pode ser descrito tanto por suas coordenadas (x_1, x_2), quanto por um vetor. Por esta razão, daqui para frente, em vez de utilizarmos o termo Exemplos de Treinamento, vamos falar em **Vetores de Treinamento** (cuja ilustração aparece na Figura 5.2). Da mesma forma, falaremos em **Vetores de Teste** em vez de Exemplos de Teste.

Para efeito didático, consideraremos um ponto no plano ou um vetor como representações equivalentes, e usaremos sempre a representação mais conveniente para o argumento em questão.

Por opção metodológica, vamos comparar as MVS com as Redes Neurais, com o propósito de mostrar como algumas das dificuldades mais sérias apresentadas pelas Redes Neurais têm sido superadas com as MVS. Mas, aproveitamos para reafirmar uma constatação empírica mencionada em outra unidade de que em Mineração de Dados não existe um algoritmo que sempre mostre desempenho superior aos demais para qualquer Base de Dados. Muitos autores consideram a Mineração de Dados mais uma arte que uma ciência, porque via de regra é preciso certo traquejo do operador e uma boa dose de experimentos empíricos para melhorar os resultados práticos.

[4] Em inglês, o termo "neural" é usado tanto para tópicos do sistema nervoso quanto para neurônios. Em português, no entanto, temos dois adjetivos distintos: neuronal e neural, sendo este último relacionado a nervos e ao sistema nervoso em geral. Como a inspiração inicial para as "*neural networks*" foram os neurônios, a tradução mais adequada para o português deveria ser "redes neuronais", mas como "redes neurais" acabou prevalecendo, vamos manter esta terminologia.

Aliás, um dos objetivos deste livro de **Sistemas Inteligentes**, e especialmente desta unidade, é propiciar experiências que favoreçam o desenvolvimento da sensibilidade indispensável a um usuário competente da **Mineração de Dados**.

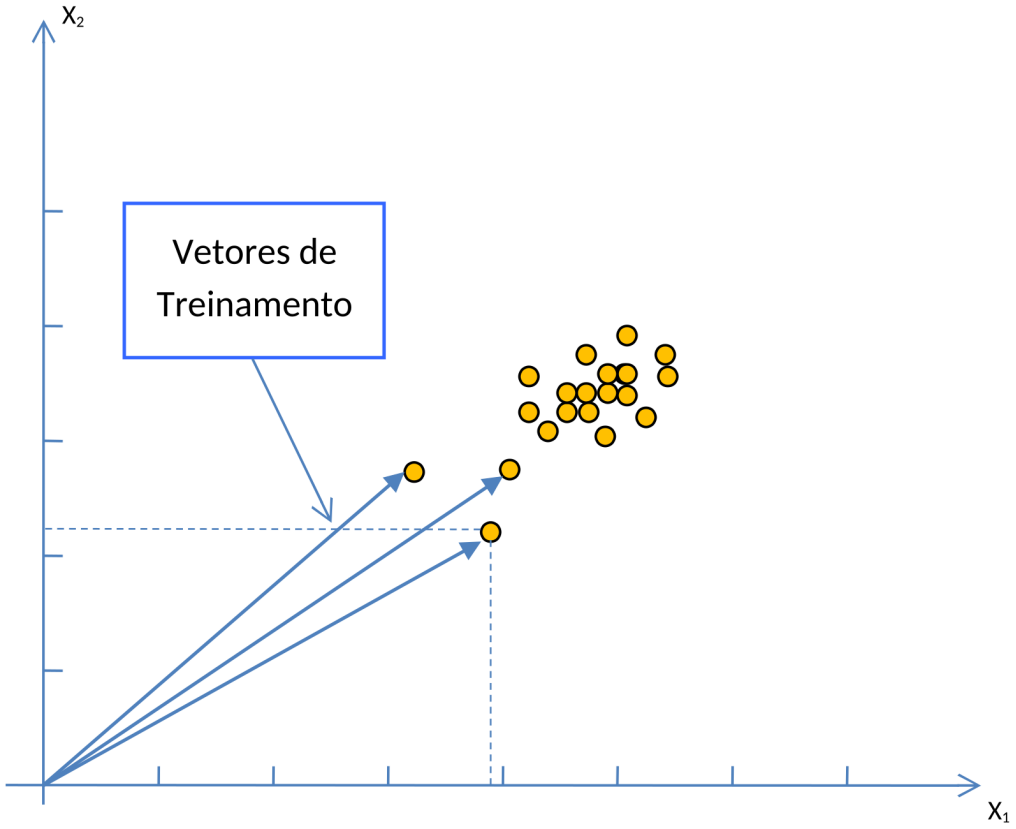


Figura 5.2: Representação de Vetores de Treinamento no Plano Cartesiano.

5.3 Classificadores Lineares

Vamos voltar ao conjunto de dados da flor Íris, porque este exemplo clássico apresenta situações típicas de um problema de classificação real. Olhando a Figura 5.1, nota-se que não há qualquer dificuldade para distinguir a Íris Setosa das Íris Versicolor e Virgínica. Mas a situação é bem diferente quando consideramos a Íris Versicolor com a Virgínica, porque há uma região comum a ambas. Encontrar um classificador que separe linearmente estes dois tipos de Íris constitui uma tarefa nada simples.

Vamos então dividir o problema em duas partes e considerar inicialmente apenas os tipos Setosa e Versicolor, já que a separação entre ambas parece ser bem mais simples. Com os resultados obtidos para os casos linearmente separáveis, vamos estendê-los aos casos não separáveis linearmente, com algumas adaptações.

A Figura 5.3 apresenta os dados apenas da Íris Setosa e Versicolor, com vários **classificadores** possíveis. Qual deles devemos escolher?

Talvez antes de dar uma resposta a esta pergunta, seria oportuno colocar outra questão ainda mais básica. Por que tentar fazer a separação entre as classes usando uma reta e não uma curva que se adapte aos dados? Porque usar uma reta, ou uma função linear para separar classes é matematicamente menos custoso do que outras formas de curvas, ou funções polinomiais, além de tornar o problema do *overfitting* menos provável. A equação de uma reta exige um tratamento matemático simples, resultando num custo computacional mais baixo do que o custo dos polinômios de grau maior ou igual a dois.

Dentre todas as possibilidades, o classificador azul parece ser o menos indicado porque ele comete vários erros de classificação durante o treinamento. Seu poder de generalização já se mostra antecipadamente comprometido. Mas dentre os restantes, qual deles vai apresentar o melhor desempenho na fase de teste? Ou seja, qual deles tem a melhor capacidade de generalização? Será este o melhor critério a utilizar para escolher um classificador?

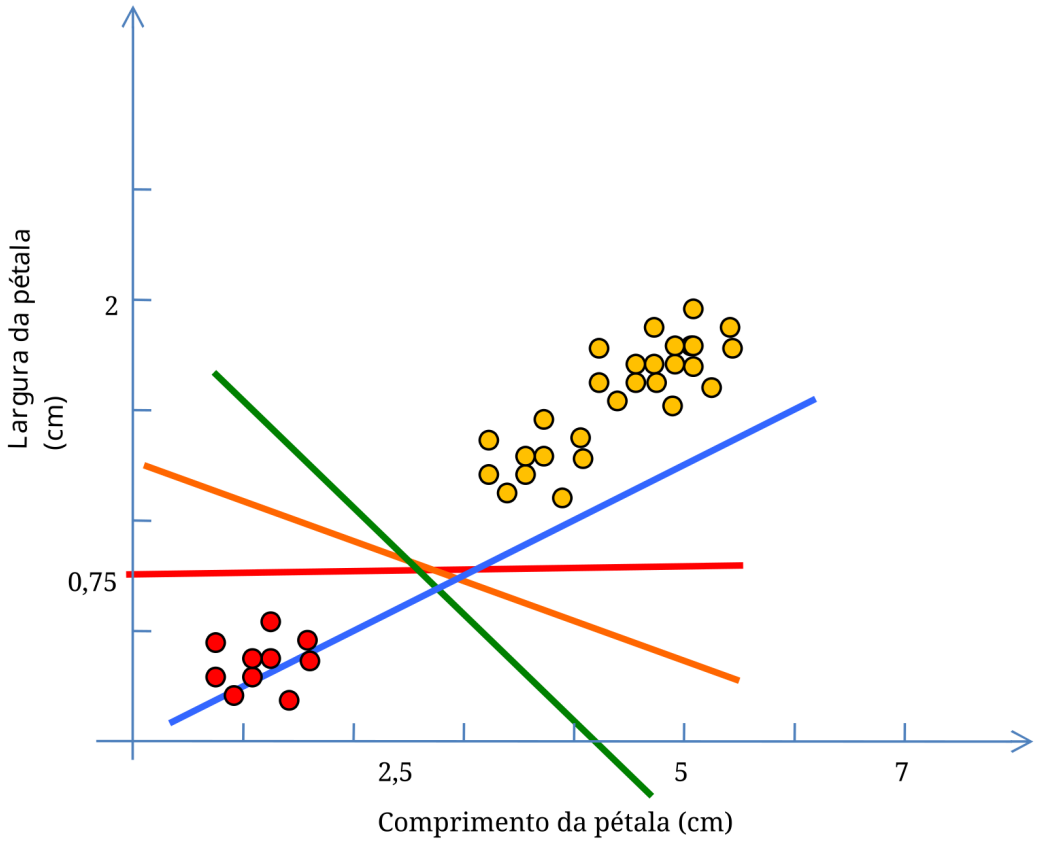


Figura 5.3: Qual Classificador Escolher?

Em outros algoritmos de Aprendizado Supervisionado de Classificadores Lineares, como o Perceptron[5] (**rosenblatt1957**), os três classificadores da Figura 5.3 que não cometem erros de classificação durante o treinamento seriam considerados igualmente bons. Aliás, como na resposta do Perceptron, e das **Redes Neurais** em geral, há um **componente probabilístico**, em cada simulação com o mesmo conjunto de Vetores de Treinamento, qualquer uma das três retas poderia ser escolhida alternadamente como o classificador linear. Para o Perceptron, desde que um classificador linear faça a separação das classes sem cometer erros durante o treinamento, ele é tão bom quanto seus pares que mostrarem o mesmo resultado.

Para as **MVS**, no entanto, que utilizam um **algoritmo determinístico**, existe um classificador considerado melhor que todos os demais, porque ele possivelmente vai minimizar os erros de classificação na fase de teste. E este classificador é o que maximiza suas margens, tornando-as as mais largas possíveis antes de tocar os primeiros pontos do plano, que representam os **Vetores de Suporte** (Figura 5.4). Note que o **vetor peso w**, também conhecido como **vetor normal w**, é perpendicular à reta que representa o classificador. Portanto, conhecendo-se o vetor **w**, a inclinação do classificador estará determinada.

Note que o classificador vermelho da Figura 5.3, por exemplo, também separaria sem erros as duas classes, mas suas margens não seriam tão largas quanto as mostradas na Figura 5.4. E como encontrar este classificador que maximiza as margens? Primeiramente considerando o conjunto de pontos como um **Conjunto Convexo**, e depois encontrando a menor distância entre eles. A reta de separação, i.e., o classificador, é perpendicular ao menor segmento que une os dois conjuntos.

5.4 Conjunto Convexo

Se conectarmos cada ponto de um conjunto aos demais pontos restantes, o polígono externo que resulta será um polígono convexo, e o conjunto de pontos delimitados por este polígono será chamado de **conjunto convexo**.

A Figura 5.5 mostra como as duas classes de Vetores de Treinamento podem ser representadas por dois conjuntos

[5] O *Perceptron* é um tipo de Rede Neural simples, com apenas uma camada de neurônios.

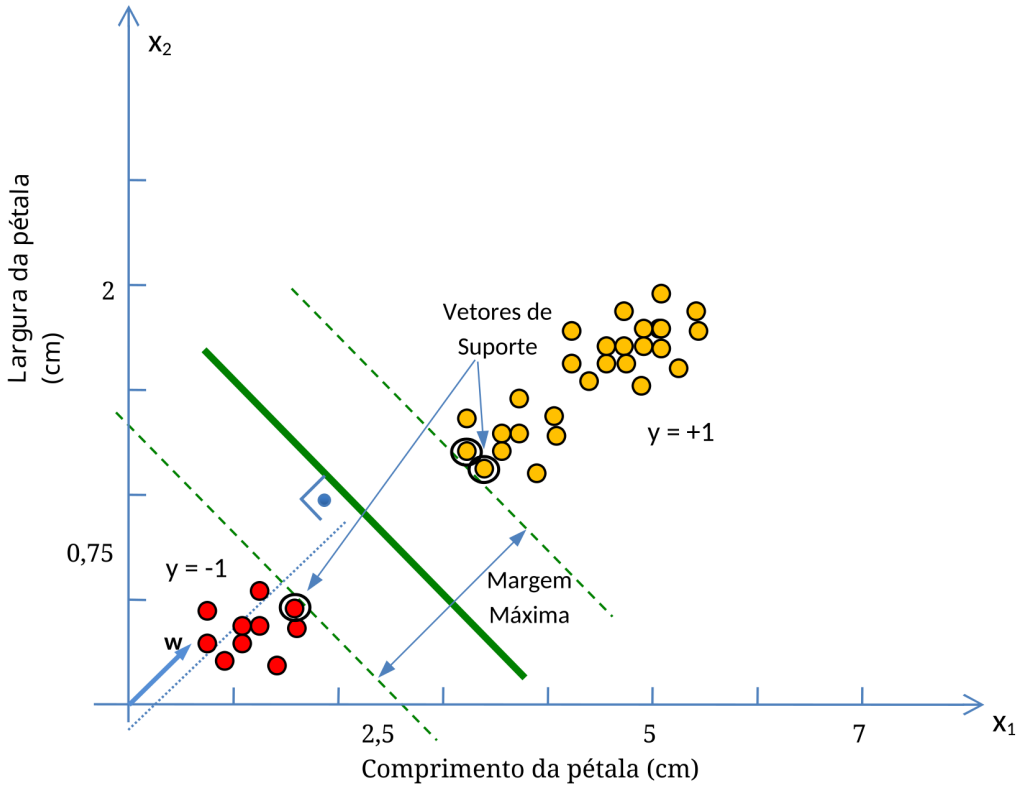


Figura 5.4: Nas MVS, o Melhor Classificador Possui Margem Máxima.

convexos (com as conexões dos pontos internos omitidas para tornar mais clara a apresentação).

A vantagem de se considerar as classes como se fossem dois conjuntos convexos é que na busca pelo menor segmento que une estes dois conjuntos, não ocorre o problema do **mínimo local**. Enquanto a distância entre ambos estiver diminuindo, a direção seguida estará correta.

A analogia para este procedimento é que o **Algoritmo de Aprendizado** pode ser visto como um método de otimização [6] de uma **Função Custo**. A cada iteração do processo de aprendizado a Função Custo é avaliada e, se seu valor decrescer, os pesos do vetor peso **w** são atualizados. O processo continua até encontrar o valor mínimo.

Esta forma de otimização produz bons resultados quando a Função Custo não apresenta mínimos locais, i.e., quando há apenas um **mínimo global**. No caso das Redes Neurais, geralmente a Função Custo apresenta mínimos locais, o que obriga a adoção de mecanismos mais complexos de otimização, como manter alguns registros do que se acredita ser o mínimo global atual, e continuar atualizando os pesos do vetor **w** mesmo que a Função Custo esteja aumentando (Figura 5.6).

Se um novo mínimo for encontrado, os registros antigos são apagados e novos valores de peso são adotados. Embora este procedimento heurístico aumente as possibilidades de detectar o mínimo global, não há nenhuma garantia de que ele será encontrado, a menos que se pague um alto custo computacional com uma busca exaustiva.

5.5 Classificadores Não Lineares

Se as duas classes não forem linearmente separáveis, a teoria de MVS prevê o mapeamento dos pontos do **Espaço de Entrada** para outro espaço de maior dimensão, conhecido como **Espaço de Característica**. A Figura 5.7 mostra uma situação em que os dados não podem ser linearmente separados. Para fazer a separação entre ambas podemos usar não uma reta, mas uma curva descrita por uma função polinomial, ou tentar um recurso matemático conhecido como **Truque do Núcleo** ou ***Kernel Trick***.

O uso de funções polinomiais, além de apresentar risco de alto custo computacional, pode favorecer o ajuste excessivo

[6] Este método é conhecido como **Otimização Convexa**.

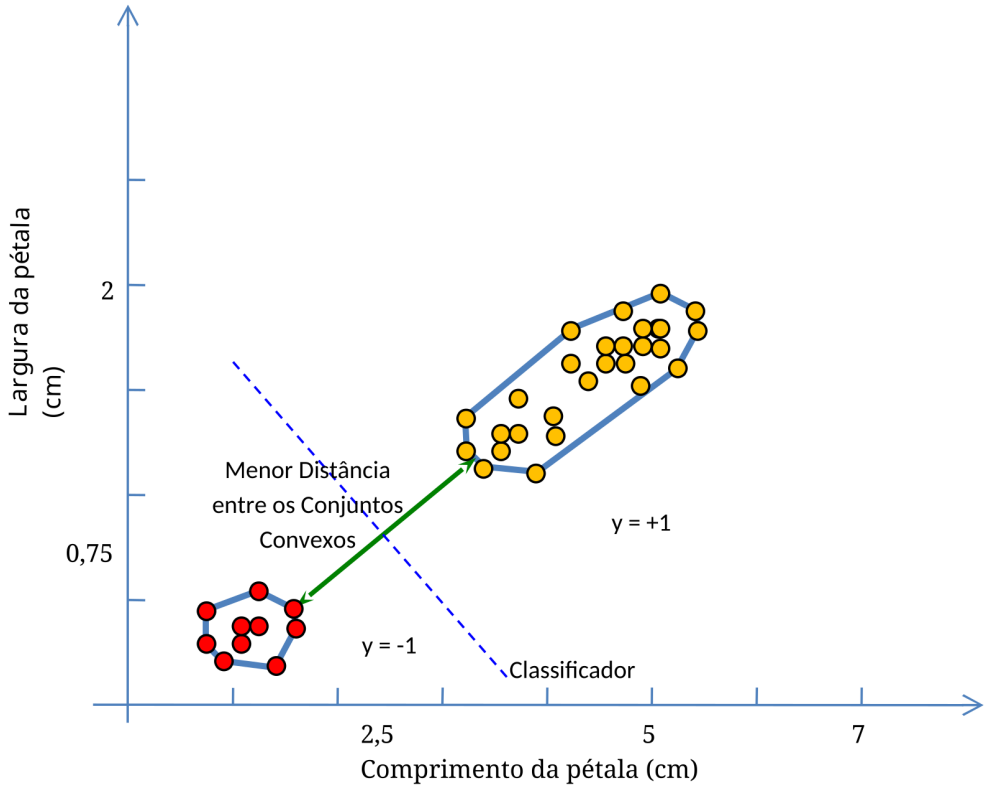
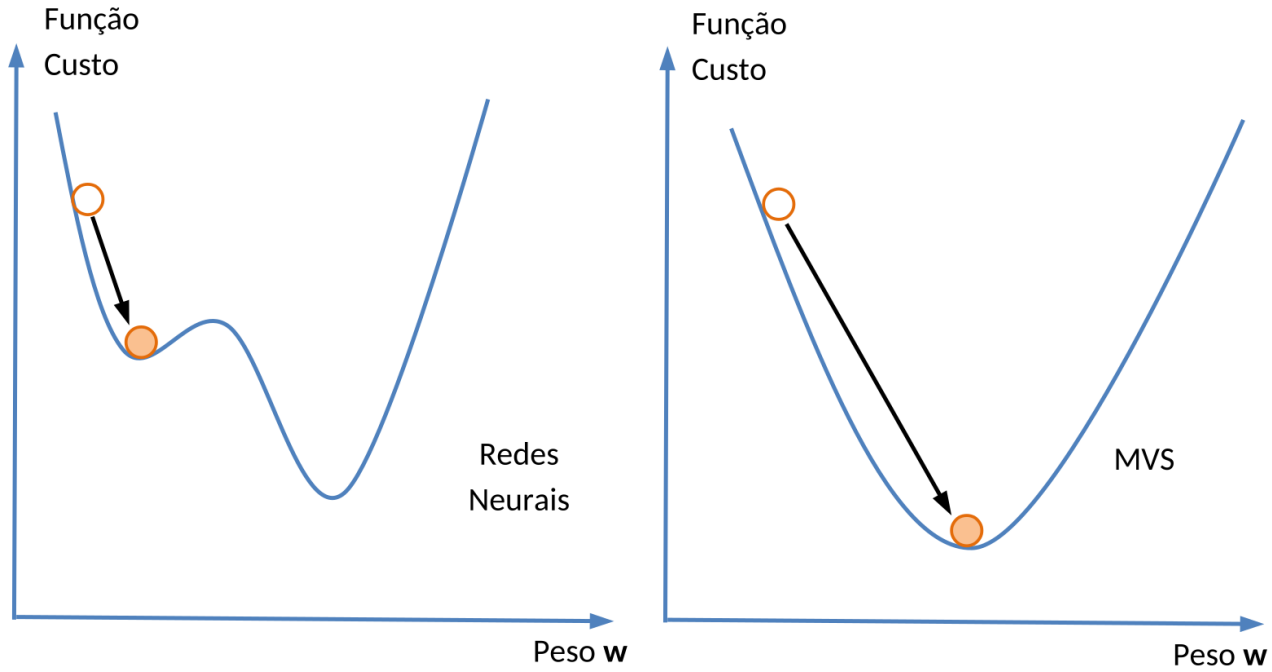


Figura 5.5: Representação das Classes como Conjuntos Convexos.



(a) Com Mínimo Local

(b) Sem Mínimo Local

Figura 5.6: Função Custo (a) Com e (b) Sem Mínimo Local.

da curva polinomial aos Vetores de Treinamento, podendo então ocorrer problemas de ***overfitting*** decorrente da **instabilidade** causada pela enorme influência que um ou outro vetor pode ter sobre a curva polinomial. Já o princípio de **margem máxima** das MVS traz mais **estabilidade** na classificação, diminuindo as chances de *overfitting*. Para entender melhor este comportamento, vamos primeiramente ver o que significa usar um *kernel* [7].

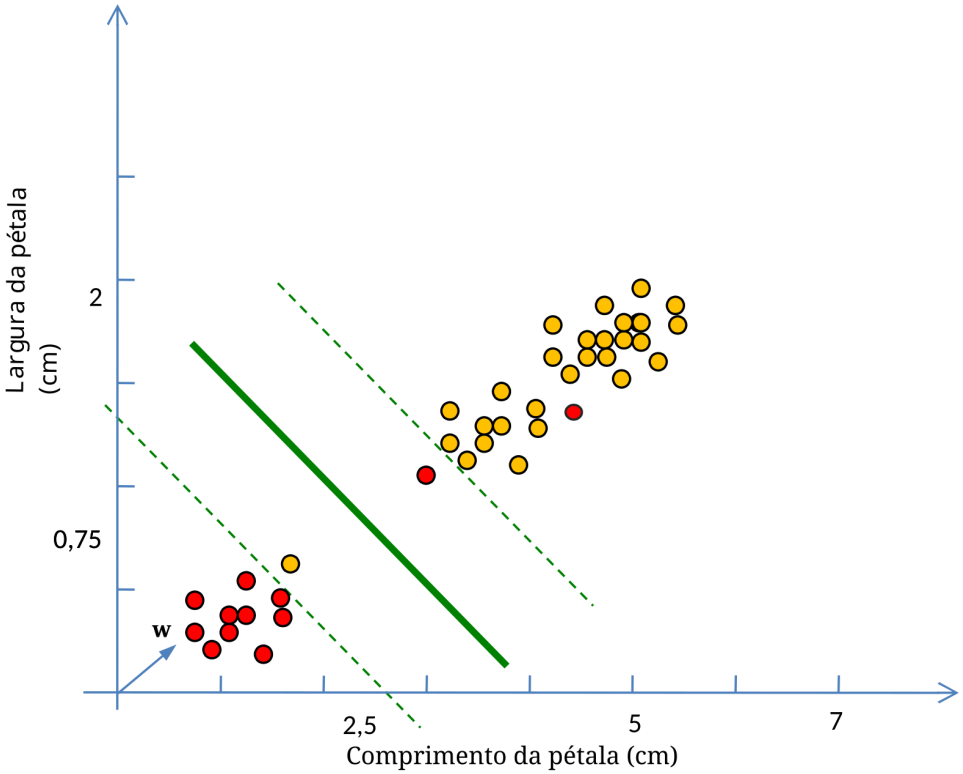


Figura 5.7: Exemplo de Classes Não Separáveis Linearmente.

A ideia por trás dos *kernels* é mapear os Vetores num espaço de dimensão mais elevada que a original. Por exemplo, se o espaço original dos Vetores de Entrada for bidimensional, podemos introduzir algumas redundâncias em suas coordenadas e mapear estes mesmos vetores num espaço tridimensional. Qual o propósito disso? É que duas classes não separáveis linearmente num espaço bidimensional podem se tornar linearmente separáveis num espaço tridimensional, como ilustra a Figura 5.8.

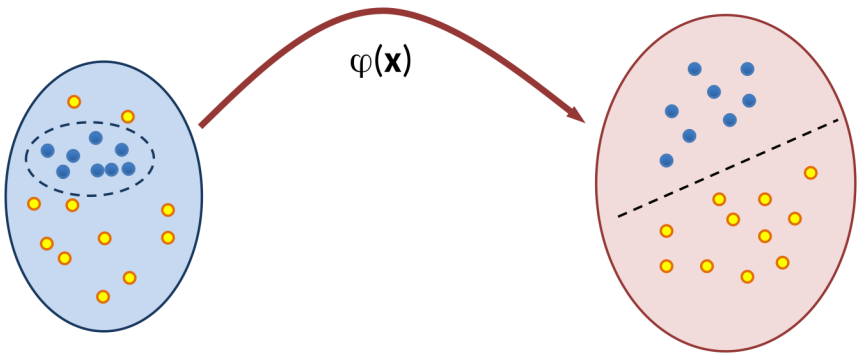


Figura 5.8: Exemplo de Transformação Não Linear do Espaço de Entrada para o Espaço de Características.

Essa transformação em que os vetores são representados num espaço de dimensão mais elevada, geralmente é uma

[7] Embora o termo "*kernel*" possa ser perfeitamente traduzido para o português como "núcleo", a tradução não parece ter prevalecido por aqui, sendo mais comum a utilização de *kernel*.

transformação não linear $\varphi(\mathbf{x})$. O espaço de partida dessa transformação não linear é conhecido como **Espaço de Entrada** e o espaço com dimensão mais alta é conhecido como **Espaço de Características**.

O que se busca com esta transformação é linearizar o Espaço de Características, e com isso tornar os dados linearmente separáveis. Pode parecer paradoxal, mas através de uma transformação não linear é possível linearizar o Espaço de Características.

Vamos reproduzir um exemplo, apresentado por **scholkopf2001**, ilustrando como esta transformação não linear pode ser feita. Suponha um **Espaço de Entrada bidimensional**, composto por duas classes não separáveis linearmente. Aplicando-se o truque do *kernel*, vamos mapear esses dados num **Espaço de Características tridimensional**, como ilustrado na Figura 5.9.

Observando-se o resultado da operação mostrada na Figura 5.9, verifica-se que a **transformação não linear** $\varphi(\mathbf{x})$ não criou novas variáveis; ainda estamos trabalhando com x_1 e x_2 no início e no fim da transformação. As variáveis z_1, z_2 e z_3 são apenas elementos intermediários que ajudam no entendimento da transformação operada nos dados. Isso significa que não será necessário computar explicitamente a transformação $\varphi(\mathbf{x})$ para obtermos no Espaço de Entrada resultados semelhantes àqueles que obteríamos se as operações tivessem sido feitas no Espaço de Características.

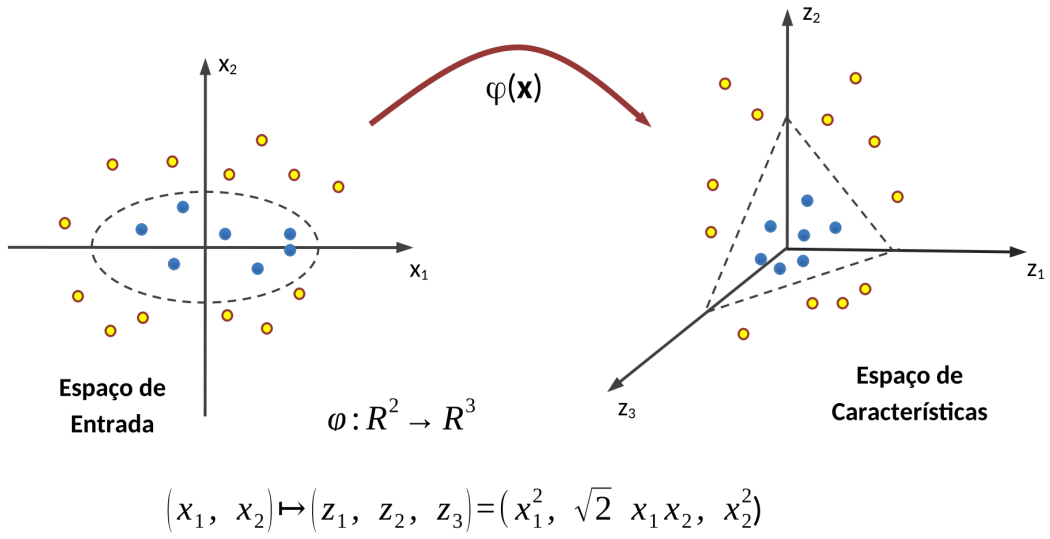


Figura 5.9: Exemplo de Transformação Não Linear do Espaço de Entrada para o Espaço de Características.

O **truque do *kernel*** permite que as operações de **produto interno** entre dois vetores no Espaço de Características sejam computadas como se ainda estivéssemos no Espaço de Entrada. Como, para efeitos práticos, não houve efetivamente aumento da dimensão do espaço onde ocorrem as operações de produto interno, o truque do *kernel* oferece a possibilidade de escapar das complicações que as operações no Espaço de Características poderiam trazer. Por isso, os matemáticos dizem que *kernels* podem ajudar fugir da **Praga da Dimensionalidade**.

Existem vários *kernels* conhecidos (*Polynomial*, RBF etc.), alguns desenvolvidos para aplicações específicas, como *kernels* para bioinformática, para classificação de imagens ou caracteres etc. Como as MVS se enquadram no paradigma do aprendizado supervisionado, isso significa que existe um Conjunto de Vetores de Treinamento, cujos rótulos de classe são previamente conhecidos. Portanto, com um pouco de teoria e uma dose de experimentação é possível desenvolver *kernels* que aumentem a taxa de sucesso durante o treinamento.

Se o Conjunto de Treinamento for uma amostra representativa dos Vetores de Teste, então uma alta taxa de sucesso no treinamento possivelmente significará boa capacidade de generalização para o teste. Caso contrário, não será possível fazer previsões confiáveis com relação à taxa de sucesso nos testes.

5.6 Margem Suave Máxima

Há casos porém em que mesmo com a linearização do Espaço de Características, obtida com o truque do *kernel*, as classes continuarão não separáveis linearmente. O que fazer nessa situação?

A solução adotada pelas MVS foi criar uma "**margem suave**" que admite ruídos, mas estabelece uma penalidade para cada caso de classificação equivocada. Considere um parâmetro C, chamado de **Parâmetro de Complexidade** C, que aplica uma pena a cada vetor classificado erroneamente. A Figura 5.10 ilustra diversas situações envolvendo violações das bordas de classificação.

Se considerarmos primeiramente as margens verdes tracejadas do classificador da Figura 5.10, notamos que embora os vetores 1 e 2 tenham violado os limites das margens verdes, eles estão classificados do lado correto do plano. Os vetores 3, 4 e 5, por sua vez, estão do lado oposto ao que deveriam estar. Portanto, a penalização para estes casos distintos deve ser diferenciada, de acordo com a distância da margem.

O ajuste do **parâmetro** C ajuda a encontrar um **compromisso entre a tolerância a vetores classificados erroneamente** devido a margens amplas (valor de C baixo) **e a minimização dos erros de treinamento** (valor de C alto, e margens estreitas). As margens verdes correspondem a um Fator de Complexidade C baixo ($\sim$1), enquanto que as margens vermelhas correspondem a um valor de C alto ($\sim$10).

Note que a minimização dos erros de treinamento nem sempre é uma garantia de elevada taxa de sucesso na fase de testes. Isso depende de quão representativo é o Conjunto de Treinamento com relação ao Conjunto de Teste, e do *kernel* escolhido. Para encontrar o melhor valor de C, geralmente são coletados vários resultados empíricos usando o método da *cross-validation*.

A Figura 5.10 ainda ilustra o fato de que, uma vez encontrada a reta de separação dos dados, apenas os Vetores de Suporte (aqueles pontos que tocam as margens) passam a ter importância para a fase de testes. Todos os demais Vetores de Treinamento podem ser desconsiderados porque eles não têm qualquer influência sobre as margens. A implicação desse fato para o desempenho do classificador é decisiva, uma vez que para classificar um novo ponto no plano, basta considerar os Vetores de Suporte, que são os delimitadores das margens.

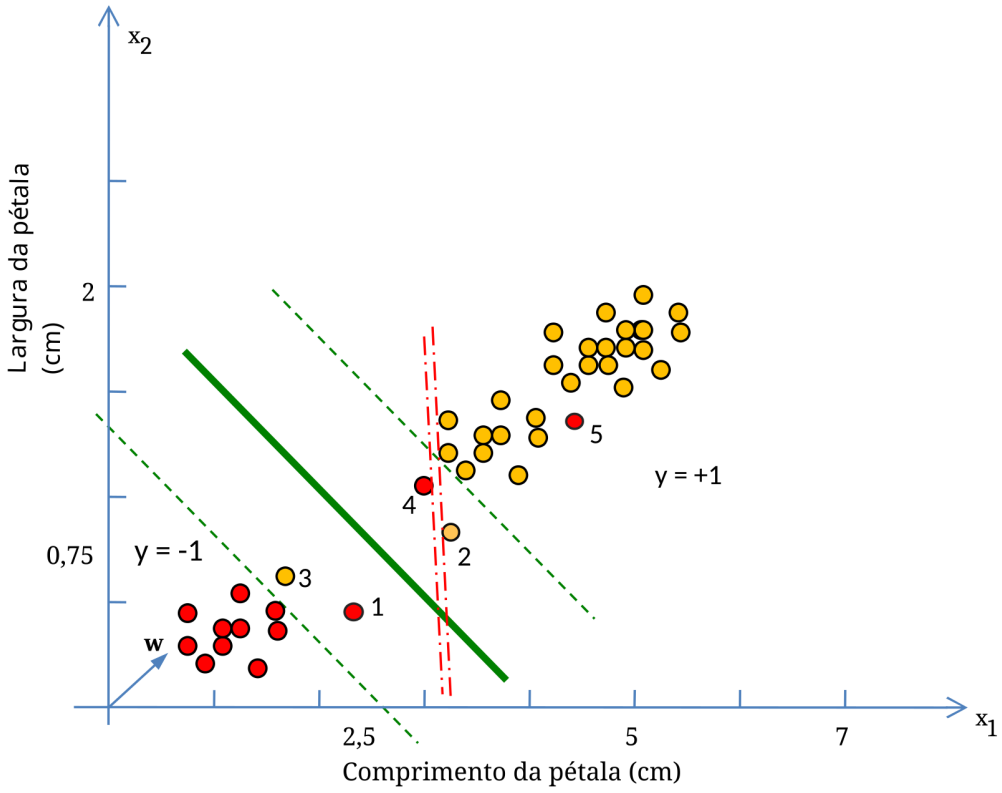


Figura 5.10: Diferentes Margens com Diferentes Capacidades de Generalização.

5.7 Ferramental Matemático Básico

Vamos agora, de forma sucinta, dar uma ideia de como as bordas de decisão de uma MVS podem ser obtidas matematicamente num espaço bidimensional. Considere o plano cartesiano da Figura 5.11 e os respectivos vetores **x** e **w**.

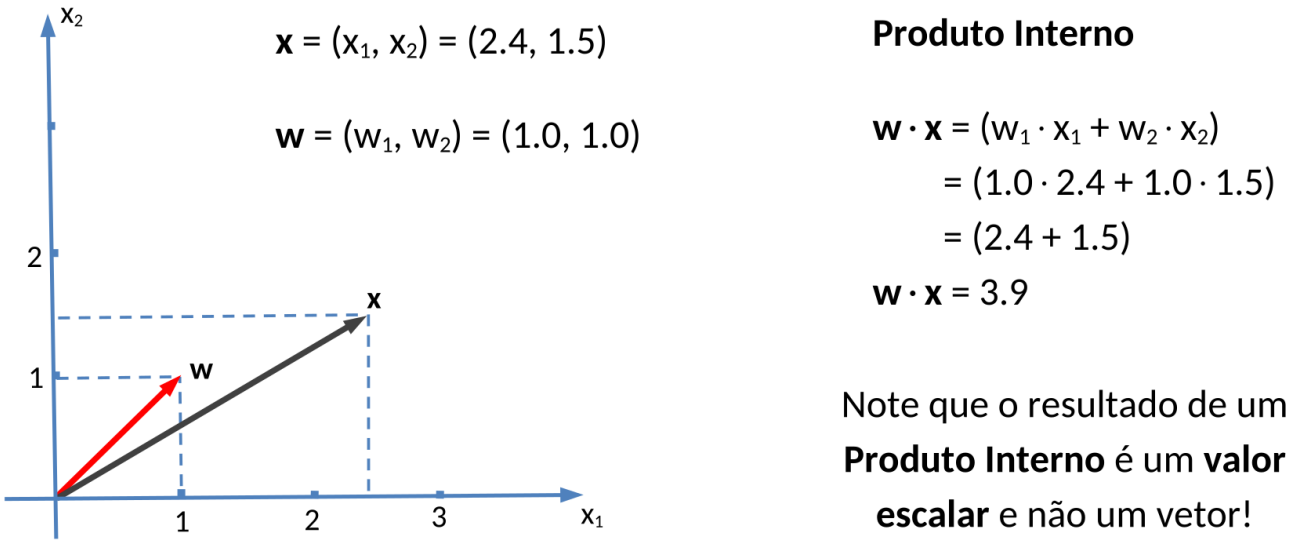


Figura 5.11: Produto Interno dos Vetores $\mathbf{w}$ e $\mathbf{x}$.

O **Produto Interno** de dois vetores, também chamado de **Produto Escalar** ou **Produto de Ponto**, gera como resultado uma grandeza escalar, ou seja, um número real. Para sua representação, podemos usar tanto um ponto, $\mathbf{w} \cdot \mathbf{x}$, quanto os sinais de maior e menor, $\langle \mathbf{w}, \mathbf{x} \rangle$. As letras em negrito representam um vetor, enquanto que as letras simples representam uma grandeza escalar. Usando Produto Escalar, qualquer segmento de reta pode ser representado no plano da forma mostrada na Figura 5.12.

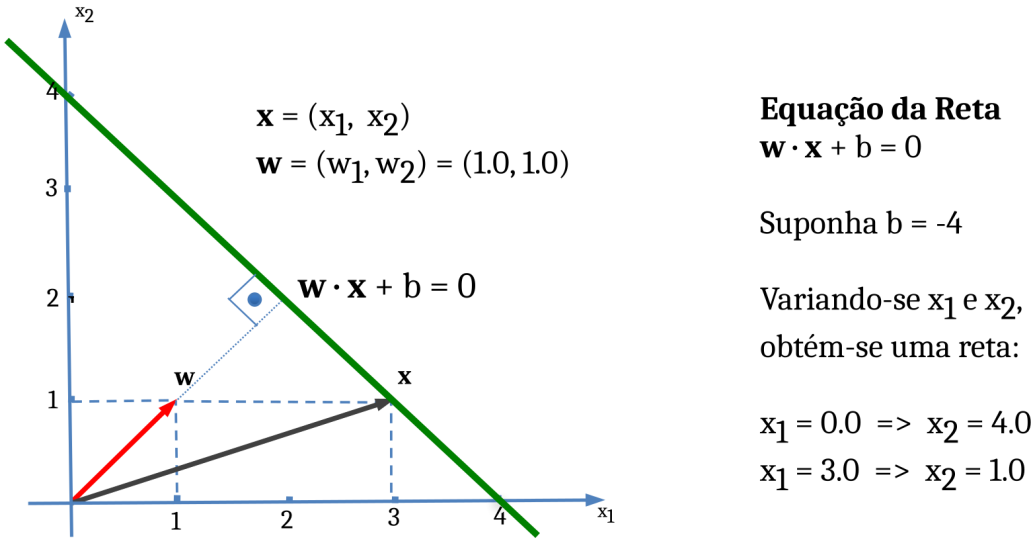


Figura 5.12: Equação da Reta: $\mathbf{w} \cdot \mathbf{x} + b = 0$.

Para representar uma reta e suas margens, usamos o mesmo procedimento (Figura 5.13).

Como estamos interessados em maximizar as margens, ou seja, tornar o módulo da subtração de $\mathbf{x}_1$ por $\mathbf{x}_2$, $\|\mathbf{x}_1 - \mathbf{x}_2\|$, o mais largo possível, isso equivale a minimizar o módulo do vetor peso $\mathbf{w}$, já que

$$\|\mathbf{x}_1 - \mathbf{x}_2\| = \frac{2}{\|\mathbf{w}\|} \tag{5.2}$$

A Equação 5.2 mostra que quanto menor o módulo de $\mathbf{w}$, mais largas serão as margens!

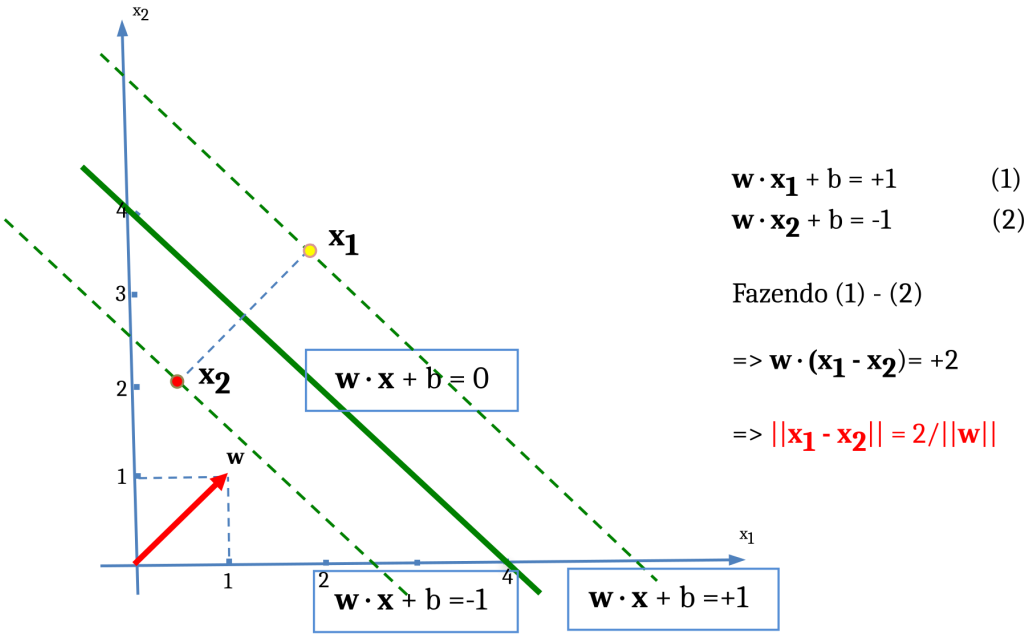


Figura 5.13: Equação do Classificador e Margens.

Dessa forma, a **Função de Aprendizado** de uma MVS consiste em "testar" coordenadas para $\mathbf{w}$ de tal modo que seu módulo decresça a um mínimo, fazendo com que as margens se alarguem, até o limite em que elas toquem os primeiros Vetores de Treinamento nas bordas de decisão. Quando isso ocorrer, estes Vetores de Treinamento que foram atingidos pelas margens do classificador se tornam os **Vetores de Suporte** da MVS.

Existem técnicas matemáticas de otimização deste problema, por exemplo a **Programação Quadrática**, bastante conhecida entre os matemáticos, e que foram aproveitadas por **cortes1995** para desenvolver o algoritmo da MVS. Sucintamente o problema pode ser formalizado da forma exposta a seguir.

Dado um **Conjunto de Treinamento**: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$

O **Hiperplano com Margem Máxima** é definido pelo par $(\mathbf{w}, b)$ que resolve o seguinte problema:

$$\text{Minimize:} \quad \|\mathbf{w}\|^2$$

$$\text{Sujeito a:} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \forall\, i = 1, \ldots, N$$

Mais detalhes de como solucionar a forma dual desse problema de Programação Quadrática podem ser obtidos na referência bibliográfica desta unidade de estudo.

Há uma interessante implementação de MVS, proposta por **platt1998**, conhecida como **SMO** (*Sequential Minimal Optimization*), que em vez de considerar todos os Vetores de Treinamento conjuntamente, eles são divididos em pares e seus valores ótimos são deduzidos analiticamente. Embora o número de operações matemáticas aumente com relação à forma mais tradicional de resolver numericamente o problema com um sistema de equações, estas operações matemáticas da forma analítica são simples, operações aritméticas básicas, e portanto muito rápidas num computador. Dessa forma, o ganho no tempo total de computação pode ser significativo.

Outro atrativo muito interessante da implementação SMO é que não é necessário carregar simultaneamente todos os Vetores de Treinamento na memória principal do computador. Considerando aplicações reais, com centenas de milhares ou milhões de Vetores de Treinamento, o algoritmo SMO pode ser o mais indicado.

5.8 Como Visualizar as Bordas de Decisão de uma MVS Usando o Weka

A ferramenta Weka (**frank2016weka**) permite rodar várias implementações de **SVM**, ajustar o **Parâmetro de Complexidade** C, entre outros, e visualizar as **Bordas de Decisão** obtidas.

Passo 1 — Primeiramente vamos carregar o arquivo "iris.arff" no Weka e eliminar dois de seus Atributos, para tornar os resultados visualmente mais interessantes. Carregue no "Weka Explorer" o arquivo "íris.arff" (anexo), selecione os Atributos "sepallength" e "sepalwidth", e dê um clique em "Remove", como mostra a Figura 5.14.

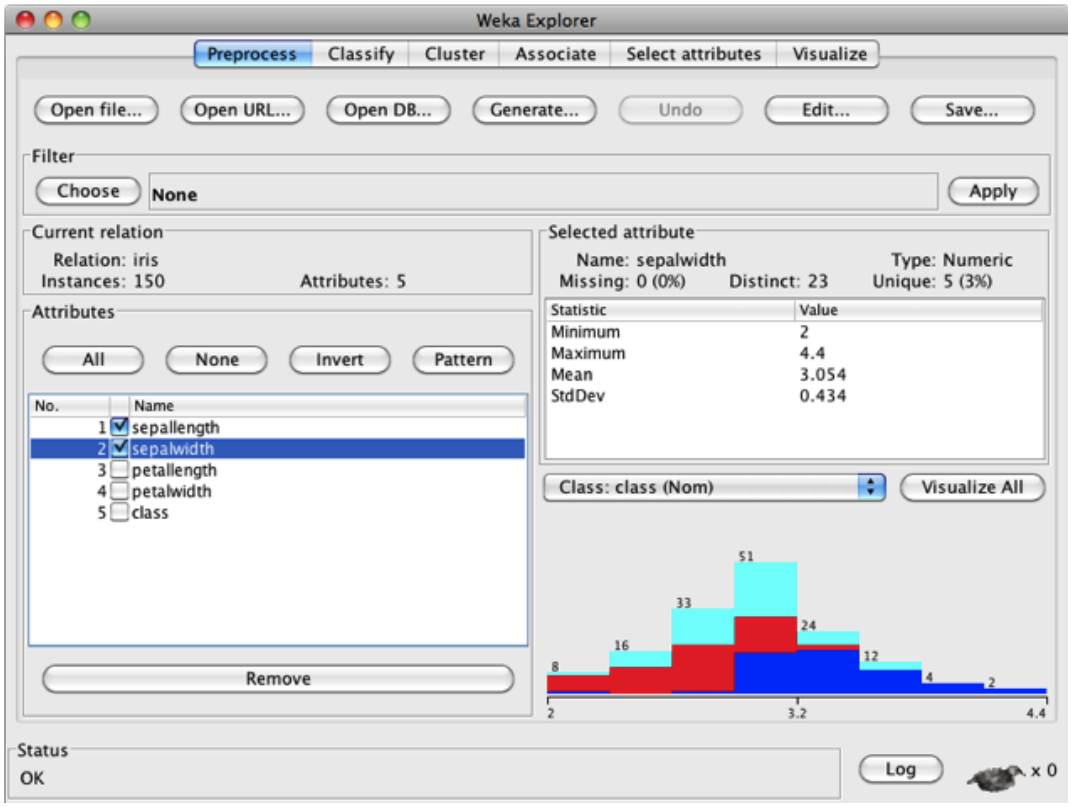


Figura 5.14: Remoção de Atributos de um Arquivo ".arff".

Salve o novo arquivo com o nome "iris_mod.arff", dando um clique no botão "Save..." (no canto superior direito do "Weka Explorer").

Passo 2 — Vamos abrir o arquivo modificado "íris_mod.arff" no "Weka GUI Chooser" (Figura 5.15), clicando primeiro na opção "Vizualization" e depois em "BoundaryVisualizer".

Uma janela semelhante à mostrada na Figura 5.16 deve aparecer.

Passo 3 – Clique no Botão "Open File", localize o arquivo "íris_mod.arff" e carregue no "BoundaryVisualizer. Os Vetores de Treinamento (representados por pontos) devem aparecer na tela de visualização do"BoundaryVisualizer" (Figura 5.17). (Como o "BoundaryVisualizer" leva em conta os ajustes da última vez em que ele foi utilizado, a imagem que aparece na tela pode variar de simulação para simulação.).

Passo 4 – Na parte superior direita, em "Classifier", clique em "Choose" (Figura 5.17), escolha "Classifiers" , depois "functions" e, finalmente "SMO", como ilustra a Figura 5.18. Inicialmente deixe os valores default do SMO.

Marque a opção "Plot training data" (Figura 5.18), na parte inferior direita, para que ao final da simulação apareçam não apenas as bordas de decisão do classificador, mas também os dados usados no treinamento. Dê um clique em "Start" (Figura 5.18).

Lentamente as bordas de decisão do classificador vão se formando, enquanto uma linha horizontal corre a tela indicando que a simulação está em progresso.

Figura 5.15: Interface do "Weka GUI Chooser".

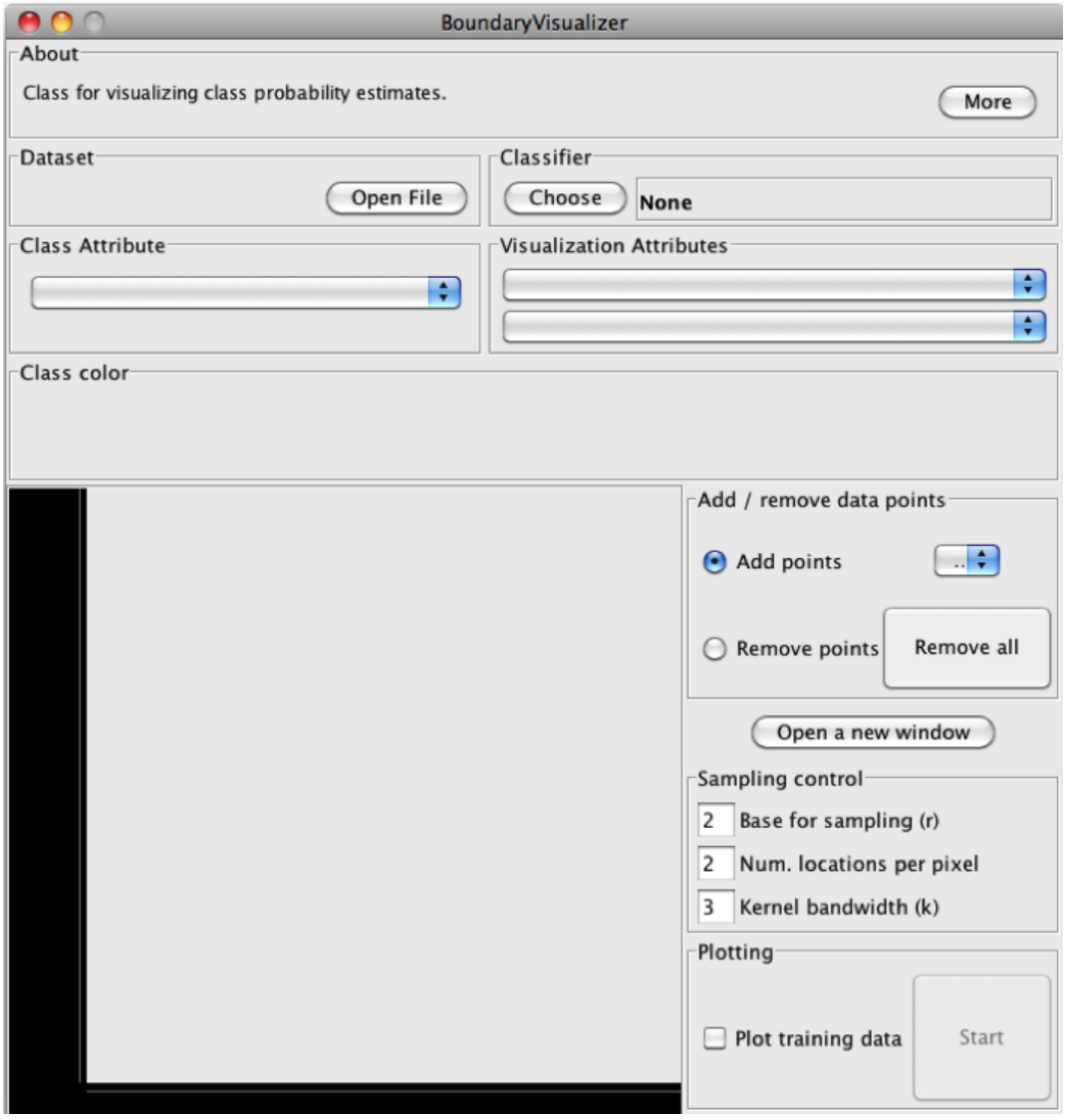


Figura 5.16: Janela do "BoundaryVisualizer".

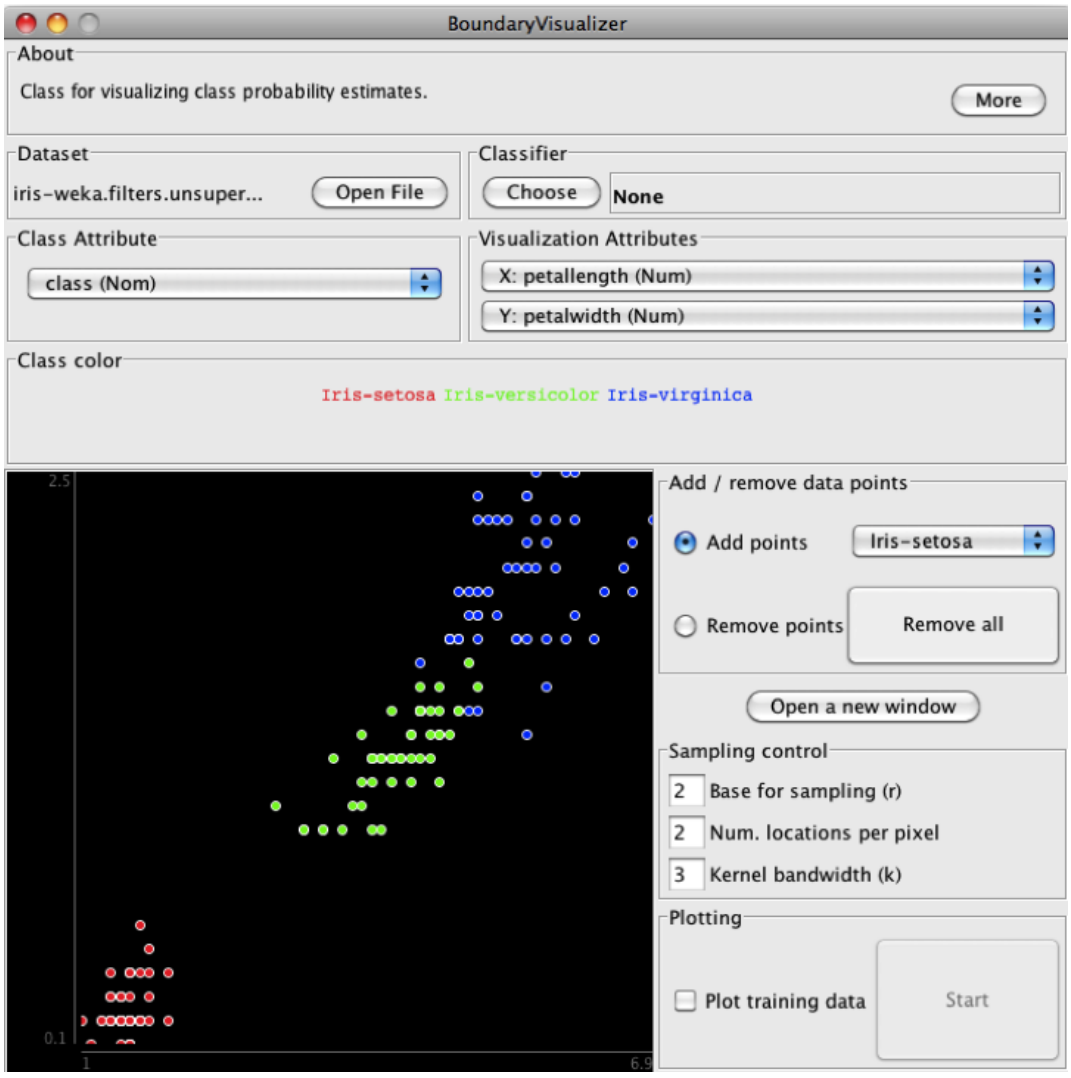


Figura 5.17: Dados do Arquivo “íris_mod.arff” na Tela do “BoundaryVizualizer.

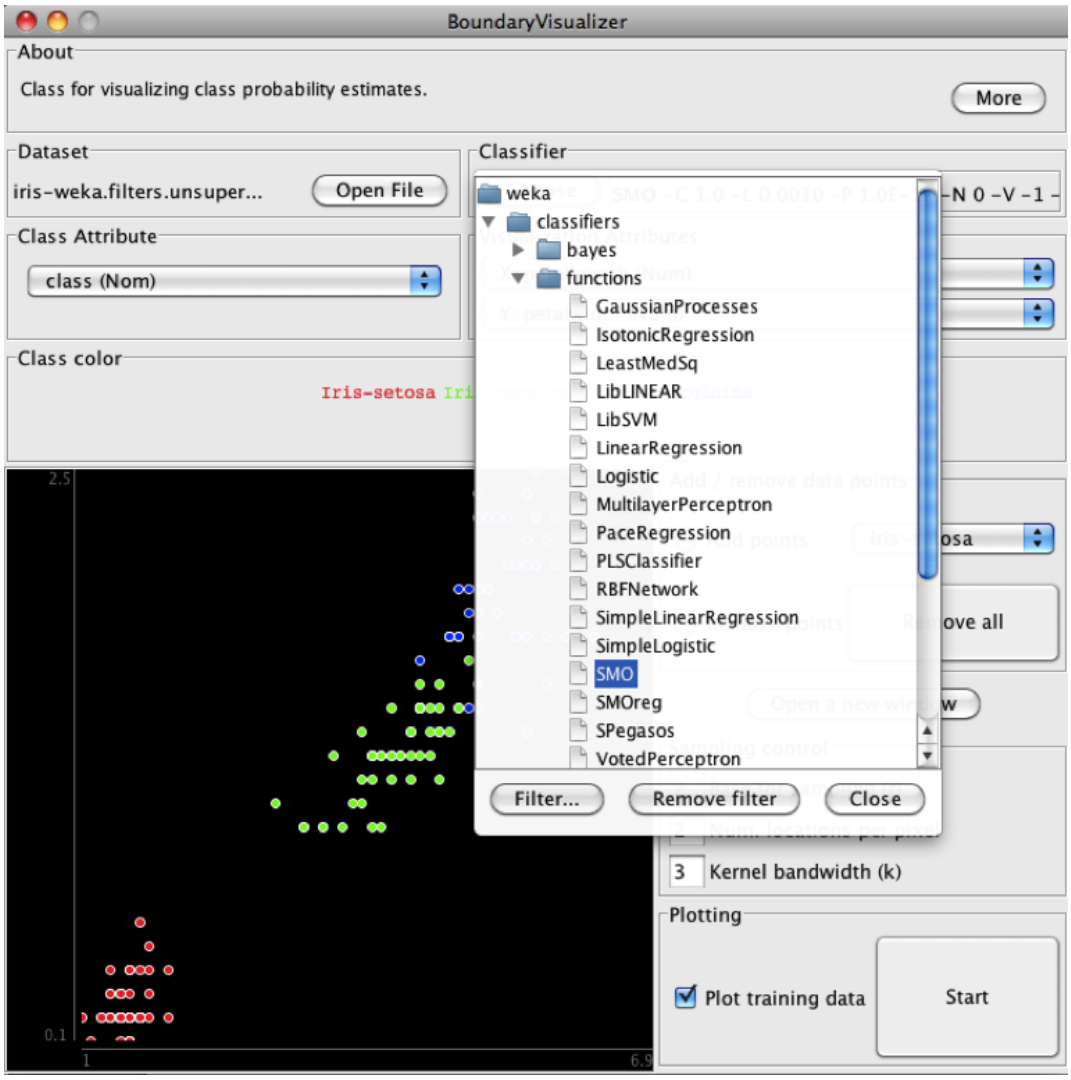


Figura 5.18: Escolha do Algoritmo SMO.

Passo 5 – O número de Vetores de Treinamento classificados erroneamente deve ser em torno de 6. Vá com o mouse novamente na parte superior direita do "BoundaryVisualizer" e clique com o botão da esquerda sobre a palavra "SMO", ao lado de "Choose", no campo "Classifieres". Uma janela de ajustes dos parâmetros do SMO, semelhante à mostrada na Figura 5.19, deve se abrir.

Figura 5.19: Janela de Ajustes de Parâmetros do SMO.

Nesta janela, no botão "More" há uma explicação sucinta de todos parâmetros mostrados.

Passo 6 – Faça novas simulações variando o valor default do Parâmetro de Complexidade C de 1.0 para 2.0, 5.0, 30.0 etc. e confira o número de Vetores de Treinamento que continuam classificados erroneamente. Com ajuste de C = 2.0 e kernel "PolyKernel" foi rodada uma simulação, cujos resultados são mostrados na Figura 5.20.

Quando o valor do parâmetro C foi alterado para 91, as Bordas de Decisão se alteraram, como mostra a Figura 5.21.

Como pode ser observado, há dois ou três pontos verdes e azuis que se parecem com "outliers". Vamos retirar alguns deles, ativando a opção "Remove points" (Figura 5.21), e ver como a Borda de Decisão se altera. A Figura 5.22 mostra o resultado da remoção de alguns pontos azuis da tela, usando o botão esquerdo do mouse.

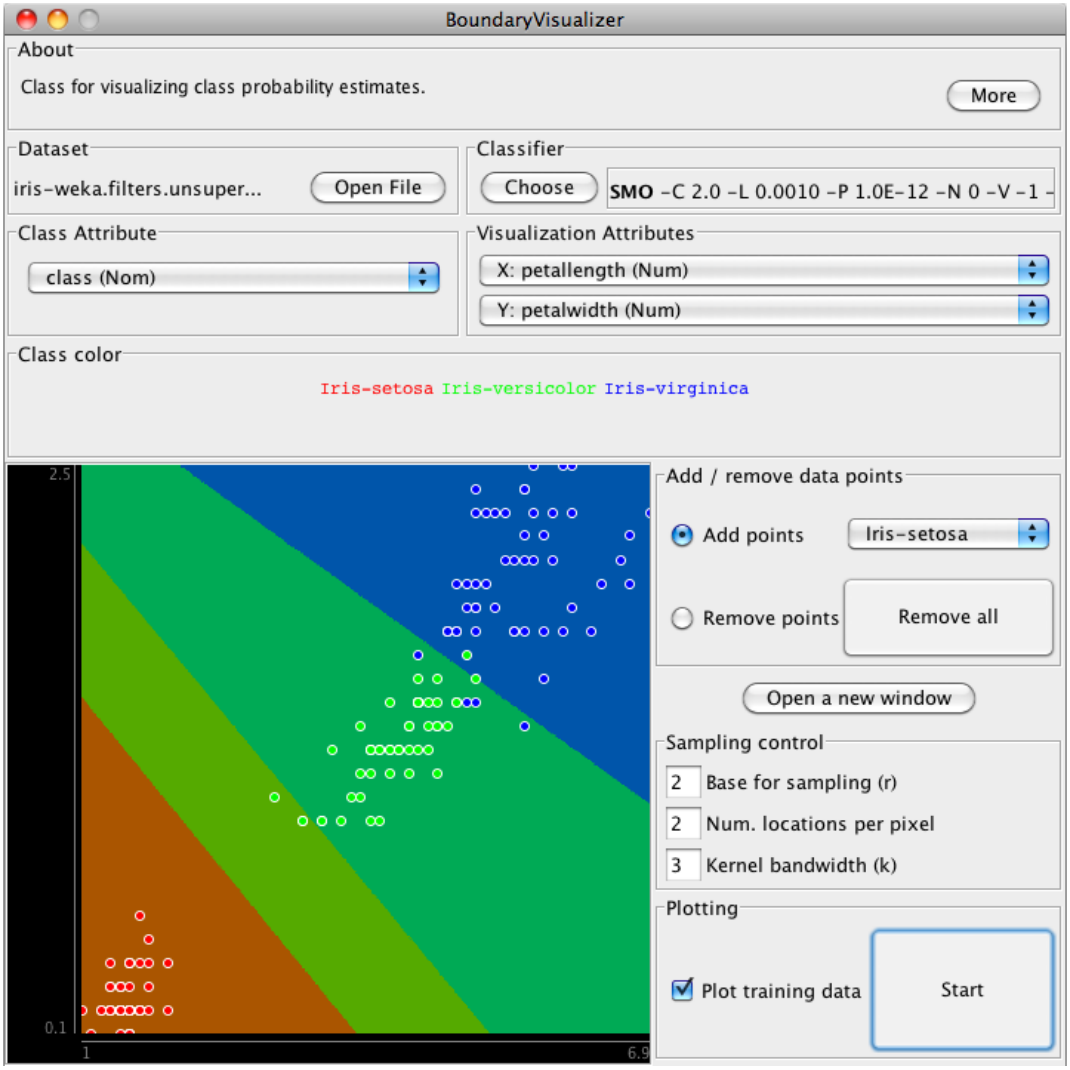


Figura 5.20: Bordas de Decisão da Simulação para C = 2.0 e kernel “PolyKernel”.

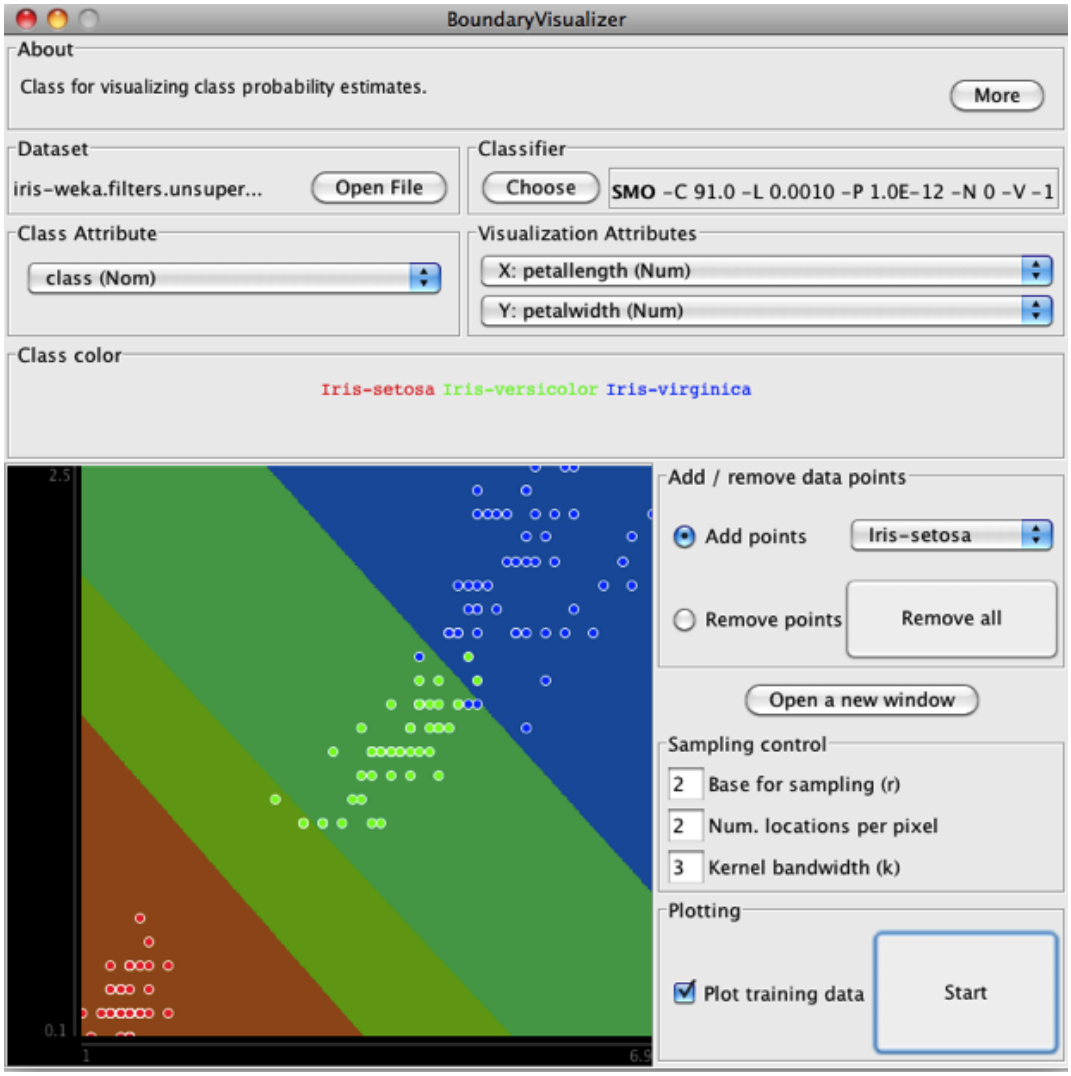


Figura 5.21: Bordas de Decisão da Simulação para C = 2.0 e kernel “PolyKernel”.

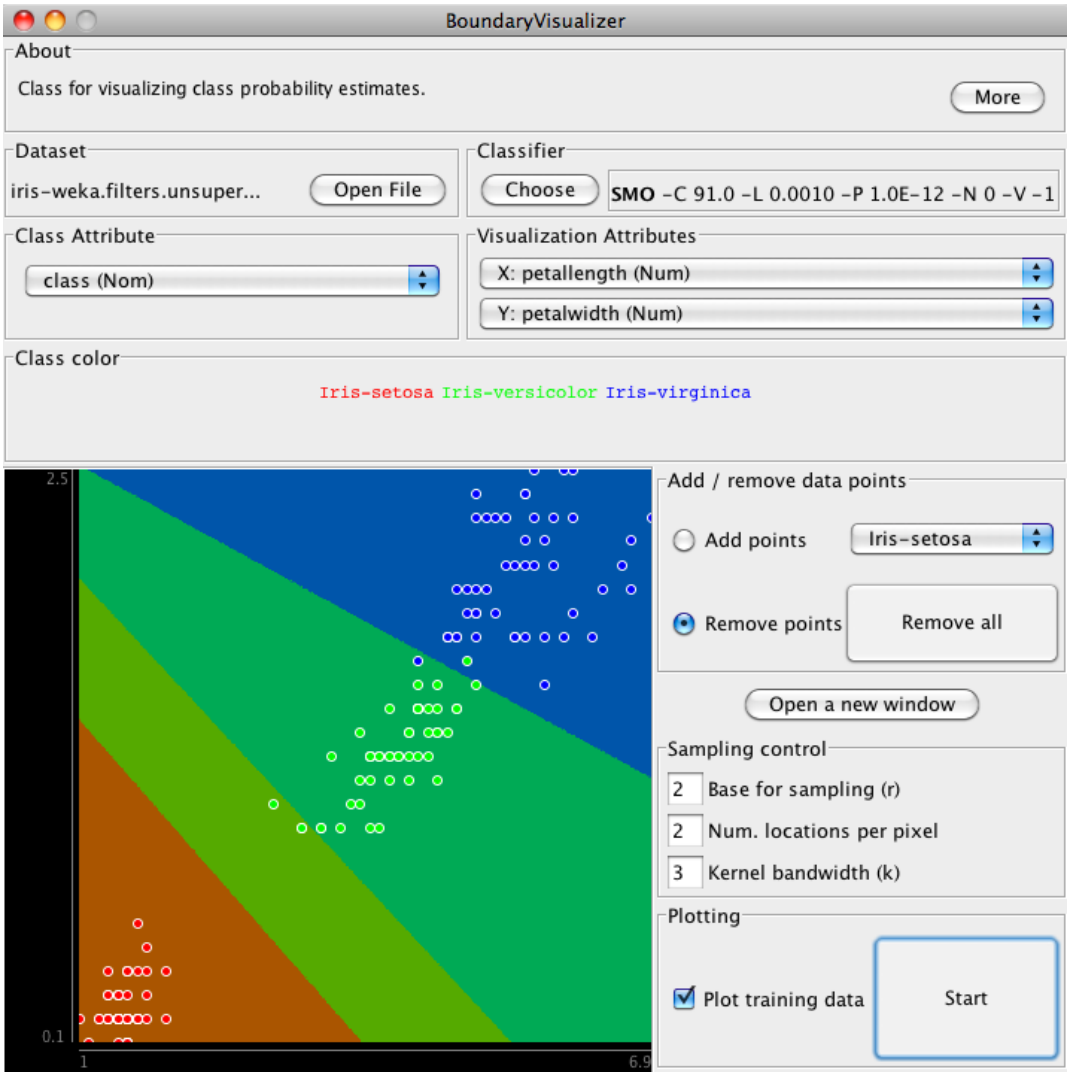


Figura 5.22: Simulação com a Remoção de Alguns Pontos Perto das Bordas.

Como pode ser observado na Figura 5.22, a Borda de Decisão superior sofreu uma alteração significativa, mostrando que Vetores de Treinamento próximos das Margens têm um peso muito grande nos resultados obtidos.

Passo 7 – Novos kernels podem ser escolhidos entre os ajustes do SMO. As cores do painel do "BoundaryVisualizer" podem ser alteradas, clicando sobre os nomes das três "iris" no campo intermediário "Class color". Outros ajustes interessantes podem ser feitos no "BoundaryVisualizer" e testados em novas simulações. Bom trabalho!

5.9 Considerações Finais

Nesta unidade, apresentamos um tipo de algoritmo de **Aprendizado Estatístico** conhecido como **Máquina de Vetores de Suporte**.

Entre suas grandes vantagens, podemos citar:

- Por ser um **algoritmo determinístico**, se uma simulação for > repetida como os mesmos valores iniciais e os mesmos parâmetros, > obtém-se o mesmo resultado (diferentemente do que ocorre com as Redes Neurais).
- Durante o aprendizado não se verifica o problema de **mínimos > locais** na otimização da Função Custo de aprendizado. Há apenas > um **mínimo global** que corresponde ao classificador com **margem > máxima**.
- As MVS estão menos sujeitas ao problema do *overfitting* > comparativamente às Redes Neurais, porque a indução do > classificador é menos sensível à retirada ou acréscimo de um ou > outro Vetor de Treinamento (a menos que seja um Vetor de Suporte. > Na prática, porém, os Vetores de Suporte constituem uma pequena fração dos > Vetores de Treinamento).

Comparando com os algoritmos de Aprendizado Orientado a Conhecimento, as MVS apresentam tempo de treinamento mais elevado, porém sua taxa de acerto costuma ser mais elevada, porque o princípio de Margem (Suave) Máxima pode ser funcionalmente equivalente a uma borda de decisão não linear.

Como desvantagem com relação aos algoritmos de Aprendizado Orientado a Conhecimento, podemos citar a dificuldade em interpretar os resultados do aprendizado, que na prática é um conjunto de pesos do vetor **w**, ou algo equivalente. Outro aspecto que pode ser visto como uma desvantagem das MVS em relação a outros modelos, é que não é possível incorporar Conhecimento do Domínio do Problema no Modelo gerado. Numa Rede Neural, por exemplo, a topologia da rede reflete o Conhecimento do Domínio do Problema. Além disso, redes com multicamadas de neurônios podem aprender a ignorar Atributos irrelevantes (Witten & Frank, 2005).

Através de exemplos e interpretações geométricas, tentamos mostrar que o desempenho de uma MVS pode ser substancialmente influenciado pela escolha do ***kernel*** e do **Parâmetro de Complexidade C**. Por se tratar de um algoritmo de aprendizado supervisionado, ou seja, os rótulos das classes dos Vetores de Treinamento já são previamente conhecidos, é possível desenvolver um *kernel* específico para uma aplicação especial (embora não seja algo muito simples) e avaliar empiricamente o desempenho do *kernel* usando o método da *Cross-validation*.

Os argumentos e ilustrações aqui utilizados para explicar o algoritmo das MVS foram baseados num **espaço bidimensional**. No entanto, eles podem ser estendidos para o **espaço tridimensional**, quando então a reta que representa o classificador deve ser substituída por um **plano**. Da forma semelhante, os mesmos argumentos podem ser aplicados a **espaços com mais de três dimensões**, e neste caso o classificador passa a ser representado por um **hiperplano**.

5.10 Lista de Exercícios

1. (20%) Explique **com suas próprias palavras** o que vem a ser um ***kernel*** no contexto de SVM.

2. (30%) Explique **com suas próprias palavras** qual a função do **Fator de Complexidade *C*** e como ele pode afetar tanto a taxa de erros de treinamento como a de classificação.

3. Carregue o arquivo **"iris.arff"** (anexo) no Weka e faça a classificação com uma **Máquina de Vetores de Suporte** usando o algoritmo **"*SMO*"**, (clique na aba "*Classify*", depois "*Choose*", escolha "*functions*" e clique em "SMO") com o método ***"Cross-validation"*** (10 *Folds*).

(a) (30%) – Escolha valores para o **Fator de Complexidade *C*** entre 1 e 5 e analise os resultados da Matriz de Confusão. (Para ajustar o valor de ***C***, clique com o botão esquerdo do mouse sobre a palavra "*SMO*", ao lado de "*Choose*", e mude o valor de **C** na janela que deve se abrir).

(b) (20%) – Compare o desempenho dos *kernels* "***PolyKernel***" e "***RBFKernel***". (Para escolher o tipo de *kernel*, clique com o botão esquerdo do mouse sobre a palavra "*SMO*", ao lado de "*Choose*", e escolha o *kernel* na janela que deve se abrir).

5.11 Referências do Capítulo

Este capítulo baseou-se nos fundamentos teóricos e práticos que consolidaram o aprendizado de máquina e a mineração de dados. A base histórica das redes neurais seguiu o trabalho clássico de **rosenblatt1957** sobre o Perceptron. Para a fundamentação de Support Vector Machines (SVM), apoiamo-nos no artigo seminal de **cortes1995** e nas otimizações práticas propostas por **platt1998**, complementados pela teoria de kernels detalhada em **scholkopf2001**. Os conceitos estruturais do processo de descoberta de conhecimento e algoritmos foram fundamentados nas obras de **witten2005**, **witten2005**, **tan2009** e **rocha2008analise**, utilizando como suporte técnico a ferramenta WEKA (**frank2016weka**). Por fim, a contextualização em sistemas inteligentes utilizou a organização de **rezende2005**.