Aprendizado semissupervisionado aplicado ao problema de valores ausentes

Glauder Guimarães Ghinozzi

SERVIÇO DE PÓS-GRADUAÇÃO DA UFMS

Data de Depósito:

Assinatura:_____

Aprendizado semissupervisionado aplicado ao problema de valores ausentes[1]

Glauder Guimarães Ghinozzi

Orientador: *Prof Dr Edson Takashi Matsubara*

Dissertação apresentada a Faculdade de Computação - FACOM-UFMS, como parte dos requisitos necessários à obtenção do título de Mestre em Ciência da Computação.

UFMS - Campo Grande
maio/2012

[1]Trabalho Realizado com Auxílio da CAPES

Ao meu filho Enzo

A essa luz, que mudou para sempre nossas vidas no pouco tempo em que esteve conosco.

Agradecimentos

Agradeço primeiramente a DEUS, por ter-me criado, dotado de inteligência e sabedoria, proporcionando estar aqui neste momento. À minha companheira e esposa Lígia Mabel, por estar sempre ao meu lado não somente durante toda esta fase, mas em todos os momentos de minha vida, sempre me incentivando e apoiando. Com você, divido os louros deste trabalho. Ao meu filho Téo, que faz cada amanhecer ser um novo e emocionante dia. Ao Professor Edson Takashi Matsubara, meu orientador, pela oportunidade, pelo seu trabalho, dedicação e paciência oriental. Pelas idéias, críticas e suas preciosas sugestões. Aos meus amigos, colegas do LIA, pelo apoio nesta árdua jornada. Aos amigos do mestrado que ainda ficam boa sorte a todos.

"Tenho pensamentos que, se pudesse revelá-los e fazê-los viver, acrescentariam nova luminosidade às estrelas, nova beleza ao mundo e maior amor ao coração dos homens". (Fernando Pessoa)

Resumo

Valores ausentes constituem um problema relativamente comum em bases de dados, devido a isso existem vários métodos para estimá-los. Essa estimativa é denominada imputação de valores ausentes. Os métodos que usam o aprendizado de máquina para realizar a tarefa imputação, costuman ser supervisionados e somente utilizam os exemplos rotulados para indução de hipotese, e desta forma não conseguem usar a informação contida nos dados não rotulados. Perdendo assim, uma potencial fonte de conhecimento. O aprendizado semissupervisionado é um paradigma de aprendizado que pode se valer tanto de dados não rotulados quanto de dados rotulados e desta maneira, teoricamente, obter um melhor desempenho em tarefas de imputação. Isso porque uma base de dados com atributos ausentes pode ser dividida em dados rotulados (exemplos sem atributos com valores ausentes) e não rotulados (exemplos com atributos com valores ausentes). Nesta dissertação é demonstrado que a imputação de dados em bases estáticas é uma tarefa de natureza intrinsicamente transdutiva e por ser a inferência trandutiva um conceito muito próximo ao aprendizado semissupervisionado é esperado que o aprendizado semissupervisionado obtenha um melhor desempenho em tarefas de imputação que outros paradigmas. Os experimentos realizados utilizando algoritmos semissupervisionados multivisão como o TRI-TRAINING com e sem seleção automática do classificador base, mostram resultados promissores quando comparados com os classificadores supervisionados J48, IBK (*k*-NN), naive Bayes e SMO (SVM) . Isso demonstra que o aprendizado semissupervisionado é uma boa opção para tarefas de imputação de valores ausentes.

Palavras-chave: aprendizado semissupervisionado, imputação de valores ausentes, algoritmos multivisão, seleção automática de classificadores.

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO

1

Introdução

1.1 Contextualização e Motivação

Pouco menos de 50 anos atrás, chegava ao mundo a primeira unidade central de processamento em um único chip. Posteriormente, o então presidente da Intel Gordon E. Moore afirmou que o número de transístores dobraria, pelo mesmo custo, a cada ano[1]. Essa afirmação se tornou a famosa Lei de Moore (Moore, 1965), que foi além dos processadores e atingiu os sistemas de armazenamento e memória, permitindo ao ser humano armazenar quantidades de dados brutos jamais vistos na história da humanidade.

Apesar da evolução do hardware ter possibilitado o acúmulo, acesso e a capacidade de processamento a grandes quantidade de dados, ainda é uma tarefa não trivial a extração de informações úteis desses dados. A tarefa de extrair informações valiosas de dados é denominada Mineração de dados (MD) e recebe este nome devido à analogia com a tarefa de mineração, pois o trabalho de um mineiro é extrair minerais preciosos de montanhas de minérios aparentemente sem valor. A mineração de dados para atingir seu objetivo de extrair conhecimento de forma automatizada ou com a menor intervenção humana possível, utiliza algoritmos de Aprendizado de Máquina (AM) para auxiliar o especialista de domínio.

Como a mineração de dados é comumente realizada sobre bases de dados reais, as pesquisas de AM tem-se beneficiado das bases de dados reais disponíveis atualmente, permitindo a criação de novos algoritmos de AM e de novos paradigmas para pesquisas. Todavia, a utilização de bases reais tem seus custos, pois dados reais normalmente possuem problemas e imperfeições que dificultam o uso de algoritmos de AM. Muitos desses algoritmos assumem que os exemplos não contém erros e provém da mesma base de dados, os atributos tem valores conhecidos e o conceito a ser aprendido é preciso, e não varia com o tempo. No entanto é comum encon-

[1] posteriormente revisada para cada 18 meses

trar em grandes bases de dados reais imprecisões, incoerências e valores ausentes, que podem comprometer a qualidade do conhecimento e até mesmo levar à hipóteses falsas.

O foco deste trabalho está no tratamento de um dos problemas que ocorre com grande frequência em bases de dados reais conhecido como valores ausentes. Os valores ausentes costumam ser um obstáculo para a descoberta de conhecimento pois geram problemas como perda de eficiência em processos de busca e tornam mais complexas a manipulação e a análise de dados (Schafer, 1997). Diversos fatores podem ocasionar a ocorrência de valores ausentes em uma base de dados: como falha no preenchimento ou preenchimento incompleto de formulários por operadores, não resposta em pesquisas por entrevistados, a morte de pacientes em uma pesquisa médica antes da coleta de todos os dados, perda de parte dos dados em uma base por falha no sistema de armazenamento e falha em sensores usados para medição, entre outros. Na Tabela[2] 1.1 é exemplificada uma base de dados com valores ausentes.

Tabela 1.1: Tabela com valores ausentes

data	hora	temperatura
28/01/2012	16:00	35,16
28/01/2012	17:00	34,98
28/01/2012	18:00	?
28/01/2012	19:00	?
28/01/2012	20:00	?
28/01/2012	21:00	?
⋮	⋮	⋮
16/03/2012	13:00	28,23
16/03/2012	14:00	28,59

No dia 28/01/2012 ocorre a queima do sensor, o que ocasiona falha na leitura da temperatura até o dia 16/03/2012 quando se realizou a troca do sensor. Como resultado, a base de dados terá valores ausentes por todo este período e possivelmente gerará problemas numa futura tarefa de MD que venha a ser feita utilizando esses dados.

Existem diversas formas de tratar os valores ausentes. A mais utilizada consiste em descartar os registros que possuem valores ausentes com o objetivo de deixar apenas os registros sem valores ausentes. No entanto, o descarte de parte da base de dados pode resultar em grande perda de informação, o que dificulta ou até mesmo inviabiliza a extração de conhecimento. Técnicas mais refinadas como a imputação (Sande, 1983) estimam os valores ausentes. A imputação faz uso de técnicas relativamente simples que vão de uma simples substituição dos valores ausentes pela média ou moda dos atributos, até técnicas que utilizam algoritmos de aprendizado de máquina. Mesmo assim, a imputação pode produzir resultados que possuem uma grande diferença do valor real do atributo, prejudicando a extração de conhecimento. Assim, estimar valores ausentes não é uma tarefa trivial.

A imputação de dados teve grandes avanços nas décadas de 50 e 60. Anderson (1957)

[2]A tabela é fictícia e serve apenas para ilustrar os conceitos.

estimou a máxima verossimilhança de uma distribuição normal multivariada quando alguns dados são ausentes. Os métodos existentes à época só eram capazes de realizar essa estimação com dados completos. Buck (1960) criou um método para estimar valores ausentes em dados multivariados usando computadores que eram uma novidade na década de 60. Afifi e Elashoff (1966) fizeram uma revisão da literatura sobre valores ausentes disponível na época. Todos esses trabalhos são na área de estatística, com o principal objetivo de tratar os dados para que posteriormente métodos estatísticos que não conseguem lidar com valores ausentes pudessem ser utilizados.

Já nas décadas de 70 e 80 surgem vários trabalhos sobre imputação na área de ciências biológicas como os de Gould (1980); Nordheim (1984); Lavori (1992). Nessa área, devido a longa duração e a natureza das pesquisas, ocorrem a morte de pacientes e cobaias antes do fim dos experimentos, gerando valores ausentes nas amostras. Nesta mesma época, Rubin (1987) cria a imputação múltipla que consiste em imputar mais de um valor para um atributo ausente, permitindo estimar o erro do valor imputado e classifica os dados ausentes em três grupos de acordo com a dependência que os atributos posuem ou não entre-si, classificação essa utilizada até os dias de hoje.

Finalmente na década de 90 com os avanços no campo do AM e da MD, surgem os trabalhos de imputação de dados utilizando aprendizado de máquina. van Buuren e Oudshoorn (1999) descrevem um método de imputação multivariada baseado em equações encadeadas denominado MICE (*Multivariate Imputation by Chained Equations*) utililizando uma biblioteca de sua autoria escrita em S-PLUS denominada MDM (*Missing Data Machine*). Já Lakshminarayan et al. (1999) utilizaram os algoritmos autoclass e C4.5 para imputação em bases de dados industriais (grandes bases de dados) com bons resultados. Wu e Barbará (2002) combinaram um modelo logístico e loglinear para imputar os valores ausentes que ocorrem em um cubo de dados.

O valor ausente pode ocorrer em um único atributo, caso denominado ausência univariada, ou em mais de um atributo denominado ausência multivariada. Entretanto, em bases de dados reais, os valores ausentes costumam estar espalhados em varíos atributos diferentes. Os métodos que solucionam esse problema podem ser divididos em duas grandes categorias (van Buuren et al., 2006): *Joint-Modeling* que estima todos os valores de uma única vez usando modelos preditivos como redes Bayesianas, e a imputação iterativa que divide o problema multivariado em problemas univariados resolvendo cada um de forma independente. Em seu trabalho, Castaneda et al. (2009) apresenta um ambiente baseado em agentes utilizando imputação iterativa visando facilitar a realização de experimentos e auxiliar o analista de dados na tarefa de imputação.

Existem vários trabalhos em imputação de dados usando algoritmos supervisionados e não supervisionados, como em Jerez et al. (2010) que utilizou algoritmos de aprendizado de máquina supervisionados e não supervisionados aplicados a uma base com dados reais. Apesar do recente interesse no aprendizado semissupervisionado o número de trabalhos em imputação de dados usando o aprendizado semissupervisionado ainda é pequeno. Bair e Tibshirani (2004) desenvolveram ferramentas utilizando aprendizado semissupervisionado para predizer

a sobrevivência de pacientes com câncer baseado no perfil genético do tumor. Williams et al. (2007) propuseram um algoritmo supervisionado que foi estendido para um algoritmo semissupervisionado com bons resultados. Matsubara et al. (2008) compara o algoritmo Corai, uma implementação do algoritmo Co-training (Blum e Mitchell, 1998), com outros métodos de imputação descritos na literatura obtendo bons resultados com grandes porcentagens de valores ausentes.

O aprendizado de máquina semissupervisionado tem despertado grande interesse na comunidade científica devido à menor necessidade de dados rotulados. Diferentemente dos algoritmos de aprendizado supervisionado que utilizam apenas dados rotulados, e dos algoritmos de aprendizado não supervisionado que utilizam apenas dados não rotulados; o aprendizado semissupervisionado pode realizar o aprendizado utilizando ambos, dados rotulados e não rotulados simultaneamente e com isso possivelmente obter um melhor resultado. O problema da imputação é muito similar a um problema de aprendizado de máquina semissupervisionado, onde existem dados rotulados (valores presentes) e dados não rotulados (valores ausentes). Assim, ao imputar os valores ausentes em um atributo, os dados completos podem ser usados como dados rotulados, e os incompletos como dados não rotulados, o que possivelmente pode melhorar os resultados devido a maior quantidade de informação disponível para extrair conhecimento.

Uma das primeiras formas de aprendizado semissupervisionado, denominado autoaprendizado (*self-learning*), surgiu entre os anos de 60 e 70 (Weston, 2008) e foi provavelmente o primeiro algoritmo a utilizar dados não rotulados na classificação. Também é conhecido como autotreinamento, autorrotulamento, aprendizado dirigido por decisão ou *bootstrapping* (não confundir com o método estatístico de amostragem de mesmo nome). O autoaprendizado consiste em um envoltório algorítmico (*wrapper*) que usa um método de aprendizado supervisionado repetidamente. O classificador inicia o treinamento utilizando os dados rotulados disponíveis e a cada iteração uma parte dos dados não rotulados é rotulada e acrescentada ao conjunto de dados rotulados. Então o método supervisionado é retreinado usando esses novos dados rotulados adicionais e o processo se repete até que todos os dados sejam rotulados. Note que o classificador usa suas próprias predições para seu treinamento, o que pode fazer com que um erro de classificação seja propagado nas outras iterações ferindo o desempenho do classificador. Alguns algoritmos tentam anular este efeito, desrotulando os exemplos erroneamente rotulados. Assim, um problema no autoaprendizado é a convergência, que depende do algoritmo envolvido, mas para alguns algoritmos básicos existem estudos da convergência como em (Haffari e Sarkar, 2007).

A propagação do erro que pode ser causado pelos dados não rotulados tem sido um dos maiores problemas de aprendizado semissupervisionado desde as suas origens. Existem duas linhas de pesquisa trabalhando na solução deste problema. Goldberg e Zhu (2009) utilizaram validação cruzada em amostras da base de dados para selecionar entre classificadores supervisionados ou semissupervisionados e assim tentar criar o aprendizado semissupervisionado seguro. Já Zhu (2011) extraiu o conhecimento dos dados não rotulados e utilizou para gerar novos atributos que auxiliam um classificador supervisionado, criando o fSSL (*formulative Semi Supervised Lear-*

ning). A extração de conhecimento e a criação de novos atributos, ao invés de simplesmente rotular os dados não rotulados, é motivada pelo fato do indutor ser muito mais sensível a ruídos no atributo de classe que em atributos comuns (Zhu e Wu, 2004). Essa abordagem ainda tem a vantagem de permitir que o conhecimento extraído seja utilizado em outras bases de dados do mesmo domínio.

Voltando a década de 70, Vapnik introduziu o conceito da inferência transdutiva ou transdução (Vapnik, 1998) que é um conceito muito próximo ao aprendizado semissupervisionado, desenvolvendo posteriormente, o TSVM (*Transductive Support Vector Machines*) que é a versão transdutiva do famoso algoritmo supervisionado SVM (*Support Vector Machines*). O SVM busca um hiperplano que divide os exemplos de cada classe, maximizando a distância mínima de cada grupo de exemplos ao hiperplano. Na versão transdutiva temos a adição da distribuição $P(x)$ para melhorar a hipótese do classificador, pressupondo que existe uma região de baixa densidade que separa os exemplos de cada classe. Porém, encontrar a solução exata do TSVM é *NP-difícil*. Os algoritmos exatos só conseguem tratar poucos exemplos não rotulados (Bennett e Demiriz, 1998; Demiriz e Bennett, 2001; Fung e Mangasarian, 2001). Por isso um grande esforço tem sido despendido em algoritmos de aproximação (Yajima e Hoshiba, 2005; Joachims, 1999b).

O interesse em aprendizado semissupervisionado cresceu na década de 1990 devido às suas aplicações em problemas de linguagem natural e classificação de texto (Yarowsky, 1995; Blum e Mitchell, 1998; Braga, 2010). Yarowsky (1995) utilizou o autoaprendizado para desambiguar várias palavras da língua inglesa como *"plant"*, que dependendo do contexto pode ter o sentido de vegetal ou uma instalação industrial. Maeireizo et al. (2004) utilizou o algoritmo Co-training para classificar diálogos como "emotivos" ou "não-emotivos". Rosenberg et al. (2005) demonstra como o aprendizado semissupervisionado pode rivalizar com o estado da arte na detecção de objetos em imagens. Merz et al. (1992) foram os primeiros a usar o termo "semissupervisionado" para a classificação com os dados rotulados e não rotulados no contexto do aprendizado de máquina. Com a expansão da internet existe uma quantidade cada vez maior de dados e devido a impossibilidade de rotular todo esse volume de dados, o aprendizado semissupervisionado vem se popularizando cada vez mais. Sua capacidade de tratar ao mesmo tempo dados rotulados e não rotulados o torna uma ótima escolha para tratar o problema valores ausentes.

1.2 Objetivos

Apesar de o aprendizado semissupervisionado ser um campo em expansão, ainda existem poucos trabalhos de imputação utilizando o aprendizado semissupervisionado. O presente trabalho tem como objetivo utilizar algoritmos e ferramentas disponíveis livremente para verificar a possibilidade do aprendizado semissupervisionado ser utilizado para tarefas de imputação de dados.

Os exemplos com valores ausentes ao serem imputados são separados em um conjunto de treino com os exemplos completos e outro conjunto denominado conjunto de predição com os

exemplos com valores ausentes. O conjunto de predição pode ser utilizado como um conjunto de dados não rotulados para um algoritmo de aprendizado semissupervisionado, fornecendo assim mais informação para o algoritmo do que seria possível para um algoritmo supervisionado e permitindo assim obter mais informações durante o treinamento o que pode possibilitar em uma melhora na qualidade da hipótese induzida.

Com essas idéias em mente para utilização de semissupervisionado para imputação, diversas são as perguntas interessantes sobre o assunto, entre as quais vale destacar:

- Os dados não rotulados podem ajudar a melhorar a qualidade da imputação?
- Qual algoritmo semissupervisionado seria o melhor para tarefas de imputação?
- Quando usar ou não usar o aprendizado semissupervisionada para imputação?

Para facilitar o direcionamento dos esforços de pesquisa, este trabalho propõe fazer um estudo sobre o assunto na tentativa de responder totalmente ou parcialmente a esses questionamentos e outros que porventura podem aparecer no decorrer do desenvolvimento deste trabalho de mestrado.

1.3 Organização

Este trabalho está organizado da seguinte forma: no Capítulo 2 é fundamentada e apresentada uma visão geral sobre Aprendizado de Máquina e a Descoberta de Conhecimento em Banco de Dados e a Mineração de Dados. No Capítulo 3 são aprofundados os conceitos relativos aos Valores Ausentes, métodos de imputação e algoritmos. Após isso, no Capítulo 4 são apresentados os experimentos realizados juntamente com a metodologia adotada e os resultados. No Capítulo 5 são apresentadas as conclusões finais.

CAPÍTULO

2

Conceitos Introdutórios de Aprendizado de Máquina

Neste capítulo são apresentados conceitos introdutórios necessários para compreensão do tema. O capítulo está organizado da seguinte forma: na Seção 2.1 é introduzido o conceito de Aprendizado de Máquina através de analogias com o aprendizado humano, também são abordados os tipos de aprendizado e a notação usada. A Descoberta de Conhecimento em Banco de Dados e a Mineração de Dados são abordadas na Seção 2.2. Os conceitos e definições, apresentados neste apítulo são importantes para a compreensão do assunto. Alguns trechos foram baseados no texto de Zhu e Goldberg (2009).

2.1 Aprendizado de Máquina

A inteligência é a chave do sucesso da espécie humana. Graças a ela o homem criou inúmeros artefatos que garantiram a sua supremacia sobre outras espécies, mesmo não sendo o de maior estatura, o mais rápido ou o mais forte entre os animais. Esta capacidade há milhares de anos intriga os estudiosos, que procuram entender como o ser humano pensa e aprende e quais são as leis que governam o processo de aprendizado. A Inteligência Artificial (IA) é uma área de pesquisa em ciência da computação que não só tenta compreender, mas também construir entidades inteligentes (Russell e Norvig, 2003).

Desde sua criação, na década de 50, a IA tem evoluído muito, mas ainda é necessário muito desenvolvimento para conseguir criar máquinas que pensem ou ajam como seres humanos de forma geral. O grande desafio desta tarefa está na compreensão de como o ser humano aprende e como melhora seu desempenho cada vez que realiza uma mesma tarefa. Por exemplo, um professor de biologia perguntou a seus alunos quantas horas eles estudaram antes da realização

de uma de suas provas e obteve os resultados mostrados na Tabela 2.1. [1]A primeira coluna representa a identificação do aluno, na segunda o número de horas de estudo para a prova e na terceira a nota da prova. De modo geral os que dedicam mais tempo ao estudo tiram notas melhores.

Tabela 2.1: Identificação única do aluno, horas de estudo e nota dos alunos

ID	horas de estudo	nota
2010011	6,0	9,2
2010035	9,0	7,5
2010052	5,0	6,7
2010079	6,0	6,2
2010075	10,0	6,1
2010022	10,0	5,7
2010041	3,0	5,2
2010060	4,5	4,0
2009019	4,0	3,2

Para explicar esse fato considere a definição de aprendizado de Simon (1983):

> *Aprendizado denota mudanças no sistema, que são adaptáveis no sentido de que elas possibilitam que o sistema realize a mesma tarefa sobre uma mesma população, de maneira mais eficiente a cada vez.*

Esta definição é genérica e abrange um amplo leque de fenômenos. As mudanças no sistema são os refinamentos de habilidades que advém com a repetição das mesmas tarefas sobre a mesma população, que neste caso é exemplificado pela matéria de estudo dos alunos, a biologia. Isso explica uma melhor nota para os alunos que dedicam mais tempo em seus estudos, pois como praticam mais, executam as tarefas de maneira mais eficiente e com isso melhoram cada vez mais seu desempenho. Entretanto, a simples repetição não explica melhores resultados pois dois alunos podem empregar o mesmo número de horas todos os dias e obterem resultados diferentes. Por exemplo na Tabela 2.1 os alunos com ID 2010011 e 2010079 estudaram o mesmo número de horas e tiraram notas diferentes. Usando a definição de aprender de Ryszard S. Michalski (1986):

> *Aprender é construir ou modificar representações sobre o que está sendo experienciado.*

Para modificar ou construir representações é necessário adquiri-las através de experiências, e a experiência pode ser adquirida de várias formas, duas delas são: por meio de estímulos sensoriais ou através de processos internos de pensamento. O aprendiz utiliza os estímulos sensoriais

[1]Os exemplos e tabelas envolvendo, professor e alunos de biologia utilizam dados reais, obtidos através de entrevista com alunos, a disciplina foi alterada para manter a anonimidade.

para perceber a realidade que o cerca e a partir disso construir ou modificar suas percepções a respeito desta realidade internamente. O processo de pensamento pode ser considerado experiência, pois permite a criação e a modificação das representações. Ainda segundo Ryszard S. Michalski (1986), quando o conhecimento é bem representado, tem-se o aprendizado, então o fato de dois alunos estudarem a mesma quantidade de horas e terem desempenhos diferentes se deve a qualidade de suas representações, e melhores representações, geralmente, resultam em um melhor aprendizado. O grande desafio dos pesquisadores de IA tem sido utilizar estes avanços na compreensão do aprendizado humano, para tentar fazer as máquinas aprenderem e com isso ampliar ainda mais o leque de tarefas que elas podem executar.

Assim surgiu o Aprendizado de Máquina, que no início estudava como compreender o processo de aprendizado humano para que se reproduzisse em máquinas. Da mesma forma a IA tinha como objetivo criar máquinas que pensassem ou agissem como seres humanos (Turing, 1950), o AM buscava entre outras coisas, compreender o aprendizado humano, para que ele pudesse ser reproduzido em máquinas. Atualmente, foram desenvolvidas abordagens que buscam realizar o aprendizado de maneira pragmática sem a preocupação de reproduzir o aprendizado humano. Assim, o AM pode ser definido comoMitchell (1997):

> *É dito que um programa aprende a partir da experiência E, em relação a uma classe de tarefas T, com medida de desempenho P, se seu desempenho em T, medido por P, melhora com E.*

Como pode ser visto, essa definição é muito semelhante à de Simon (1983), ou seja, as máquinas também podem aprender com a experiência. Além disso, os algoritmos de aprendizado recebem como entrada exemplos de treino. Estes exemplos podem ser vistos como os estímulos sensoriais dos humanos, cada exemplo é um conjunto de características individuais daquilo que se deseja aprender. O conceito que se deseja aprender a partir desses exemplos é chamado de classe. Fazendo analogia com o aprendizado humano, quando um aluno de biologia está aprendendo a classificar plantas, ele busca por vários exemplares de plantas que podem ser vistos como os exemplos de treino dos algoritmos de AM. A cada exemplar mostrado, os seus sentidos podem distinguir as características como cor, cheiro e forma de cada planta. Com a experiência adquirida ele pode classificar melhor os novos exemplares que são mostrados, separando cada planta por sua espécie que é o conceito que se deseja aprender, ou seja, a classe.

Outro fator que diferencia os estudantes é o tipo de inferência que realizam sobre os dados disponíveis, definindo assim várias estratégias de aprendizado que podem ser um bom critério de classificação dos processos de aprendizado de máquina. As estratégias básicas em ordem crescente de complexidade são as seguintes: hábito, instrução, dedução, analogia e indução (Monard et al., 1997). Neste trabalho o foco é no aprendizado indutivo que é descrito a seguir.

2.1.1 Aprendizado de Máquina Indutivo

Apesar do aprendizado indutivo ser o de maior complexidade, ou seja, o que o aprendiz gasta mais tempo para aprender (Monard et al., 1997), ele é um dos mais úteis pois permite obter

novos conhecimentos a partir de um conjunto de exemplos, ou casos observados previamente. Permite obter conclusões genéricas com exemplos particulares, por exemplo a seguinte indução:

Todas as amostras de mamíferos tinham ossos,
portanto todos os mamíferos tem ossos.

Neste exemplo, a partir de amostras foi feita a generalização de que todos os mamíferos tem ossos. Esta generalização pode ou não estar correta, pois parte de um caso específico para o todo e devido a isso, o novo conhecimento geralmente é chamado de hipótese, pois não existem garantias que de que é válido. Mesmo assim, a inferência indutiva é um dos mecanismos mais utilizados por algoritmos de AM para a aquisição de novos conhecimentos e previsão de eventos futuros. Como a veracidade das premissas não garante a veracidade da conclusão, mas apenas que ela provavelmente deva ocorrer, sempre existe um erro associado à conclusão.

O aprendizado indutivo por exemplos é um tipo de aprendizado indutivo que induz descrições gerais de conceitos utilizando-se de exemplos específicos desses conceitos (Ryszard S. Michalski, 1986). Por exemplo, o aluno que estuda para a prova resolvendo uma lista de exercícios similares aos que existirão na prova, extrai conhecimento desses exemplos e quando encontra exercícios similares na prova tem maior chance de resolvê-los. Vários algoritmos de AM utilizam aprendizado indutivo por meio de exemplos, neste caso, estes algoritmos também são chamados de indutores e realizam generalizações e especializações na indução de uma hipótese h. Esses sistemas tomam decisões baseadas em experiências acumuladas contidas em outros casos resolvidos com sucesso (Weiss e Kulikowski, 1991). Basicamente, o aprendizado de máquina indutivo pode ser dividido em aprendizado supervisionado, semissupervisionado e aprendizado não supervisionado como mostrado na Figura 2.1.

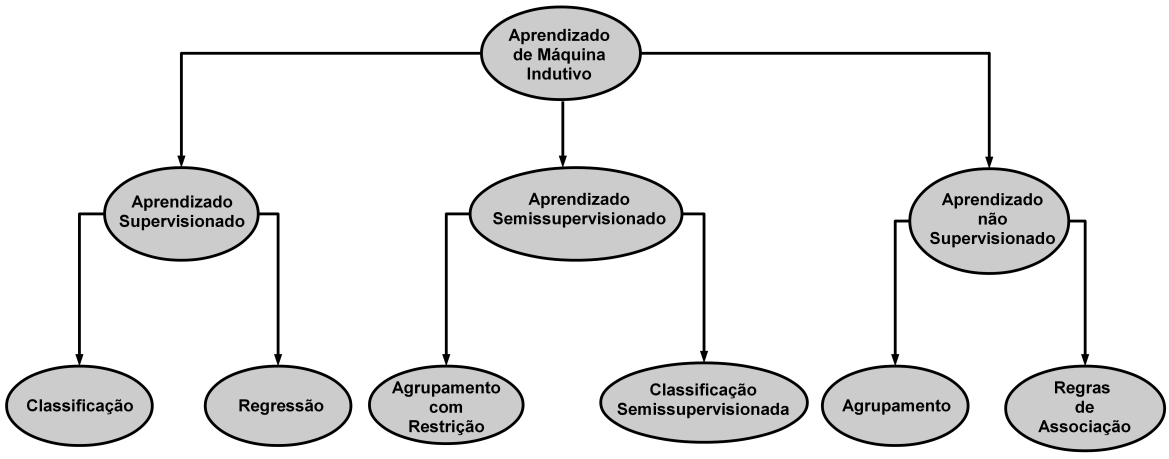


Figura 2.1: Tipos de aprendizado

No aprendizado supervisionado, a indução dos conceitos é realizada a partir de exemplos pré-classificados, ou seja, os exemplos já estão rotulados com uma classe ou conceito conhecido. Se as classes possuem valores contínuos, por exemplo se for considerado a nota dos alunos

que pode assumir qualquer valor no intervalo de 0 a 10, a tarefa é conhecida como regressão. Caso esses valores sejam discretos, como conceitos A, B, C ou D, a tarefa é chamada de classificação. Já no aprendizado não supervisionado as classes são desconhecidas. Nesse caso, as tarefas mais comuns estão relacionadas ao agrupamento dos exemplos por meio de alguma métrica de similaridade, e encontrar associações interessantes em grandes conjuntos de dados usando regras de associação. O aprendizado semissupervisionado é uma mistura do aprendizado supervisionado com o não supervisionado e consegue induzir conceitos tanto de exemplos rotulados como de não rotulados.

No entanto, em qualquer uma dessas tarefas é necessário definir uma linguagem de descrição a ser utilizada para descrever os exemplos. Dentre as linguagens de descrição disponíveis para representar os exemplos foi escolhida para este trabalho a linguagem proposicional, por ser utilizada por muitos algoritmos de AM. Nessa linguagem os exemplos ou instâncias são descritos utilizando um conjunto de atributos. Um exemplo é representado por um vetor de características n_{atr}-dimensional $x = (a_1, a_2, \ldots, a_{n_{atr}}) \in \mathbb{R}^{n_{atr}}$, ou simplesmente como um vetor $\vec{x}$, onde cada dimensão é chamada de atributo. O comprimento n_{atr} do vetor de atributos é chamado de dimensão do vetor. Cada exemplo representa uma linha (tupla) em uma tabela no formato atributo valor.

Tabela 2.2: Conjunto de dados no formato atributo valor

	A_1	A_2	$\ldots$	$A_{n_{atr}}$	Y
x_1	a_{11}	a_{12}	$\ldots$	$a_{1n_{atr}}$	y_1
x_2	a_{21}	a_{22}	$\ldots$	$a_{2n_{atr}}$	y_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$x_{n_{ex}}$	$a_{n_{ex}1}$	$a_{n_{ex}2}$	$\ldots$	$a_{n_{ex}n_{atr}}$	$y_{n_{ex}}$

A Tabela 2.2 ilustra o formato geral de uma tabela atributo valor com x_i exemplos com $i = 1, \ldots, n_{ex}$ e atributos A_j com $j = 1, \ldots, n_{atr}$, sendo que cada exemplo é tipicamente um vetor na forma $(\vec{x}_i, y_i)$. Os valores y_i que podem ou não existir, referem-se ao valor do atributo de classe Y, que é um atributo especial cujo valor é a classe ou rótulo do exemplo. Em tarefas de classificação, os valores do atributo de classe pertencem a um conjunto Y de classes discretas y_c onde $c = 1, \ldots n_{classes}$. Quando o atributo classe existe, ele é obtido através de observações passadas ou através de um especialista do domínio. Neste caso o conjunto é denominado conjunto rotulado. No caso em que o atributo classe não existe, o conjunto é denominado conjunto não rotulado. Note que como existem múltiplos exemplos, será usado x_{id} para denotar a i-ésima instância do d-ésimo atributo. Dado um conjunto de exemplos, é importante distinguir os quatro tipos de conjuntos comumente utilizados na maioria dos algoritmos de AM:

Conjunto de treinamento ou treino: Esse conjunto é fornecido como entrada dos algoritmos de AM, também é conhecido como casos vistos pois refere-se aos exemplos "vistos" pelo algoritmo de AM. A partir dele são construídos os modelos ou hipóteses.

Conjunto de teste: É o conjunto utilizado para avaliar o modelo. Também é conhecido como casos não vistos. Esse conjunto não deve ser "visto" pelo algoritmo na fase de indução do modelo e idealmente os exemplos desse conjunto devem ser disjuntos do conjunto de treino.

Conjunto de validação: Em alguns casos é necessário realizar ajustes no modelo, para isso é necessário um conjunto de exemplos que não foram utilizados na construção do modelo. Como os exemplos de teste seriam indiretamente "vistos", é necessário um novo conjunto com exemplos distintos do conjunto de teste.

Conjunto de predição: Conjunto de exemplos não rotulados que devem ter seu rotulo predito pelo modelo induzido. O indutor pode "ver" os exemplos desse conjunto para melhorar o modelo induzido, é utilizado no aprendizado semissupervisionado ou transdutivo.

Nas próximas Subseções serão detalhados cada um dos tipos de aprendizado, com o intuito de facilitar a leitura dos capitulos subsequentes.

2.1.2 Aprendizado Supervisionado

Como já foi visto, no aprendizado supervisionado existe um rótulo $y_i \in Y$ que é a predição desejada para um exemplo $x_i \in X$ na amostra de teste. Os rótulos podem vir de um conjunto de valores finitos. Esses valores distintos entre si, são denominados classes. Retornando ao exemplo da Tabela 2.1, o atributo de classe poderia ser aprovado ou reprovado, sendo aprovado = 1 e reprovado = -1. Essa codificação em particular é usada para rótulos binários, e as duas classes são chamadas genericamente de positiva e negativa, respectivamente. Para codificações de mais de uma classe existem várias técnicas que podem ser utilizadas dependendo do algoritmo de aprendizado utilizado. Uma comumente utilizada é a codificação $y_i = 1, 2, ..., n_{classes} \in \mathbb{N}$, onde $n_{classes}$ é o número de classes. Por exemplo, pode-se ter as seguintes classes que representam as disciplinas cursadas por alunos: biologia =1, cálculo = 2 e inteligência artificial = 3. Note que a classe com rótulo 1 não está necessariamente mais próxima da classe com rótulo 2 ou mais distante da classe 3, ou seja, não existe qualquer noção de ordem entre os valores, e os rótulos são atribuídos normalmente de maneira arbitrária.

Com esses conceitos pode-se definir o aprendizado supervisionado (Zhu e Goldberg, 2009) como sendo o domínio dos exemplos X e o domínio dos rótulos Y, tendo $P(x,y)$ como uma junção de distribuições desconhecidas dos exemplos e rótulos $X \times Y$. Dada uma amostra de treinamento, o aprendizado supervisionado induz uma função $h : X \longrightarrow Y$ que se aproxima da verdadeira função desconhecida $f : X \longrightarrow Y$, com o objetivo de $h(x)$ predizer o verdadeiro rótulo y_i em um exemplo futuro x_i. Em outras palavras, o algoritmo de aprendizado supervisionado constrói um modelo que mapeia o exemplo x_i a um rótulo de classe y_i. Dependendo do domínio do rótulo y_i, os problemas de aprendizado supervisionado podem ser divididos em classificação e regressão:

Classificação: a classificação é caracterizada pelo tipo de valor do atributo classe Y que é discreto. Por exemplo se a Tabela 2.1 for modificada de forma que o atributo nota seja transformado em um conceito, resultando na Tabela 2.3. Após isso o algoritmo irá induzir uma função $h : X \longrightarrow Y$, onde $y \in \{A, B, C, D\}$, com o objetivo de predizer o verdadeiro rótulo do exemplo.

Regressão: Constituem os problemas de aprendizado supervisionado onde o conjunto de valores do atributo de classe Y é contínuo $Y \in \mathbb{R}$. Por exemplo no caso dos alunos de biologia da Tabela 2.1, as notas podem ser vistas como atributos contínuos, se for considerado que podem ter qualquer valor na faixa de 0 a 10.

Tabela 2.3: Identificação única do aluno, horas diárias de estudo e média final dos alunos

ID	horas de estudo	nota
2010011	6,0	A
2010035	9,0	B
2010052	5,0	B
2010079	6,0	B
2010075	10,0	B
2010022	10,0	B
2010041	3,0	B
2010060	4,5	C
2009019	4,0	C

Pode ser visto que as tarefas de regressão e classificação tem como objetivo predizer o valor da classe correspondente para exemplos previamente não vistos. Assim, as tarefas de classificação e regressão consistem na generalização dos exemplos que tenham uma classe conhecida, em uma linguagem capaz de reconhecer a classe de um exemplo com rótulo desconhecido. Como já mencionado, o conjunto de dados é dividido em dois conjuntos disjuntos: o conjunto de treino que é usado para induzir a hipótese, e um conjunto de teste que é usado para validar a hipótese, verificando se ela é suficientemente genérica, mostrando que o algoritmo "aprendeu" com os exemplos. A generalização do conceito aprendido pode sofrer os seguintes problemas:

Super ajuste: Do inglês *overfitting.* É o caso onde a hipótese induzida é altamente especializada para o conjunto de treino, atingindo uma alta taxa de acertos neste conjunto, mas uma baixa taxa de acertos no conjunto de teste, demonstrando que na verdade o algoritmo "decorou" os exemplos e não conseguiu generalizar o conceito. Por exemplo, em uma rede neural, se o número de neurônios for muito grande, a função induzida pode incorporar o ruído dos dados, perdendo sua capacidade de generalização. Essa situação é conhecida como super ajuste.

Sub ajuste: Do inglês *underfitting*. É a situação oposta. Quando o algoritmo de aprendizado não consegue abstrair o conhecimento do conjunto de treino, isso pode ocorrer devido a um subdimensionamento de parâmetros do algoritmo. Por exemplo, em uma rede neural, se o número de neurônios não for suficientemente grande, a função induzida pode não ser genérica o suficiente para representar o conhecimento, gerando uma baixa taxa de acerto tanto no conjunto de teste quanto no conjunto de treino. Essa situação é chamada de sub ajuste.

Além disso, as bases de dados podem ser volumosas, contendo milhares de exemplos, o que torna difícil um algoritmo ter um bom desempenho em diversos cenários. Por esse motivo, uma grande variedade de algoritmos assume pressupostos sobre a distribuição dos dados, na tentativa de obter um bom desempenho. Os algoritmos de aprendizado supervisionado podem ser divididos pela forma que fazem esses pressupostos sobre a distribuição dos dados. Segundo Chapelle et al. (2006) eles se dividem em duas grandes famílias de modelos de algoritmos:

Gerativos: Tentam modelar $p(x|y)$, ou seja, a distribuição condicional de x dado y.

Discriminativos ou preditivos: Algoritmos que tentam aprender diretamente a partir dos exemplos um mapeamento $h : X \longrightarrow Y$.

Por exemplo, considere o seguinte problema de classificação: um aluno de biologia tentando classificar flores do tipo íris entre setosa ou virgínica. Ele pode analisar as flores e criar modelos de distribuição com parâmetros diferentes para cada um dos tipos de íris. Supõe-se que estes modelos geram os exemplos de cada classe portanto, o modelo é denominado gerativo. Quando for classificar um novo exemplar de flor o aluno utiliza estes modelos para calcular as probabilidades de a flor pertencer a cada uma das classes e realizar a classificação; esta é a abordagem gerativa do problema. Entretanto, se este aluno representar através de um gráfico os dados das flores e encontrar uma função que divida os dois tipos de íris permitindo que ele as classifique diretamente, sua abordagem é a discriminativa. Desta forma não existe um suposto modelo para gerar os dados, mas apenas uma função para discriminá-los.

Tanto para o modelo gerativo quanto para o discriminativo, assume-se que se tem dados rotulados suficiente para a indução de um bom classificador. Liang e Jordan (2008) sugerem o uso de algoritmos gerativos quando o conjunto de treino é limitado. Nos casos em que não se tem dados rotulados disponíveis suficiente, sua obtenção é muito custosa ou trabalhosa e pode requerer o auxílio de especialistas no domínio. Uma solução é a utilização do aprendizado semissupervisionado que reduz a necessidade de dados rotulados e será visto em detalhes mais a frente.

2.1.3 Aprendizado Não Supervisionado

No aprendizado não supervisionado não existe a supervisão de um "professor" na manipulação dos exemplos e essa é a propriedade que define o aprendizado não supervisionado (Zhu e Goldberg, 2009). A entrada do algoritmo de aprendizado não supervisionado é um conjunto

de dados $X = (x_1, x_2, \ldots, x_{n_{ex}})$, onde cada exemplo consiste somente do vetor $\vec{x}$, sem o atributo de classe Y. A tarefa mais frequente no aprendizado não supervisionado tem como objetivo encontrar alguma regularidade no conjunto de dados, formando agrupamentos de exemplos que tem alguma similaridade, ou seja, os exemplos são descritos em agrupamentos (*clusters*) de acordo com alguma medida de similaridade. Por exemplo: considere um grupo de pessoas que poderiam ser agrupadas por sexo gerando dois agrupamentos, um para homens e outro para mulheres. Diferentemente de tarefas preditivas como a classificação; as tarefas descritivas consistem na identificação de propriedades intrínsecas do conjunto de dados que nem sempre possuem uma classe especificada. As tarefas mais comuns do aprendizado não supervisionado incluem:

Agrupamento: o objetivo é agrupar os exemplos usando alguma medida de similaridade;

Detecção de novidade: identifica os poucos exemplos que se diferem da maioria;

Redução da dimensionalidade: cujo objetivo é reduzir a dimensão do vetor de características enquanto mantém características essenciais da amostra de treino;

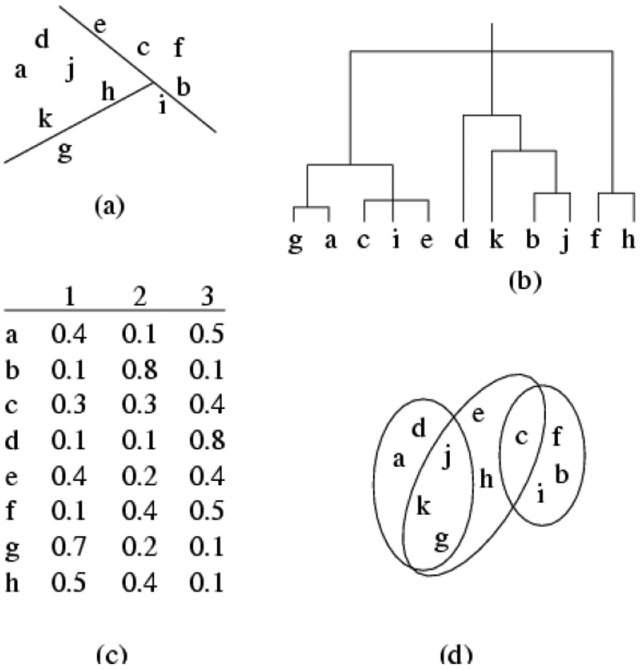
Regras de associação: cujo objetivo é encontrar associações interessantes, como relacionamentos e dependências em grandes conjuntos de dados.

Entre essas tarefas, o agrupamento é o mais relevante para este trabalho, por isso será explanado em mais detalhes. Existem algumas questões importantes que devem ser levadas em consideração quando se realiza agrupamento:

- Qual a melhor forma de normalizar os dados?
- Qual medida de similaridade usar para a situação em questão?
- Como deve ser utilizado o conhecimento do domínio?
- Como agrupar um grande conjunto de dados de forma eficiente? e
- Como avaliar os agrupamentos encontrados?

A seleção do algoritmo de AM adequado a um determinado problema deve ser criteriosa. Pode-se utilizar um conjunto de critérios de admissibilidade para comparar os diferentes algoritmos de agrupamento. Esses critérios são baseados: em como os agrupamentos são formados, na estrutura dos dados e na "sensibilidade" da técnica de agrupamento a mudanças que não afetem a estrutura dos dados (Fisher e Ness, 1971).

Não existe uma técnica de agrupamento universal, ou seja, deve-se avaliar entre as técnicas existentes, qual se adapta melhor ao problema, domínio e aos dados. As diferentes técnicas de agrupamento são classificadas de acordo com o tipo de resultado obtido pelo algoritmo. Na Figura 2.2 são ilustradas as técnicas de agrupamento: por otimização, hierárquica, probabilística e *clumping*.



	1	2	3
a	0.4	0.1	0.5
b	0.1	0.8	0.1
c	0.3	0.3	0.4
d	0.1	0.1	0.8
e	0.4	0.2	0.4
f	0.1	0.4	0.5
g	0.7	0.2	0.1
h	0.5	0.4	0.1

Figura 2.2: Técnicas de agrupamento: (a) Otimização (b)Hierárquica (c) Probabilística (d) Clumping. (Martins, 2003) com adaptações

Otimização: Visa encontrar uma k-partição ótima. Dado um valor k, o conjunto de dados é dividido em k agrupamentos, mutuamente exclusivos. As técnicas de otimização costumam ser caras, pois normalmente fazem buscas exaustivas pela partição k ótima. Devido a isso, seu uso é restrito a pequenos valores de k.

Hierárquica: Cria árvore onde as folhas representam os exemplos e os nós internos representam os grupos de exemplos. Pode ser divididas pela forma como constroem a árvore em aglomerativos e divisivos. Os aglomerativos constroem a árvore de baixo para cima, ou seja, da folha para a raiz; os divisivos o fazem de forma inversa.

Probabilística: Esta técnica associa a cada exemplo a probabilidade dele pertencer ou não a um agrupamento.

***Clumping*:** Estas técnicas retornam grupos onde os exemplos podem se sobrepor, ou seja, podem pertencer a um ou mais grupos. Por exemplo, embora as técnicas probabilísticas retornem a probabilidade de um exemplo pertencer a um grupo, cada exemplo pertence a apenas um único grupo. Entretanto, a técnica probabilística pode ser vista como uma técnica de clumping, se a probabilidade for usada para que um mesmo exemplo pertença a vários grupos.

Nem sempre os agrupamentos encontrados estarão ligados a uma representação conceitual relacionada a uma classe. Devido a isso, dois ou mais agrupamentos podem conter exemplos de uma mesma classe. Por exemplo, na Figura 2.3.a, o algoritmo encontrou quatro agrupamentos, para um determinado conjunto de dados, e na Figura 2.3.b pode-se verificar a existência de

duas classes predominantes nos agrupamentos. Neste exemplo existem 4 *clusters* e 2 classes, apresentado uma ilustração das suas diferenças conceituais.

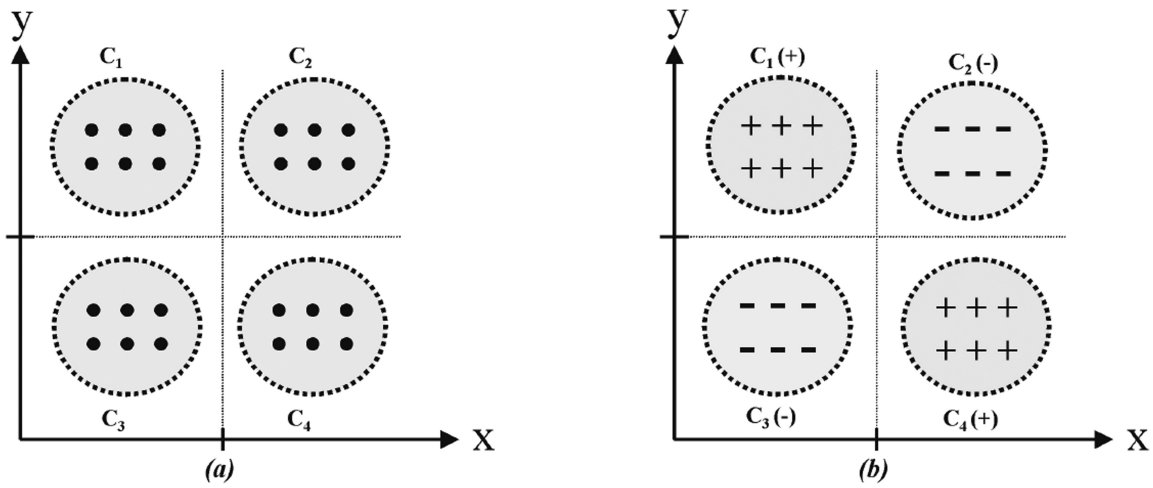


Figura 2.3: Diferença entre agrupamento e classe (Martins, 2003)

A seguir, de posse de todos esses conceitos, pode-se compreender o aprendizado semissupervisionado, que se situa entre o supervisionado e o não supervisionado e é mostrado em detalhes na próxima Seção.

2.1.4 Aprendizado Semissupervisionado

Como o nome sugere, o aprendizado semissupervisionado é o meio termo entre o aprendizado supervisionado e o não supervisionado. O aprendizado supervisionado generaliza padrões nos dados rotulados de treinamento, e o aprendizado não supervisionado identifica propriedades intrínsecas do conjunto de dados não rotulados, o aprendizado semissupervisionado aproveita a informação contida tanto nos dados rotulados quanto nos não rotulados. Tem surgido como uma alternativa quando existe uma grande quantidade de dados não rotulados disponíveis e poucos dados rotulados. Isso pode ocorrer devido ao custo ou à dificuldade de se rotular os dados. Muitos dos algoritmos de AM semissupervisionados são extensões de algoritmos supervisionados ou não supervisionados, incluindo informação adicional do outro paradigma (Zhu e Goldberg, 2009). Geralmente, as duas estratégias mais estudadas no aprendizado semissupervisionado são:

Agrupamento com restrição: Que está relacionado com o agrupamento do aprendizado não supervisionado. Além da informação dos exemplos não rotulados, existe alguma "informação supervisionada" sobre os agrupamentos fornecida por um especialista no domínio. Essa informação pode, por exemplo, serem restrições da forma *must-link* ou *cannot-link* entre exemplos da base de dados. Enquanto o *must-link* indica que os exemplos pertencem a um mesmo agrupamento, o *cannot-link* indica o contrário.

Classificação semissupervisionada: Também conhecida como classificação com dados rotulados e não rotulados, é uma extensão da classificação supervisionada. Além dos exemplos rotulados, utilizam-se os exemplos não rotulados. Seu uso é indicado quando o número de exemplos não rotulados é maior que o de exemplos rotulados. Porém nada impede sua utilização em outros cenários.

Em uma pesquisa realizada por Cozman e Cohen (2006), sobre a utilização de dados não rotulados para melhorar o desempenho de classificadores foi demonstrado que o uso de dados não rotulados nem sempre aumenta o desempenho do classificador. A degradação, segundo os autores, ocorre devido ao uso de modelos estatísticos incorretos para definir o comportamento dos dados não rotulados. Eles sugerem o uso de aprendizado supervisionado quando existirem dados rotulados suficientes. Somente nos casos onde o desempenho do classificador supervisionado deixa a desejar, um classificador semissupervisionado é recomendado. Outras pesquisas afirmam (Singh et al., 2008) que existem muitos casos onde a classificação semissupervisionada pode oferecer melhor desempenho que a classificação supervisionada, desde que os seguintes pressupostos sejam cumpridos:

Pressuposto de suavidade: Define que os rótulos dos exemplos devem variar de forma suave da área de alta densidade para a de baixa densidade.

Pressuposto de agrupamentos: Assume-se que os exemplos pertencentes a um mesmo agrupamento pertencem à mesma classe. Note que isso é apenas uma tendência, pois podem existir exemplos que não pertençam a classe, mas em menor proporção. Na verdade este pressuposto é uma extensão do pressuposto de suavidade, com a diferença que neste pressuposto a separação em agrupamentos implica na existência de áreas de baixa densidade separando as áreas de alta densidade em agrupamentos.

Pressuposto de variedades: do inglês *"manifold"*, apesar dos dados terem centenas de atributos, alguns conjuntos de dados podem ser representados em um espaço de menor dimensão, pois sua dimensionalidade é artificialmente alta. Um exemplo é ilustrado na Figura 2.4, onde os dados estão em um espaço tridimensional formando um rolo suíço. Nesta forma, dependendo de como os dados são observados, as classes parecerão misturadas, mas se estes mesmos dados forem mapeados para um *isomap* (Tenenbaum et al., 2000) de duas dimensões a separação entre as classes é linear.

Um outro conceito importante que tem uma forte conexão com o aprendizado semissupervisionado é o **Aprendizado transdutivo**. A transdução foi introduzida por (Vapnik e Chervonenkis, 1974) em contraste com a inferência indutiva, que tem como objetivo induzir hipóteses que sejam válidas para exemplos fora do conjunto de dados não rotulados. Na inferência transdutiva nenhuma regra de decisão geral é inferida, apenas os rótulos dos dados de teste são preditos (Chapelle et al., 2006), sem generalizar um modelo para os dados desconhecidos, ou seja, no aprendizado transdutivo todos os exemplos de teste precisam ser previamente conhecidos. Para facilitar a compreensão desse tipo de aprendizado, considere a seguinte analogia: um detetive

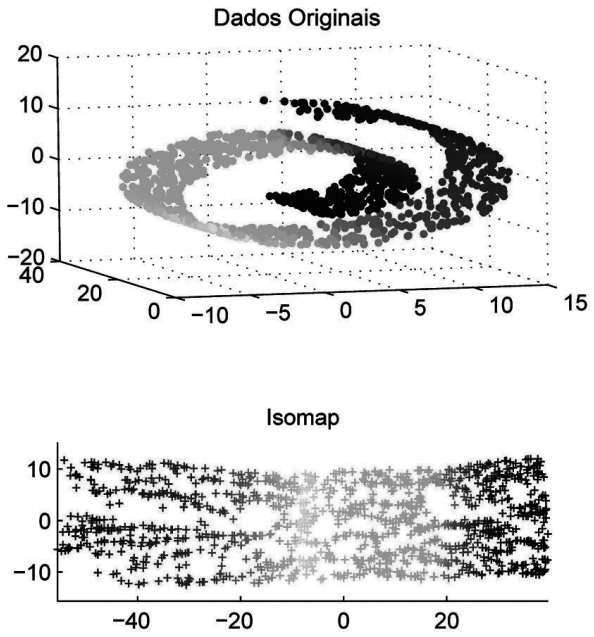


Figura 2.4: Rolo suíço (acima) e sua representação bidimensional (abaixo) (Cayton, 2005) com adaptações

tentando solucionar um crime de posse de uma lista com os nomes de todos os suspeitos, cuja tarefa é descobrir quem nesta lista é o criminoso. Considere essa situação onde o detetive não necessita procurar novos suspeitos, pois sabe-se que o criminoso é alguém da lista. Isto é equivalente a classificar todos da lista de suspeitos em criminoso ou inocente, sem a necessidade de classificar alguém fora dessa lista. Desta maneira, o detetive trabalha como um classificador transdutivo investigando cada pessoa da lista, ignorando a possibilidade de pessoas fora dessa lista serem futuramente classificadas em criminosos ou inocentes. Sua metodologia de investigação não consegue lidar com novos suspeitos, apenas os que ele investigou previamente. Desse modo, na transdução é necessário obter a lista de suspeitos previamente. Em contraste com esse problema, a classificação de mercadorias de um fiscal de alfândegas, dificilmente poderia ser considerado um problema transdutivo, dado que dificilmente um fiscal terá a lista completa de mercadorias, durante o treinamento. O conceito da Transdução e os pressupostos servem como base para vários algoritmos de aprendizado semissupervisionado (Chapelle et al., 2006). A Transdução será vista em detalhes no Capítulo 3.

O aprendizado semissupervisionado é muito útil quando se tem mais dados não rotulados do que rotulados. Pois dados não rotulados são baratos de se obter, ao contrário dos rotulados que costumam ser custosos. Com isso áreas como bioinformática, processamento de páginas web e reconhecimento de fala podem se beneficiar dos dados não rotulados com o uso do aprendizado semissupervisionado. Na próxima seção é abordada a descoberta de conhecimentos em banco de dados uma área que constantemente lida com valores ausentes. Alguns trechos foram baseados no texto de Batista (2003).

2.2 Descoberta de Conhecimento em Banco de Dados

Nos últimos três séculos, o mundo passou por grandes mudanças tecnológicas. O século XVIII foi a era dos sistemas mecânicos e da revolução industrial. Já o século XIX foi a era da máquina a vapor, e no século XX a chamada era da informação. Devido à redução de custo e aumento de capacidade dos dispositivos de armazenamento, os bancos de dados vêm crescendo em tamanho ano após ano. Em 2008, empresas como o Yahoo, já estavam com banco de dados na casa de 2 *petabytes*. Atualmente, projetos científicos como o acelerador de partículas Large Hadron Collider (LHC) vão ainda mais longe. Estima-se que coletará anualmente 15 petabytes de dados. Com toda essa enorme massa bruta de dados, a maior dificuldade reside na análise desses dados. Por exemplo: Kepler gastou quatro anos para analisar os dados que Tycho Brahe coletou durante 20 anos. Estes dados consistiam da posição de 777 estrelas e os planetas conhecidos na época, o que hoje seria equivalente a uns poucos kilobytes. Com esse exemplo pode-se ver que bases de dados como a do LHC não podem ser analisadas sem auxílio de ferramentas automatizadas ou semiautomatizadas, pois seriam necessários séculos para se realizar a análise manual dos dados. Devido a essa necessidade, surgiu uma nova área de pesquisa chamada descoberta de conhecimento em banco de dados (*knowledge-discovery in databases* - KDD). Existem várias definições para KDD, uma das mais utilizadas é a de Fayyad et al. (1996):

> *A Descoberta de conhecimento em banco de dados é um processo não trivial para identificar padrões válidos, novos, potencialmente úteis e compreensíveis em dados existentes.*

Por essa definição KDD é um processo. Por isso, é composto de diversas fases, que podem ser repetidas em várias iterações. Por não ser trivial, se entende que o KDD realiza inferências ou busca sobre os dados para a descoberta de padrões interessantes. Esses padrões devem ser novos e potencialmente úteis, ou seja, devem trazer algum benefício para o usuário. Além disso, devem ser compreendidos pelos analistas humanos.

Frequentemente o processo de KDD possui várias partes interessadas que são denominadas atores. Os três principais são:

Analista de dados: Detêm as técnicas necessárias para realização do processo de KDD. Tem conhecimento sobre os algoritmos e ferramentas envolvidas, mas não é necessário que possua conhecimento sobre o domínio do problema.

Especialista no domínio: Aquele que possui conhecimentos sobre o domínio ao qual o processo de KDD será aplicado. Por exemplo, em uma pesquisa que analisa dados sobre a biodiversidade do pantanal, o especialista de domínio será um biólogo.

Usuário: Peça fundamental no processo de KDD. É quem irá utilizar o resultado final do processo de KDD.

Note que apesar dos três atores terem tipos de habilidades diferentes, não significa necessariamente que serão pessoas diferentes. A mesma pessoa pode realizar diversos "papéis", inclusive o caso mais comum que é uma mesma pessoa realizando os três. Um papel que sempre existe nos processos de KDD é o papel do usuário, pois o problema a ser solucionado é de seu interesse. Como já foi mencionado, o KDD é um processo que possui diversas fases, cuja ilustração se encontra na Figura 2.5. Nela, podem ser vistas as fases de coleta, pré-processamento, transformação, mineração de dados e interpretação de dados, que posteriormente serão explanadas com mais detalhes.

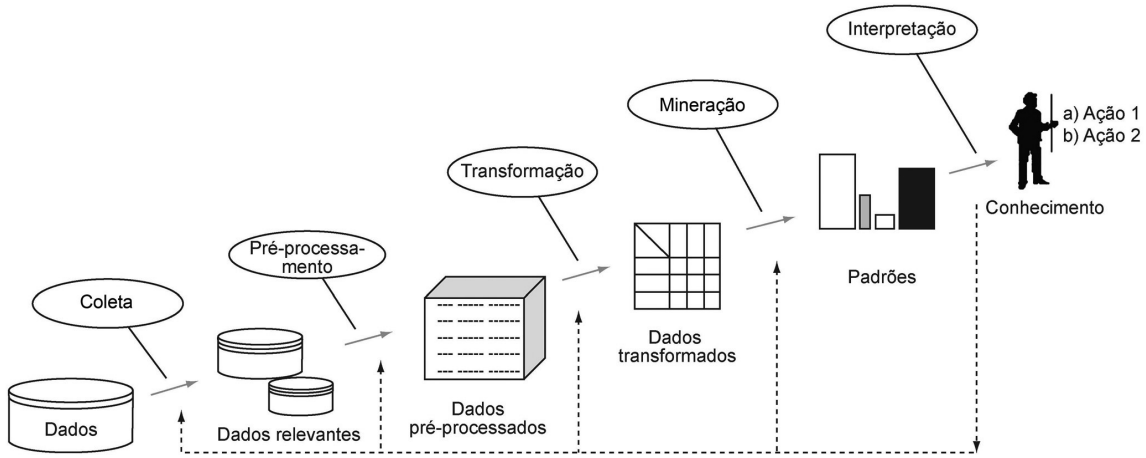


Figura 2.5: Fases do KDD (Fayyad et al., 1996) com adaptações

Antes de iniciar o processo de KDD é necessário identificar e compreender o problema. Para isso, o analista é responsável por "ouvir" o usuário e mapear seus problemas em algum método de Mineração de Dados. O analista conta com a ajuda do especialista no domínio para obter informações sobre o domínio da aplicação. Uma vez que o problema foi compreendido, nem sempre é fácil mapeá-lo para algum método de MD. Muitas vezes pode ser impossível. Nesses casos, costuma-se identificar subproblemas mais pertinentes. Também é importante analisar quais são os atributos mais importantes nos dados e verificar se os dados necessários já se encontram no banco de dados. Após isso, iniciam-se as fases do processo de KDD:

Coleta de dados: Caso os dados necessários não estejam presentes no banco de dados é necessário que sejam coletados de outras fontes. Essa é uma atividade crítica, pois os dados podem não estar formatados ou estar em um formato incompatível com os algoritmos utilizados. Muitas vezes, mesmo se disponíveis, necessitam ser rotulados com auxílio do especialista no domínio. Também é muito comum os dados estarem distribuídos entre muitos sistemas de banco de dados e em formatos variados. Nesses casos utiliza-se de bancos de dados chamados *Data Warehouses* (Kimball e Ross, 2002) para integração das diferentes fontes de dados de forma consistente. A fase de coletas de dados é considerada uma das mais trabalhosas no processo de KDD e nela pode-se encontrar muitas dificuldades como as citadas em Pyle (1999).

Pré-processamento de dados: Essa fase tem como objetivo melhorar a qualidade dos dados,

através da retirada de ruído (atributos com valores incorretos), valores ausentes, atributos de baixa relevância, entre outros.

Transformação de dados: Após o pré-processamento pode ser necessário transformar os dados para superar limitações intrínsecas dos algoritmos de extração de padrões que serão utilizados. Por exemplo, alguns algoritmos não conseguem trabalhar com datas, nesse caso pode-se aplicar alguma transformação como converter para número de dias decorridos a partir de uma data pré-fixada.

Mineração de dados: Nesta fase são utilizados métodos de diversas áreas diferentes como: Aprendizado de máquina, Estatística, Banco de dados, e entre outras. Uma boa escolha dos algoritmos aplicados é essencial. Isso se deve ao fato que nenhum algoritmo é ótimo para todas as aplicações.

Avaliação e interpretação dos resultados: Esta fase envolve todos os participantes no processo de KDD. O analista verifica se os classificadores usados atingiram a expectativa, utilizando métricas como taxa de erro, tempo de CPU gasto e complexidade do modelo. O especialista do domínio verifica se os resultados são compatíveis com o domínio. Ao usuário cabe decidir se os resultados são aplicáveis.

O resultado final do processo de KDD deve ser compreensível, mas compreensibilidade não é simples de definir e até o momento não existe um meio efetivo para medi-la. Craven e Shavlik (1995) dizem que a compreensibilidade é útil para validar o conhecimento, descobrir novos padrões, sugestão de quais atributos são mais relevantes, e refinar o conhecimento; pois resultados mais compreensíveis podem ser mais facilmente entendidos pelos especialistas do domínio. Uma outra medida frequentemente utilizada é chamada de interessabilidade (Tuzhilin, 1995), e consiste em um valor padronizado combinado das métricas de validade, utilidade e simplicidade. No final do processo de KDD o resultado são informações interessantes e úteis que devem ajudar na tomada de decisões. No próximo capítulo é visto um pouco mais da fase de pré-processamento de dados em detalhes, pois o foco deste trabalho é a imputação de valores ausentes que se encontra nesta fase do processo de KDD.

Nesse capítulo foram apresentados os conceitos introdutórios de Aprendizado de máquina, descoberta de conhecimento em banco de dados, bem como notações e outros conceitos necessários para explorar as idéias contidas neste trabalho. No próximo capítulo serão explanados os conceitos necessários para a compreensão da substituição de valores ausentes.

CAPÍTULO

3

Valores Ausentes

Este capítulo visa aprofundar os conceitos relativos aos valores ausentes e aos métodos tradicionais para lidar com tais valores. Além disso, descreve métodos de imputação comumente utilizados na literatura, algoritmos e métodos utilizados para o tratamento de valores ausentes e a utilização do aprendizado supervisionado e semissupervisionado a problemas de imputação de dados. Este capítulo é dividido da seguinte forma: na Seção 3.1 são apresentados os conceitos introdutórios de valores ausentes. Os mecanismos geradores de valores ausentes são tratados na Seção 3.2, os métodos de tratamento de valores ausentes e os métodos de imputação são tratados respectivamente nas Seções 3.3 e 3.4, os tipos de ausência são tratados na Seção 3.5 e por fim as Seções 3.6 e 3.7 tratam da utilização dos aprendizados supervisionados e semissupervisionados a tarefas de imputação e dos algoritmos usados.

3.1 Valores Ausentes

Os dados coletados diretamente de bancos de dados muitas vezes podem ser de má qualidade; tendo informações incorretas, imprecisas, valores ausentes e muitos outros problemas. Embora muitos algoritmos usados em mineração de dados estejam preparados para lidar com esses problemas, espera-se que obtenham melhores resultados se os problemas forem removidos ou corrigidos. Durante a fase de pré-processamento de dados são aplicados algoritmos que melhoram a sua qualidade, removendo ou minimizando esses problemas. O pré-processamento de dados é um processo semiautomático, já que depende do analista, pois este necessita ter um bom conhecimento do domínio para identificar os problemas presentes nos dados coletados e utilizar os métodos apropriados para solucioná-los. De acordo com Batista (2003) as tarefas de pré-processamento podem ser divididas de acordo com o nível de dependência do conhecimento de domínio:

Tarefas fortemente dependentes do conhecimento de domínio: Só podem ser realizadas com uso de conhecimento específico ao domínio. Um método automático pode executar uma tarefa deste tipo, mas necessitará de conhecimentos específicos fornecidos pelo analista. Por exemplo: no caso de verificação de integridade de dados, em um atributo nota, deve-se informar que tem de ser positiva. Isso pode ser feito com um conjunto de regras dependentes do conhecimento do domínio.

Tarefas fracamente dependentes do conhecimento de domínio: Nessas tarefas, os métodos usados conseguem extrair automaticamente dos dados as informações necessárias para tratar o problema. Deve-se notar que pode ser necessário conhecimento de domínio suficiente para saber qual método usar, mesmo assim é menor que o necessário para os métodos que dependem fortemente de conhecimento de domínio. Por exemplo: para se imputar um dado não é necessário ter um profundo conhecimento da base de dados, mas é necessário ter conhecimento suficiente para saber qual método de imputação pode ser usado nesta base.

Quando o tratamento de valores ausentes é uma tarefa fracamente dependente do conhecimento de domínio pode ser em grande parte automatizada. É nesta tarefa de tratamento de valores ausentes de forma automatizada, utilizando o aprendizado semissupervisionado que este trabalho tem seu foco. Os valores ausentes são muito comuns em banco de dados, consistem na não medição de um atributo para alguns casos (Batista, 2003). São causados por diversos motivos, tais como: recusa em responder perguntas, defeito em sensores, perda de dados em dispositivos de armazenamento, entre outros. Por ser um problema comum, costuma ter papel relevante na qualidade dos dados. Por isso deve-se tratá-los de maneira adequada. Apesar de sua relevância, muitos analistas costumam tratá-los de forma simplista, com a simples remoção dos registros onde ocorrem. Este tratamento pode, em muitos casos, invalidar a hipótese induzida.

3.2 Mecanismos de Ausência

Quando se trata com valores ausentes é muito importante conhecer o mecanismo (distribuição) de aleatoriedade que os gerou, para que se faça uma escolha acertada do método de tratamento. Em sua forma mais simples, os valores ausentes podem estar aleatoriamente distribuídos nos dados. Assim, a probabilidade de encontrar um valor ausente é a mesma para qualquer atributo. Contudo, existem casos onde os valores ausentes podem não estar aleatoriamente distribuídos e dependem do valor da própria variável ausente ou de relacionamentos entre os atributos.

Segundo Little e Rubin (2002) os valores ausentes podem ser divididos em três classes de acordo com seu valor de aleatoriedade:

Ausentes de forma completamente aleatória: Os valores ausentes distribuídos de forma completamente aleatória (*Missing completely at random* - MCAR) são os de maior grau de

aleatoriedade. Eles ocorrem quando o valor do atributo ausente não está relacionado com esse atributo ou qualquer outro no conjunto de dados. Normalmente, são gerados por fatores externos, tais como falhas em dispositivos, queima de sensores, entre outros. Por exemplo: na Tabela 3.1 que contém dados de alunos, pode ocorrer devido a algum motivo fortuito, a perda de parte dos dados por uma falha no sistema de armazenamento, resultando na Tabela 3.2 que contém valores ausentes. Os valores ausentes gerados nesse caso são do tipo MCAR, pois não existe nenhuma relação destes com a própria variável que contém os valores ausentes ou as outras variáveis contidas na tabela.

Tabela 3.1: Identificação única do aluno, horas de estudo e nota dos alunos

ID	horas de estudo	nota
2010011	6,0	9,2
2010035	9,0	7,5
2010052	5,0	6,7
2010079	6,0	6,2
2010075	10,0	6,1
2010022	10,0	5,7
2010041	3,0	5,2
2010060	4,5	4,0
2009019	4,0	3,2

Tabela 3.2: Tabela de identificação única do aluno, horas de estudo e nota dos alunos, com valores ausentes por mecanismo de ausência MCAR

ID	horas de estudo	nota
2010011	?	9,2
2010035	9,0	7,5
2010052	?	6,7
2010079	6,0	6,2
2010075	10,0	?
2010022	?	5,2
2010060	4,5	4,0
2009019	4,0	?

Ausentes de forma aleatória: Os valores ausentes de forma aleatória (*Missing at random* - MAR) ocorrem quando o valor do atributo não está relacionado com esse atributo, mas sim com um ou mais atributos do conjunto de dados. Por exemplo: na Tabela 3.1 os alunos que tiraram notas abaixo da média poderiam não responder o número de horas estudadas, resultando na Tabela 3.3. Assim, os valores ausentes teriam relação com a nota do aluno, mas não com seu próprio valor.

Tabela 3.3: Tabela de identificação única do aluno, horas de estudo e nota dos alunos, com valores ausentes causados por mecanismo de ausência MAR

ID	horas de estudo	nota
2010011	6,0	9,2
2010035	9,0	7,5
2010052	5,0	6,7
2010079	6,0	6,2
2010075	10,0	6,1
2010022	?	5,7
2010041	?	5,2
2010060	?	4,0
2009019	?	3,2

Ausentes de forma não aleatória: Os valores ausentes de forma não aleatória (*Not missing at random* - NMAR) ocorrem quando o valor do atributo ausente esta relacionado ao próprio atributo e a um ou mais atributos do conjunto de dados. Por exemplo: se os alunos da Tabela 3.4 fossem entrevistados e os alunos que estudaram mais horas e tiraram notas mais baixas que a maior nota da prova, não respondessem o número de horas que estudaram para a prova. Os valores ausentes resultantes seriam não aleatórios, pois saberia-se que todas as não respostas são de alunos que estudaram muitas horas mas com notas abaixo da maior nota.

Tabela 3.4: Tabela de identificação única do aluno, horas de estudo e notal dos alunos, com valores ausentes causados por mecanismo de ausência NMAR

ID	horas de estudo	nota
2010011	6,0	9,2
2010035	?	7,5
2010052	5,0	6,7
2010079	?	6,2
2010075	?	6,1
2010022	?	5,7
2010041	3,0	5,2
2010060	4,5	4,0
2009019	4,0	3,2

É importante notar que a simulação de valores NMAR não costuma ser utilizada na literatura para a simulação de valores ausentes em experimentos controlados. Isso se deve ao fato de que os algoritmos de imputação utilizam a informação disponível para imputar os valores ausentes, mas no caso de ausência NMAR esses valores podem não existir. Por exemplo: na Tabela 3.4

o especialista do domínio pode saber que os valores ausentes representam alunos que tiveram um desempenho insatisfatório, mas um algoritmo de imputação não terá dados suficientes para realizar a imputação dos valores.

3.3 Métodos de Tratamento de Valores Ausentes

Na literatura existem vários métodos para se tratar valores ausentes, mas muitos deles foram desenvolvidos para pesquisas de opinião como o método da substituição de casos, possuindo limitações quando são analisados do ponto de vista de KDD. Já outros métodos são muito simples, causando grandes distorções nos resultados das análises se forem utilizados sem os devidos cuidados.

Little e Rubin (2002) proporam a divisão dos métodos de imputação nas seguintes categorias:

Descarte de dados: Nesta abordagem realiza-se o descarte de exemplos que possuem valores ausentes. Existem duas formas de descarte dos dados: o descarte de exemplos e descarte de atributos. Na primeira, descartam-se todos os exemplos com valores ausentes; na segunda, é feita análise da extensão dos valores ausentes e são descartados apenas os atributos que possuem uma grande quantidade de valores ausentes. Este é um dos métodos mais simples e o mais amplamente utilizado; é o método padrão para tratamento de valores ausentes em muitos programas de mineração de dados, devido a sua simplicidade. Deve-se notar que o descarte só pode ser feito se os valores ausentes estiverem distribuídos de forma aleatória, pois o descarte de valores não distribuídos aleatoriamente pode alterar os resultados.

Estimativa de parâmetros: Utiliza métodos estatísticos para estimar parâmetros de modelos, e a partir desses modelos são estimados os valores ausentes.

Imputação: A imputação estima os valores ausentes obtidos da análise de valores coletados de outros exemplos (Sande, 1983). Existem várias técnicas de imputação que vão de técnicas simples como o uso da média ou moda, até técnicas mais avançadas que usam aprendizado de máquina. Os métodos mais avançados costumam ser mais complexos e custosos, mas geralmente geram melhores resultados. A imputação de dados pode ser univariada, quando existe apenas um atributo com valores ausentes e multivariada quando mais de um atributo possui valores ausentes.

3.4 Métodos de Imputação

Como citado anteriormente, a imputação consiste em substituir os valores ausentes por valores estimados. Os métodos de substituição de valores ausentes tem sido estudados há anos, mas muitos desses métodos foram criados para pesquisas de opinião e normalmente não são diretamente aplicados ao KDD.

A seguir são descritos alguns dos métodos encontrados na literatura :

Substituição de casos: Utilizado em pesquisas de opinião, se uma amostra possui valores ausentes ela é simplesmente substituída por outra. Por exemplo: quando um entrevistado se recusa a responder alguma pergunta na pesquisa, simplesmente é substituído por outra amostra, isto é, outra pessoa que não fazia parte da amostra original.

Imputação pela média ou moda: É um dos métodos mais utilizados, consiste na substituição dos valores ausentes pela média (no caso de atributos numéricos) ou pela moda (atributos nominais), calculadas a partir dos valores observados dos atributos. A média normalmente é uma boa estimativa de um valor ausente, quando não se tem nenhuma outra informação a respeito do conjunto de dados. É um método conservador que possui como principal vantagem não alterar a média geral do atributo, mas altera a variância dos dados reduzindo seu valor e também pode alterar o relacionamento entre os atributos. Uma forma de minimizar esses problemas em tarefas de classificação é utilizar o atributo de classe para separar os exemplos em grupos e calcular a média ou moda separadamente para cada classe.

Conhecimento de domínio: Um especialista no domínio pode usar seu conhecimento para substituir os valores ausentes por valores estimados por meio de sua experiência. Esse procedimento pode ser considerado seguro se o especialista está familiarizado com a aplicação utilizada, se o conjunto de dados é grande e o número de valores ausentes é pequeno. Além disso, os valores dos atributos numéricos podem ser discretizados para aumentar a margem de segurança com que o especialista imputa aos valores. Por exemplo: no caso dos alunos, o atributo nota poderia ser transformado em conceitos como A, B, C ou D, de forma a aumentar a margem de confiança ao se imputar o valor ausente. Não se pode esquecer que essa transformação pode causar perda de informação.

Normalmente, o especialista no domínio costuma ter uma precisão maior que a simples substituição pela média ou moda. Entretanto, suas estimativas são limitadas ao seu conhecimento sobre os dados do domínio, o que de certa forma pode direcionar o conhecimento a ser aprendido.

Imputação local (procedimentos *Hot-deck* ou *Cold-deck*): Neste método os valores ausentes são imputados a partir de um doador similar escolhido aleatoriamente. O nome *Hot-deck* vem da época em que os dados eram armazenados em cartões perfurados e significa que o exemplo doador vem do mesmo conjunto de cartões *(deck)* que o exemplo com valores ausentes, ou seja do conjunto de cartões que estavam sendo processados naquele momento, já no *Cold-deck* o exemplo viria de um outro conjunto (cartão) do qual o exemplo com valores ausentes não fazia parte. O processo consiste de duas etapas: na primeira é utilizado um método de agrupamento não supervisionado para se separar os exemplos similares, posteriormente os exemplos com valores ausentes são associados a cada um desses agrupamentos e os seus valores são preenchidos usando a média ou moda do agrupamento escolhido ou simplesmente escolhendo aleatoriamente um exemplo doador e

utilizando os valores de seus atributos para preencher os valores que estão ausentes no outro exemplo.

Imputação múltipla: Método criado por Rubin (1987), consiste na criação de vários conjuntos de dados completos com valores imputados que podem ser criados usando, por exemplo, o algoritmo EM ou outras técnicas de imputação. Após isso, cada um desses conjuntos é analisado e os resultados são reunidos usando operações simples denominadas Regras de Rubin, cuja principal vantagem é permitir a estimação do erro gerado pela imputação. Esse método é considerado como o estado da arte em imputação na área de estatística e, atualmente, é o método padrão em vários programas estatísticos.

Modelos preditivos: Os Modelos preditivos atualmente é o estado da arte no tratamento de valores ausentes em KDD. São utilizados métodos estatísticos ou de aprendizado de máquina para a criação de um modelo preditivo para estimar os valores que serão usados para substituir os valores ausentes. O atributo com valor ausente é utilizado como atributo de classe e o restante dos atributos é utilizado como entrada para o modelo preditivo.

Os modelos preditivos utilizam as correlações entre os atributos para estimarem os valores ausentes, por isso é necessário que exista correlações entre os atributos no conjunto de dados, caso contrário os valores ausentes não serão corretamente estimados. Outra limitação dos modelos preditivos é que os valores ausentes resultantes são mais bem comportados que os valores ausentes reais, ou seja, serão mais consistentes com as relações entre os atributos do conjunto do que poderiam ser os dados reais.

3.5 Tipos de Ausência

Muitos dos métodos citados anteriormente lidam apenas com ausência de valores em apenas um atributo (ou coluna), ou seja, ausência univariada de dados. Contudo, em bases de dados reais é mais comum encontrar a ausência multivariada, ou seja, valores ausentes em mais de um atributo na base de dados. Ao aplicar estes métodos na resolução de problemas de ausência multivariada de dados, surgem diversos problemas que impedem sua utilização direta (van Buuren et al., 2006):

- Os exemplos utilizados na imputação podem apresentar valores ausentes em mais de um atributo;
- É possível encontrar casos de dependência circular, ou seja, um ou mais atributos possuem forte correlação entre si, mas todos podem estar ausentes em um exemplo, impedindo sua correta estimação;
- As variáveis são de tipos diferentes, restringindo a gama de modelos que podem ser aplicados;
- A relação entre um valor ausente e seus preditores pode ser complexa e não-linear; e

- A imputação pode criar casos impossíveis, por exemplo: homens grávidos.

Segundo van Buuren et al. (2006) os principais métodos capazes de solucionar problemas de ausência multivariada de dados podem ser divididos em duas categorias:

Joint-Modelling: utiliza modelos preditivos para estimar todos os valores ausentes de uma única vez, como redes Bayesianas.

Imputação iterativa: consiste na divisão de um problema multivariado em *n* problemas univariados, onde cada atributo com valor ausente é resolvido de maneira independente, utilizando técnicas para a solução de problemas univariados.

A Imputação Iterativa pode ser considerado um método de divisão e conquista, pois reduz a complexidade de um problema multivariado transformando-o em vários problemas univariados e devido a isso é considerado o método mais eficiente por diversos pesquisadores (Gelman e Raghunathan, 2001; Little e Rubin, 2002; van Buuren et al., 2006).

A imputação iterativa pode ser considerada uma generalização da imputação univariada, em que esta é aplicada em cada atributo que apresente valores ausentes. Na Figura 3.1 é ilustrado tal processo.

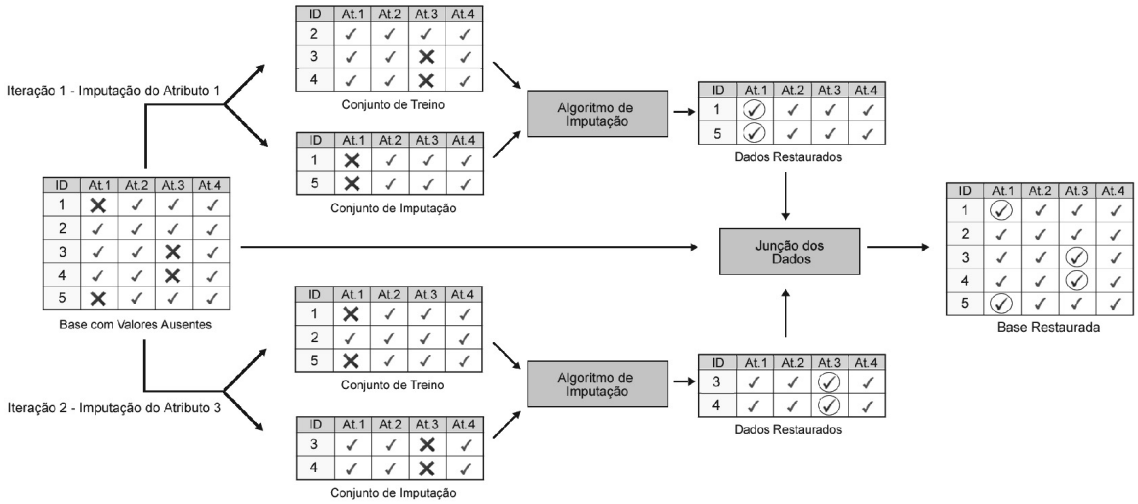


Figura 3.1: Processo de imputação univariada (Castaneda et al., 2008)

Na primeira iteração a base de dados divide-se em dois conjuntos; um conjunto de treino e um de imputação. O atributo *At.1* do conjunto de imputação é o atributo que terá seus valores ausentes estimados. O algoritmo de imputação é treinado com o conjunto de treino usando o *At.1* como atributo de classe. Posteriormente, usa-se para predizer os valores de *At.1*. Na segunda iteração, realiza-se o mesmo processo, só que agora o atributo a ser imputado é o atributo *At.3*. Após o término da segunda iteração, realiza-se a junção dos dados e a base de dados é restaurada. Desta forma, o processo de imputação ocorre em duas iterações em que cada atributo com valores ausentes é tratado como um problema univariado, sendo resolvido de maneira completamente independente do outro. Esta abordagem apresenta uma série de benefícios. É

capaz de reduzir a complexidade de um problema multivariado para *n* problemas univariados, onde *n* é o número de colunas com valores ausentes. Tornando-se possível a aplicação de diversas técnicas conhecidas na resolução de problemas univariados, como o algoritmo *k-NN*, redes Bayesianas, entre outros (Gelman e Raghunathan, 2001; Little e Rubin, 2002).

Uma deficiência da imputação iterativa reside no fato dos atributos serem imputados de maneira completamente independente, perdem-se assim, as correlações que possam existir entre os atributos, podendo gerar problemas de inconsistência. Outra questão, surge quando ocorrem valores ausentes no mesmo exemplo de forma que um atributo ausente tenha forte correlação com outro atributo ausente, o que os impossibilita de serem corretamente imputados, criando o problema denominado de ausência colinear (Abayomi et al., 2008).

Uma solução para esses problemas da imputação iterativa é a utilização de realimentação dos dados imputados a cada iteração criando o método denominado imputação iterativa com realimentação (Castaneda et al., 2008). Desta forma, os dados imputados para um atributo são adicionados a base de dados original antes do próximo atributo ser imputado, e não ao final de todas as iterações. O objetivo é preservar com mais eficiência a correlação entre os atributos imputados. Na Figura 3.2 ilustra-se o uso da realimentação entre as iterações.

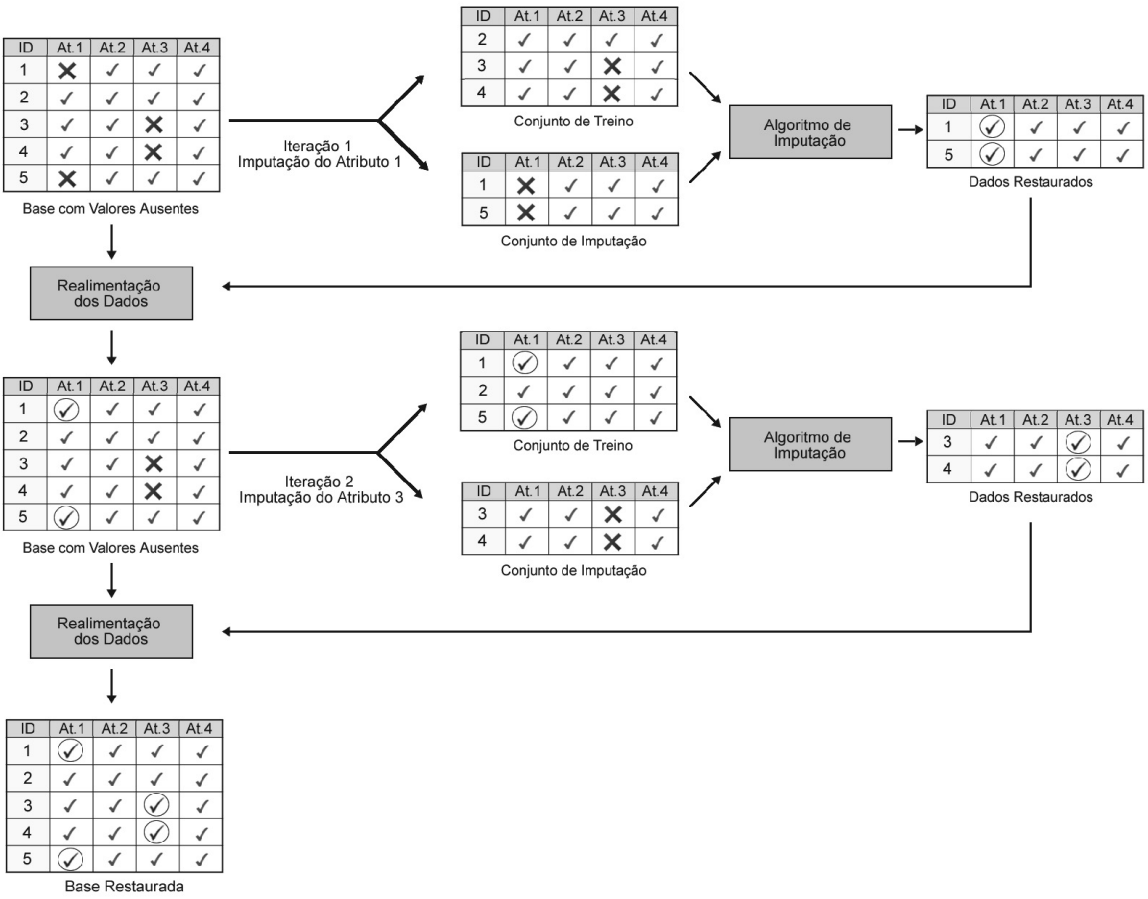


Figura 3.2: Processo de imputação univariada com realimentação (Castaneda et al., 2008)

Com a utilização da realimentação entre cada iteração, o conjunto de treino completa-se progressivamente, aproximando-se da completude a cada iteração. Suavizando os problemas em exemplos com a ocorrência de muitos atributos com valores ausentes, a relação entre os

atributos também é preservada. Contudo, é importante lembrar que valores imputados quase sempre possuem um percentual de erro, e ao serem reintroduzidos no conjunto de dados a cada iteração, o analista poderá contribuir para uma propagação de erros, imputando novos valores utilizando valores previamente imputados e a ordem de imputação dos atributos também altera o resultado final da imputação.

3.6 Imputação Utilizando Aprendizado Supervisionado

O processo básico de imputação de dados é apresentado na Figura 3.3. O conjunto de dados é dividido em dois conjuntos: o de treino que possui apenas os exemplos completos e o de imputação que tem os exemplos com valores ausentes. Após isso, os conjuntos são processados por um algoritmo supervisionado de imputação que preenche os valores ausentes e como resultado se obtém dados completos. Note que, apesar de qualquer algoritmo de imputação poder ser usado, a divisão em treino e imputação é semelhante à divisão treino e teste do aprendizado supervisionado. E apenas os dados completos do conjunto de treino são usados para induzir a hipótese que irá estimar os valores ausentes. Desta forma, o conjunto de imputação não é utilizado pelo algoritmo de imputação na indução da hipótese.

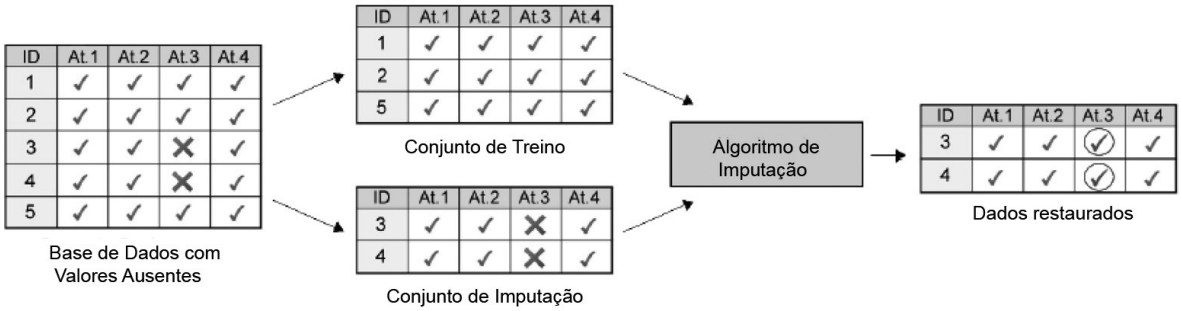


Figura 3.3: Imputação utilizando algoritmo supervisionado (Castaneda et al., 2009)

Teoricamente qualquer algoritmo supervisionado pode ser utilizado no tratamento de valores numéricos e nominais multi classe, entre outros, mas alguns algoritmos possuem características que os tornam mais interessantes para serem utilizados na imputação de dados. Estes algoritmos são comumente utilizados para tarefas de imputação e por isso são mostrados em detalhes a seguir.

3.6.1 k-NN (k-Nearest Neighbor)

O algoritmo *k*-NN (k-Nearest Neighbor) é um algoritmo de aprendizado preguiçoso (*lazy*) e não paramétrico. O não paramétrico se deve ao fato de que o *k*-NN não faz qualquer pressuposto sobre a distribuição dos dados, isto é muito útil em bases de dados que possuem uma distribuição desconhecida. O termo preguiçoso se refere ao fato de o algoritmo postergar a maior parte do processamento para o momento da predição. Com isso, o *k*-NN reterá todo ou grande parte do conjunto de treino para realizar a etapa de teste, o que pode ser muito custoso dependendo

do tamanho da base de dados. Na Figura 3.4 são ilustrados os passos usados pelo k-NN para rotular um novo exemplo.

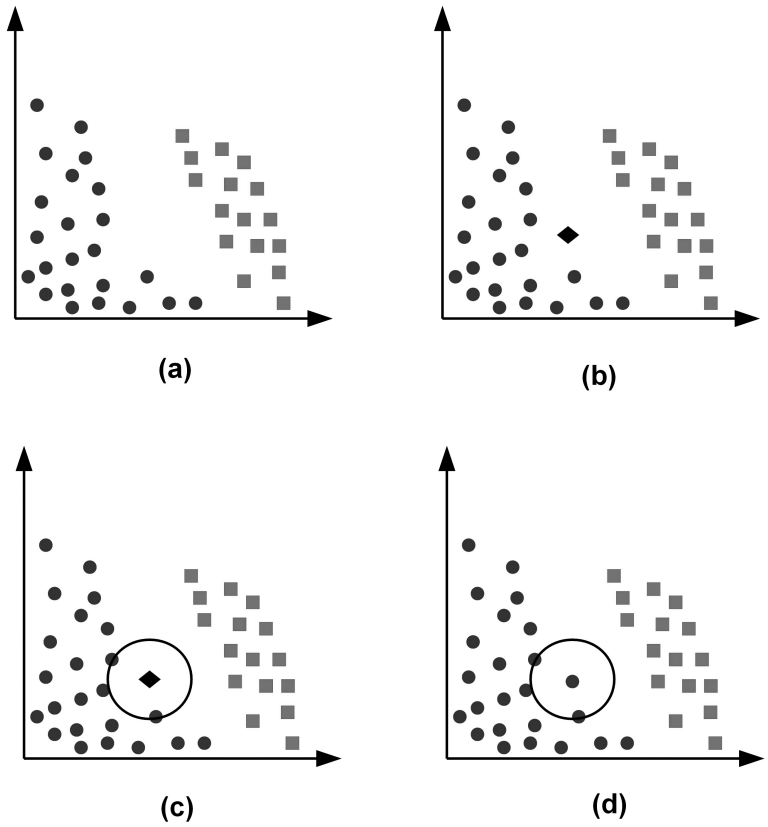


Figura 3.4: Funcionamento do algoritmo k-NN

Na Figura 3.4.a, são ilustrados os exemplos de treino armazenados pelo algoritmo k-NN. Posteriormente, um novo exemplo precisa ser classificado (Figura 3.4.b), para isso a distância entre o novo exemplo e cada um dos exemplos de treinamento é calculada utilizando-se alguma medida de similaridade, por exemplo, a distância Euclidiana, ilustrada na Figura 3.4.c, que entre dois exemplos x_i e x_j cada um representado por um vetor de características $n-$dimensional $x = (a_1, a_2, \ldots, a_n) \in \mathbb{R}^n$ atributos é definida como:

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_{ir} - a_{jr})^2}$$

Os k exemplos de treino mais próximos, isto é, mais similares ao novo exemplo são selecionados. O novo exemplo é então classificado pela moda (atributo discreto) ou média (atributo contínuo) do atributo de classe dos k exemplos mais próximos como ilustrado na Figura 3.4.d. O k-NN é frequentemente usado em imputação por possuir as seguintes características (Acuna e Rodriguez, 2004):

- Consegue lidar tanto com atributos contínuos (usando a média entre o k-vizinhos mais próximos) quanto atributos discretos (utilizando a moda);

- Não necessita construir um modelo para cada atributo com valores ausentes;
- Pode facilmente lidar com instâncias contendo múltiplos valores ausentes; e
- Leva em consideração estrutura de correlação entre os dados.

Por outro lado também, existem alguns problemas associados ao seu uso:

- A escolha da métrica que será usada para medir a similaridade entre os exemplos, que pode ser a distância Euclidiana, Manhattan, Mahalanobis, entre inúmeras outras;
- O algoritmo k-NN tem um custo computacional elevado, pois varre todo o conjunto de exemplos de treino procurando os exemplos mais similares. Isso consome muito tempo e pode ser crítico quando se trabalha em grandes bases de dados; e
- A escolha do valor de k também é crítico para tarefas de imputação, pois valores pequenos podem degradar a performance do classificador, uma vez que poucos exemplos podem ser dominantes no processo de estimação e grandes valores podem incluir instâncias com diferença significativa da instância com valores ausentes, também degradando a estimação.

Em experimentos realizados por Troyanskaya et al. (2001), foram avaliadas diferentes medidas de similaridade (correlação de Pearson, distância Euclidiana e minimização da variância), concluindo-se que a distância Euclidiana é suficiente. Além disso, analisou-se também a sensibilidade ao valor de k. É sugerido por alguns autores o uso de k=10, desde que existam pelo menos 10 vizinhos mais próximos a uma instância que contém valores ausentes.

3.6.2 Árvores de Decisão

Os algoritmos de árvores de decisão comumente são baseados na estratégia de divisão e conquista, árvore é construída de forma recursiva através da divisão sucessiva do conjunto de dados em subconjuntos cada vez mais "puros" em relação ao atributo classe. A estrutura recursiva da árvore de decisão é definida como:

- Um nó folha que indica uma classe, ou
- um nó de decisão que testa o valor de um atributo, o resultado do teste leva a uma subárvore e toda subárvore tem a mesma estrutura da árvore.

Por exemplo: considere o conjunto de exemplos *jogar tênis* (Mitchell, 1997) com modificações, que é descrito na Tabela 3.5 e representa a decisão de jogar ou não tênis de acordo com as condições climáticas do dia. Esse conjunto possui os seguintes atributos:

aparência: se o dia é ensolarado, chuvoso ou nublado;

temperatura: se a temperatura no dia é agradável, fria ou normal ;

umidade: se a umidade do dia é alta ou normal;

ventando: se está ou não ventando; e

jogar: se é um bom dia para jogar tênis ou não.

Tabela 3.5: Conjunto de dados *jogar tênis*

dia	aparência	temperatura	umidade	ventando	jogar tênis
d_1	sol	quente	alta	falso	não
d_2	sol	quente	alta	verdadeiro	não
d_3	nublado	quente	alta	falso	sim
d_4	chuva	agradável	alta	falso	sim
d_5	chuva	fria	normal	falso	sim
d_6	chuva	fria	normal	verdadeiro	não
d_7	nublado	fria	normal	verdadeiro	sim
d_8	sol	agradável	alta	falso	não
d_9	sol	fria	normal	falso	sim
d_{10}	chuva	agradável	normal	falso	sim
d_{11}	sol	agradável	normal	verdadeiro	sim
d_{12}	nublado	agradável	alta	verdadeiro	sim
d_{13}	nublado	quente	normal	falso	sim
d_{14}	chuva	agradável	alta	verdadeiro	não

Na Figura 3.5 é ilustrada uma árvore de decisão entre as possíveis de serem obtidas a partir do conjunto de exemplos *jogar tênis*. A classificação de um novo exemplo é realizada testando as condições a partir do nó raiz até atingir um nó folha que contenha a classe que será atribuída ao novo exemplo.

Uma árvore de decisão pode também ser representada na forma de um conjunto de regras. Cada regra começa no nó raiz da árvore e caminha em direção as folhas, cada nó de decisão acrescenta uma premissa a regra e o nó folha representa a conclusão da regra. Por exemplo: o caminho do nó raiz ao primeiro nó folha corresponde a expressão:

$$aparência = sol \wedge umidade = alta$$

A classe "não" da árvore de decisão é ilustrada na Figura 3.5 corresponde à seguinte regra:

$$(aparência = sol \wedge umidade = alta) \vee (aparência = chuva \wedge ventando = verdadeiro)$$

Essa representação da árvore de decisão como disjunções de regras é muito útil, pois facilita a leitura e a interpretação da árvore induzida. As árvores de decisão particionam o espaço de

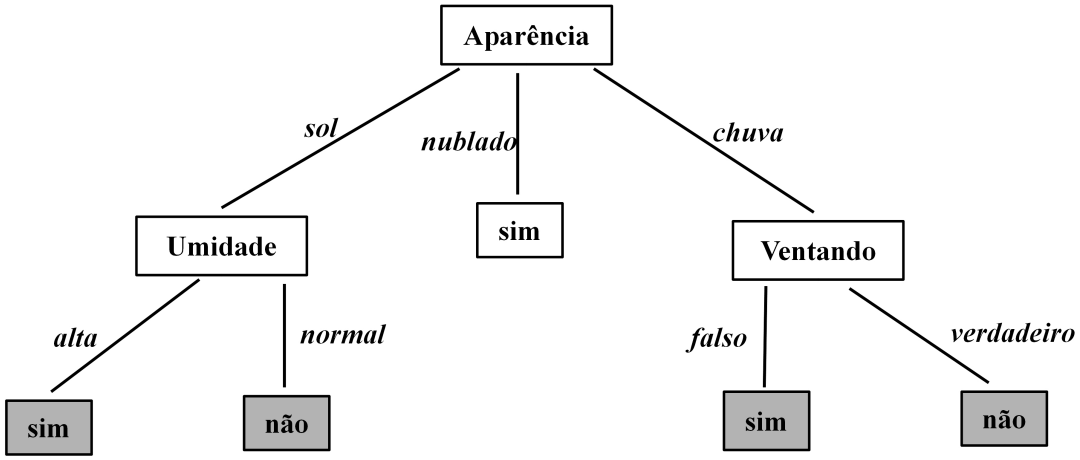


Figura 3.5: Árvore de decisão induzida utilizando o conjunto de dados *jogar tênis*

exemplos de forma que os subconjuntos gerados sejam cada vez mais "puros". Isso é feito através da escolha do atributo que melhor separa os exemplos. Por exemplo: o atributo *umidade* pode dividir os exemplos em dois subconjuntos determinados pelos valores "alta" e "normal", no subconjunto "alta" temos aproximadamente 57% de exemplos da classe "não" e no subconjunto "normal" temos 86% de exemplos da classe "sim". O atributo *ventando* também divide os exemplos em dois subconjuntos, mas um possui 75% de exemplos da classe "sim" e o outro possui 50% de exemplos de ambas as classes. Desta forma, o atributo *umidade* gera subconjuntos com maior probabilidade dos exemplos pertencerem a uma das classes, ou seja, de serem mais "puros". A medida mais utilizada para a impureza dos conjuntos é a entropia de Shannon (Shannon, 1949), podendo ser definida em problemas de classe binária para um conjunto de exemplos S como:

$$entropia(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus} \tag{3.1}$$

Onde $p\oplus$ é a proporção de exemplos positivos e $p\ominus$ a de negativos, o resultado é expresso em bits. Uma explicação para a entropia pode ser feita da seguinte forma: se variarmos a proporção de exemplos positivos na Equação 3.1 a entropia será máxima quando se obtiver uma proporção de 0,5, ou seja, uma quantidade igual de exemplos positivos e negativos; e será mínima quando se obtiver as proporções de 0 ou 1, ou seja, respectivamente somente exemplos negativos ou somente positivos. Esta situação é ilustrada na Figura 3.6.

Do ponto de vista da teoria da informação a entropia é vista como o número de bits de informação necessários para codificar um membro qualquer de S, ou seja, se por exemplo: É necessário transmitir a informação referente a um sorteio, e a proporção de exemplos positivos é igual a 1, o receptor saberá de antemão que o exemplo sorteado é positivo e nenhuma mensagem precisa ser enviada. Com isso, a entropia é 0. Contudo, se a proporção é 0.5, então o receptor

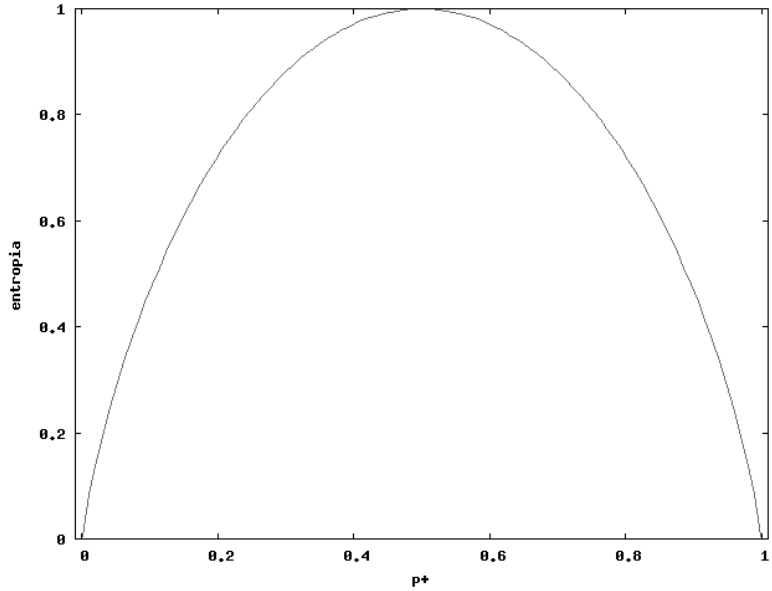


Figura 3.6: Entropia para classe binária

não tem como saber se o exemplo sorteado é positivo ou negativo sendo necessário utilizar 1 bit de informação para dizer qual é a classe do exemplo sorteado e com isso a entropia é 1. Todavia, se a proporção de exemplos positivos for igual a 0,7, pode-se enviar uma série de mensagens e utilizar-se de códigos menores para codificar a informação de um conjunto de exemplos positivos e codificar o conjunto de exemplos negativos que são mais incomuns com códigos mais longos.

O conjunto de exemplos *jogar tênis* possui um total de 14 exemplos sendo 9 positivos e 5 negativos e será adotada a notação $[p\oplus, p\ominus]$ para descrever o número de exemplos por classe. A entropia para este conjunto é dada por:

$$Entropia([9+,5-]) = -\frac{9}{14} \times \log_2 \frac{9}{14} - \frac{5}{14} \times \log_2 \frac{5}{14} = 0,94029$$

Tendo a entropia como uma medida da impureza em um conjunto de exemplos, pode-se definir uma medida que mede quão bem um atributo divide os exemplos positivos e negativos. Essa medida chamada ganho de informação mede a redução da entropia causada pelo particionamento dos exemplos usando-se um determinado atributo e é definida pela Equação:

$$GanhoInfo(S, Atributo) \equiv Entropia(S) \; - \sum_{v \in Dom(Atributo)} \frac{|S_v|}{|S|} \times Entropia(S_v)$$

Onde; S é um conjunto de exemplos; v é um valor do atributo e Sv é o subconjunto de S no qual o valor desse atributo é v.

Note-se que o primeiro termo é simplesmente a entropia do conjunto S, já o segundo termo é a soma da entropia de cada subconjunto pertencente a Sv, ponderada por $\frac{|S_v|}{|S|}$.

Para se construir uma árvore de decisão, primeiramente cada atributo é avaliado como ilustrado na Figura 3.7.

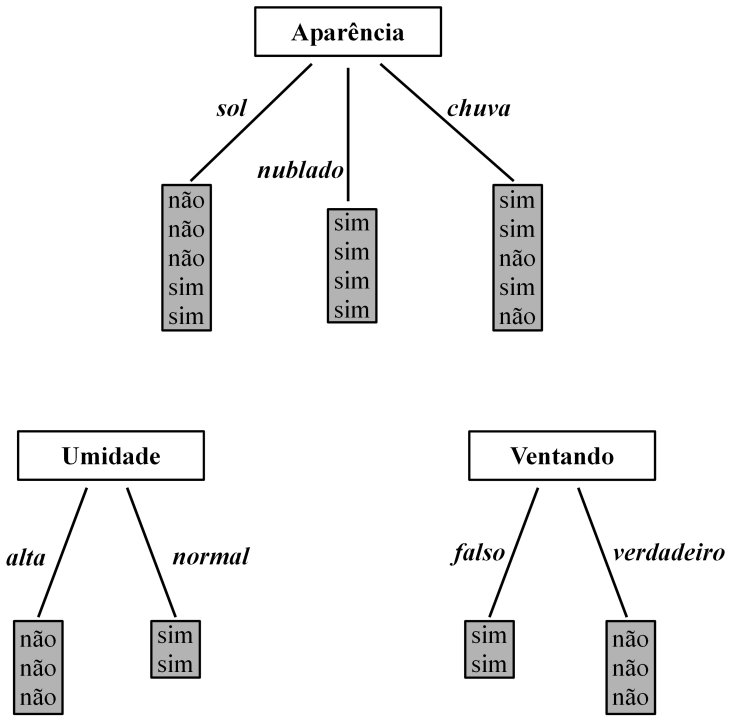


Figura 3.7: Conjunto *jogar tênis*: árvore de decisão parcial (*decision stumps*)

O valor de cada atributo divide os exemplos em subconjuntos. Por exemplo, o atributo *aparência* pode ser dividido em três subconjuntos : $aparência_{sol}[2+,3-]$, $aparência_{nublado}[4+,0-]$ e $aparência_{chuva}[3+,2-]$, o cálculo do ganho de informação para esse atributo é calculado da seguinte forma:

$$\begin{aligned}
GanhoInfo([9+,5-],aparência) &= Entropia([9+,5-]) - \sum_{v\in\{sol,nublado,chuva\}} \frac{|S_v|}{|S|} \\
&\times Entropia(S_v) \\
&= Entropia([9+,5-]) - \frac{5}{14} \times Entropia([2+,3-]) - \frac{4}{14} \\
&\times Entropia([4+,0-]) - \frac{5}{14} \times Entropia([3+,2-]) \\
&= 0.94029 - \frac{5}{14} \times 0.97095 - \frac{5}{14} \times 0 - \frac{5}{14} \times 0.97095 \\
&= 0.94029 - 0.69354 \\
&= 0.24675
\end{aligned}$$

Calculando para os outros atributos:

$$\begin{aligned}
GanhoInfo([9+,5-],temperatura) &= 0.94029 - 0.91106 = 0.02922 \\
GanhoInfo([9+,5-],umidade) &= 0.94029 - 0.78845 = 0.15184 \\
GanhoInfo([9+,5-],ventando) &= 0.94029 - 0.89216 = 0.04813
\end{aligned}$$

Comparando-se os valores, o atributo com maior ganho de informação foi *aparência* e por isso será o nó raiz da árvore, como o atributo *nublado* somente possui exemplos positivos, torna-se um nó folha da árvore. Ao se expandir o nó do atributo $aparência_{sol}$ os mesmos cálculos devem ser feitos mas agora para os exemplos que possuem o valor "sol" no atributo *aparência*.

$$GanhoInfo([2+,3-],temperatura) = 0.97095 - 0.4 = 0.57095$$
$$GanhoInfo([2+,3-],umidade) = 0.97095 - 0 = 0.97095$$
$$GanhoInfo([2+,3-],ventando) = 0.97095 - 0.95098 = 0.01997$$

umidade é o atributo com maior ganho de informação, como $umidade_{alta}$ possui somente exemplos negativos, é acrescentado a árvore como um nó folha, o mesmo acontece para $umidade_{baixa}$. O próximo ramo a ser expandido é $aparência_{chuva}$ como ilustrado na Figura 3.8.

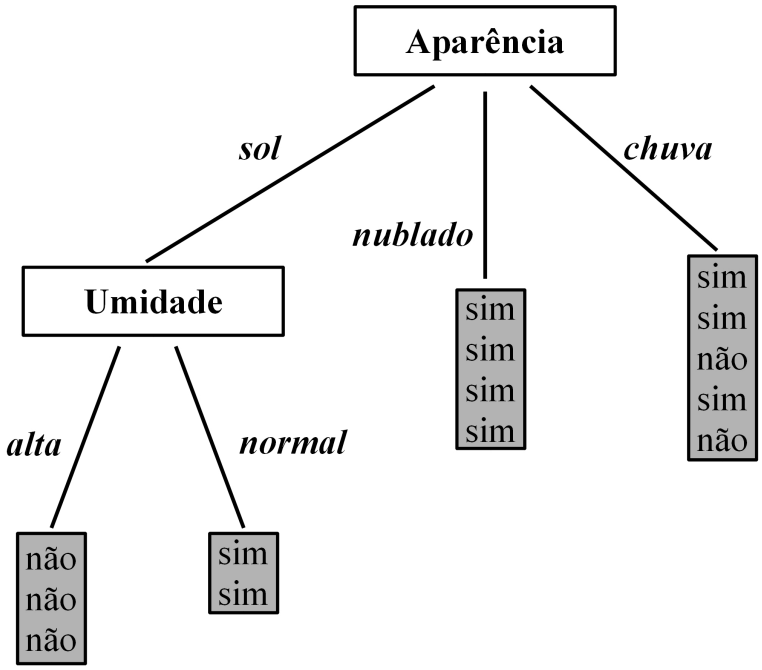


Figura 3.8: Conjunto *jogar tênis*: expandindo a sub-árvore esquerda

Calculando-se os ganhos de informação para $aparência_{chuva}$ obtém-se os seguintes valores:

$$GanhoInfo([3+,2-],temperatura) = 0.97095 - 0.95098 = 0.01997$$
$$GanhoInfo([3+,2-],umidade) = 0.97095 - 0.95098 = 0.01997$$
$$GanhoInfo([3+,2-],ventando) = 0.97095 - 0 = 0.97095$$

O atributo com maior ganho de informação é *ventando* e como $ventando_{falso}$ possui somente exemplos positivos, é transformado em um nó folha, o mesmo ocorre com $ventando_{verdadeiro}$ que possui somente exemplos negativos.

Como as folhas só possuem exemplos ou positivos ou negativos o ganho de informação é zero e a construção da árvore é interrompida resultando na árvore ilustrada na Figura 3.9.

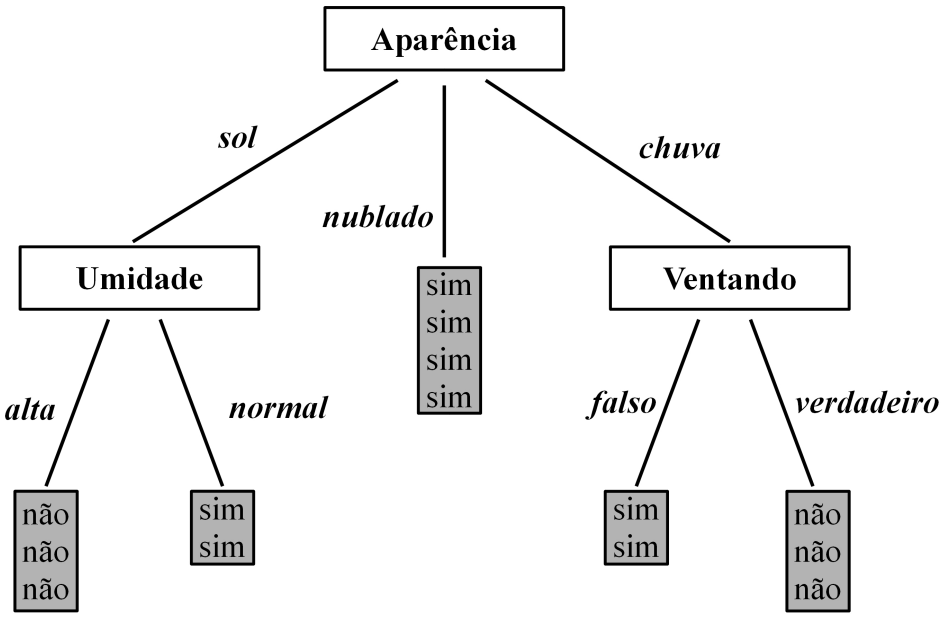


Figura 3.9: Conjunto *jogar tênis*: árvore resultante

Em conjuntos de dados maiores ou mais complexos pode acontecer da árvore resultante ser muito longa para se atingir um ganho próximo a zero. Nesses casos pode-se ocorrer o super ajuste na árvore, fazendo com que ela tenha alta precisão para o conjunto de treino e alta taxa de erro para o de teste. Para evitar esse problema, costuma-se definir um limiar do critério de particionamento que determina se a construção da árvore continua ou deve ser interrompida. Mesmo assim, pode ocorrer o super ajuste e nesse caso a árvore pode ser podada. A poda consiste em uma validação cruzada interna, ou seja, poda-se a árvore enquanto o erro do conjunto de teste for reduzindo. Quando o erro aumentar o processo é interrompido.

Uma das grandes vantagens da árvore de decisão é a facilidade de interpretação da hipótese por ela representada, sendo assim amplamente utilizada em processos de mineração de dados. Outra vantagem é que a classificação de exemplos usando-se uma árvore de decisão não requer a avaliação de todos os atributos que compõem o exemplo, requerendo-se a avaliação apenas dos atributos que compõem a árvore. Devido a isso, o classificador de árvores de decisão é um dos mais rápidos algoritmos para classificação de exemplos.

3.6.3 naive Bayes

Os métodos Bayesianos oferecem uma abordagem probabilística para a inferência de hipóteses. O aprendizado bayesiano diminui ou aumenta a probabilidade associada a uma hipótese ao invés de simplesmente descartar ou aceitar a hipótese. Os algoritmos de aprendizado de máquina geralmente tentam encontrar a melhor hipótese em um conjunto de exemplos de treinamento. Para o aprendizado bayesiano a melhor hipótese é a hipótese mais provável (Mitchell,

1997).

Para introduzir o teorema de bayes que é a base do aprendizado bayesiano, primeiramente faz-se necessário definir a seguinte notação: considere uma hipótese h onde, $P(h)$ denota a probabilidade inicial da hipótese h antes mesmo do conjunto de dados ser visto e é chamada de probabilidade a priori. Ela reflete qualquer conhecimento prévio que se possui sobre h, por exemplo, ao lançar uma moeda justa, sabe-se de antemão que a probabilidade de dar cara ou coroa é 0,5 para cada lado. Similarmente será definido $P(D)$ como a probabilidade a priori do conjunto de dados. $P(D|h)$ como a probabilidade condicional, ou seja a probabilidade de um conjunto de dados D dada a hipótese h. Note-se que em aprendizado de máquina tem-se interesse em $P(h|D)$ que é a chamada probabilidade a posteriori que pode ser calculada usando a regra de Bayes:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

No aprendizado de máquina, o interesse é encontrar uma hipótese h de um conjunto de hipóteses H onde $h \in H$ que seja a mais provável para o conjunto de dados D. Sendo Y o conjunto de classes equivalente ao conjunto de hipóteses H, $P(y_c)$ irá denotar a probabilidade a priori de um exemplo pertencer a classe y_c, sendo que $y_c \in Y$, $P(y_c)$ reflete qualquer conhecimento sobre a chance de y_c ser a classe correta. Similarmente $P(x)$ será a probabilidade a priori de um exemplo do conjunto de treino $x \in X$ sendo que x é um vetor n-dimensional $x = (a_1, a_2, \ldots, a_{n_{atr}}) \in \mathbb{R}^{n_{atr}}$ atributos, e $P(x|y_c)$ a probabilidade desta classe pertencer ao exemplo x.

A probabilidade a posteriori $P(y_c|x)$ pode ser calculada usando a probabilidade máxima a posteriori (MAP) que será denominada y_{MAP} .

$$\begin{aligned} y_{MAP} &= \arg\max_{y_c \in Y} P(y_c|x) \\ &= \arg\max_{y_c \in Y} \frac{P(x|y_c)P(y_c)}{P(x)} \\ &= \arg\max_{y_c \in Y} P(x|y_c)P(y_c) \end{aligned} \tag{3.2}$$

No último passo $P(x)$ foi desconsiderado por ser uma constante independente de h. A probabilidade $P(y_c)$ pode ser estimada através da frequência com que cada valor $y_c \in Y$ ocorre nos exemplos de treinamento. Mas para estimar $P(a_1, a_2, \ldots a_{n_{atr}}|y_c)$ é necessário que a probabilidade condicional seja conhecida para qualquer valor possível para cada atributo. Por exemplo, em um conjunto de dados com 20 atributos binários seria necessário estimar 2^{20} probabilidades condicionais, isso equivale a calcular mais de um milhão de probabilidades condicionais. Esta abordagem é inviável, pois para estimar essas probabilidades com algum grau de confiança o conjunto de exemplos teria de ser muito grande.

O classificador naive Bayes faz o pressuposto ingênuo *"naive"* de que os atributos são condicionalmente independentes, ou seja, supõe que a probabilidade de observar o exemplo $x = (a_1, a_2, \ldots, a_{n_{atr}})$ é dada pelo produto das probabilidades de cada atributo individual: $P(a_1, a_2, \ldots a_{n_{atr}}|y_c) = \prod_i P(a_i|y_c)$, substituindo isso na Equação 3.2 é obtida a aproximação

usada pelo naive Bayes:

$$y_{NB} = \arg\max_{y_v \in Y} P(y_c) \prod_{i=1}^{n_{atr}} p(a_i|y_c)$$

O resultado da classificação do naive Bayes é idêntico à classificação y_{MAP} se os atributos forem considerados independentes. A complexidade do naive Bayes é muito menor que a do método $y_{MAP.}$ Pois o número de termos que devem ser calculados usando os exemplos de treinamento é dado apenas pelo número de atributos e os valores desses atributos multiplicado pelo número de classes.

Para exemplificar, considere um novo exemplo relativo a Tabela 3.5 onde é considerado o problema de jogar tênis:

$$(aparência = sol, temperatura = agradável, umidade = alta, ventando = verdadeiro)$$

O objetivo é predizer o valor da classe "sim" ou "não" para saber se é um bom dia para se jogar tênis. Para isso primeiramente os valores são instanciados na Equação 3.6.3:

$$\begin{aligned} y_{NB} &= \arg\max_{y_c \in Y} P(y_c) \prod_{i=1}^{n_{atr}} p(a_i|y_c) \\ y_{NB} &= \arg\max_{y_c \in Y} P(y_c) \times P(aparência = sol|y_c) \times P(temperatura = agradável|y_c) \\ &\quad \times P(umidade = alta|y_c) \times P(ventando = verdadeiro|y_c) \end{aligned}$$

Agora a probabilidade a priori pode ser estimada usando a frequência dos 14 exemplos da Tabela 3.5:

$$P(classe = sim) = \frac{9}{14} = 0.64$$

$$P(classe = não) = \frac{5}{14} = 0.36$$

As probabilidades condicionais $P(a_i|y_c)$ também podem ser estimadas, por exemplo, para o atributo *ventando* = "verdadeiro":

$$P(ventando = verdadeiro|\, jogar\ tênis = sim) = \frac{3}{9} = 0.33$$

$$P(ventando = verdadeiro|\, jogar\ tênis = não) = \frac{3}{5} = 0.66$$

Realizando-se o mesmo cálculo para os outros atributos as probabilidades resultantes podem ser utilizadas para estimar a classe do exemplo:

Como $0,041 > 0,007$, a classe do exemplo é "não", o que significa que esse não é um bom dia para se jogar tênis.

classe	*aparência*	*temperatura*	*umidade*	*ventando*
$P(sim)\times$	$P(sol\|sim)\times$	$P(agradável\|sim)\times$	$P(alta\|sim)\times$	$P(verdadeiro\|sim) =$
$\frac{9}{14}\times$	$\frac{2}{9}\times$	$\frac{4}{9}\times$	$\frac{3}{9}\times$	$\frac{3}{9} = 0,007$

classe	*aparência*	*temperatura*	*umidade*	*ventando*
$P(não)\times$	$P(sol\|não)\times$	$P(agradável\|não)\times$	$P(alta\|não)\times$	$P(verdadeiro\|não) =$
$\frac{5}{14}\times$	$\frac{3}{5}\times$	$\frac{2}{5}\times$	$\frac{4}{5}\times$	$\frac{3}{5} = 0,041$

Quando existem muitos atributos, o resultado da multiplicação desses valores se torna próximo de zero, o que pode causar problemas de precisão nos cálculos. Algumas implementações utilizam um espaço logarítmico para que a multiplicação se transforme em soma, e os valores não fiquem tão pequenos.

Em suma, o método de aprendizado utilizado pelo algoritmo naive Bayes estima várias probabilidades, usando as frequências observadas nos exemplos de treinamento. As probabilidades estimadas são utilizadas para o aprendizado de uma hipótese, que é usada para classificar novos exemplos. Apesar de o espaço de busca do naive Bayes pressupor que os atributos são independentes e com isso limitar seu espaço de busca, ele possui um desempenho relativamente bom na prática, mesmo quando existe dependência entre os atributos.

3.6.4 SVM (Support Vector Machines)

O algoritmo SVM (Máquinas de Vetores de Suporte) é um algoritmo de aprendizado de máquina baseado na teoria do aprendizado estatístico (TAE), introduzida em 1992. (Boser et al., 1992). Tornou-se popular por obter sucesso no reconhecimento de caracteres manuscritos (Bottou et al., 1994), obtendo uma taxa de erro semelhante a de uma rede neural cuidadosamente construída para essa tarefa. O algoritmo SVM busca a minimização do erro em relação ao conjunto de treinamento, também chamado de risco empírico, juntamente com a complexidade da função induzida, que está ligada a erros no conjunto de teste denominado risco estrutural. Surgiu da necessidade de desenvolver e formalizar limites teóricos para a capacidade de generalização dos sistemas de aprendizado de máquina, pois uma maior generalização, geralmente, implica um número maior de acertos na fase de teste, e não na fase de treino.

Muitos algoritmos, na tentativa de minimizar o erro empírico, criam uma superfície de decisão muito ajustada aos dados do conjunto de treinamento, aumentando as chances do superajuste e o risco estrutural. O objetivo primordial da SVM consiste na obtenção do equilíbrio entre ambos os erros, minimizando com isso, o superajuste e o subajuste, melhorando assim, a capacidade de generalização. A SVM é basicamente um separador linear que utiliza o conceito do hiperplano ótimo desenvolvido por Vapnik e Chervonenkis em 1965 (Vapnik e Chervonenkis, 1974). O hiperplano ótimo tem como característica possuir a margem de separação p maximizada. Essa margem de separação é medida a partir dos pontos mais próximos do hiperplano ótimo, chamados vetores de suporte. Os vetores de suporte são particularmente importantes

na SVM, pois são os pontos mais próximos à superfície de decisão e por isso, geralmente, de classificação mais difícil.

Na Figura 3.10 é ilustrado o hiperplano ótimo. A margem de separação p possui a mesma distância ao hiperplano, tanto do lado dos círculos quanto do lado dos quadrados e essa distância é máxima. Os vetores de suporte são os exemplos que são interceptados pela margem.

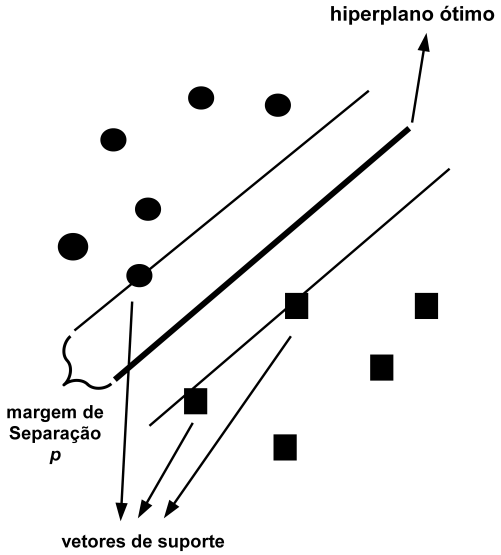


Figura 3.10: Hiperplano ótimo

Em suas primeiras versões, a SVM era chamada de SVM de margens rígidas, uma vez que só resolvia problemas linearmente separáveis, onde as classes não tivessem sobreposição, pois não permitia que exemplos ficassem entre as margens. Posteriormente, foi modificada para margens suaves (através da introdução de variáveis de folga), que permitiu a separação de classes sobrepostas, mas que fossem linearmente separáveis. O grande avanço da SVM veio com a introdução de funções kernel. Estas funções permitem o mapeamento dos dados não separáveis linearmente no espaço de atributos para o espaço característico, onde são linearmente separáveis. Na Figura 3.11 é ilustrado o conceito de kernel. Os exemplos da Figura 3.11.a. necessitam de uma fronteira não linear para separá-los (Figura 3.11.b). Contudo, se forem mapeados do espaço de atributos $\mathbb{R}^2$ para um espaço de características $\mathbb{R}^3$, como ilustrado na Figura 3.11.c, tornam-se linearmente separáveis.

A SVM apresenta vantagens em relação aos classificadores convencionais, principalmente quando o número de amostras de treinamento for pequeno e a dimensionalidade dos dados for grande, isso porque os classificadores convencionais não possuem mecanismos para maximizar a margem, e a maximização desta permite aumentar a capacidade de generalização do classificador (Abe, 2005).

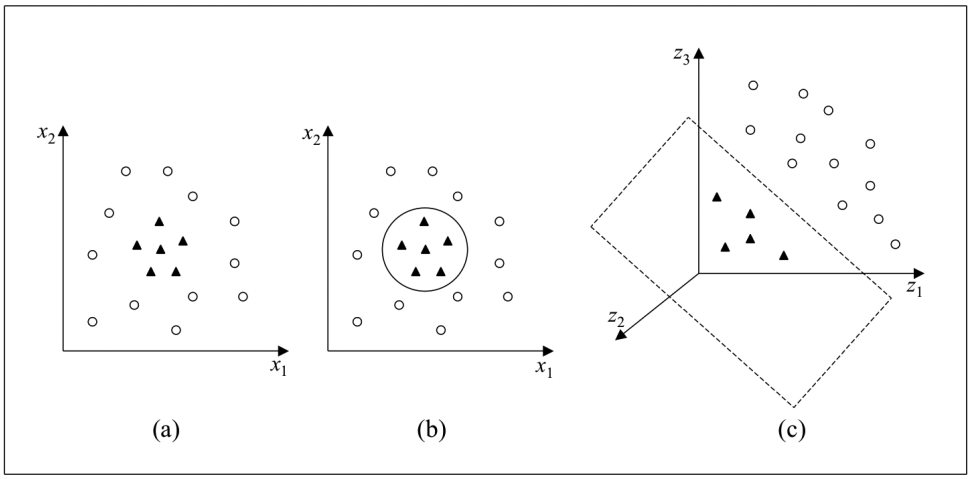


Figura 3.11: (a) Conjunto de exemplos não separável linearmente; (b) Fronteira não linear no espaço de entradas; (c) Fronteira linear no espaço de características (Lorena e de Carvalho, 2007)

3.7 Imputação Utilizando Aprendizado Semissupervisionado

Para utilizar o conjunto de imputação com a finalidade de induzir a hipótese, é proposto o modelo ilustrado na Figura 3.12, que utiliza aprendizado semissupervisionado. Neste modelo os dados não são divididos em treino e imputação. Todo o conjunto de dados com seus exemplos completos ou incompletos, que são equivalentes aos conjuntos vistos respectivamente como rotulados e não rotulados, servem de entrada a um algoritmo de aprendizado semissupervisionado. Assim, todos os exemplos são utilizados para gerar a hipótese que é usada para imputar os valores ausentes.

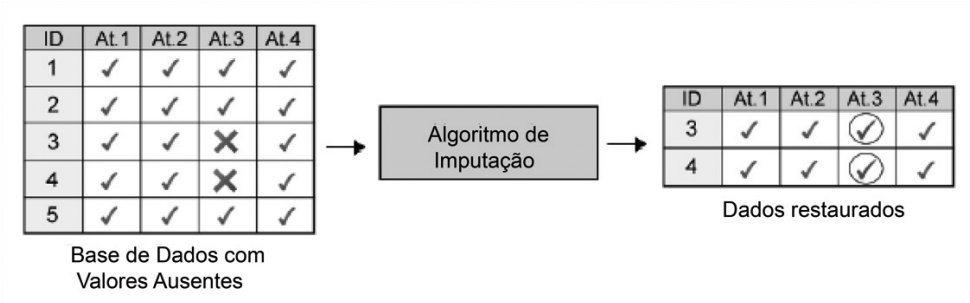


Figura 3.12: Imputação semissupervisonada (Castaneda et al., 2009) com adaptações

Espera-se que com a utilização dos dados não rotulados seja obtida uma melhora na qualidade dos dados imputados, pois a informação contida no conjunto de imputação é considerada durante a indução da hipótese.

A hipótese induzida desta maneira é fundamentada na Inferência Transdutiva que, como já mencionado no Capítulo 2, foi introduzida por Vapnik (1998), cuja idéia central é construir um classificador utilizando dois conjuntos de dados: um primeiro, em que as amostras já se encontram previamente rotuladas chamado de conjunto de treinamento; e um segundo em que as amostras não estão rotuladas, denominado conjunto de predição. O objetivo é justamente classificar os exemplos do conjunto de predição e isso é feito treinando o classificador com ambos

conjuntos de exemplos. Assim, os exemplos do conjunto de predição podem ser classificados em um único passo, o que diferencia do método indutivo tradicional, no qual dois passos devem ser executados, conforme ilustra a Figura 3.13.

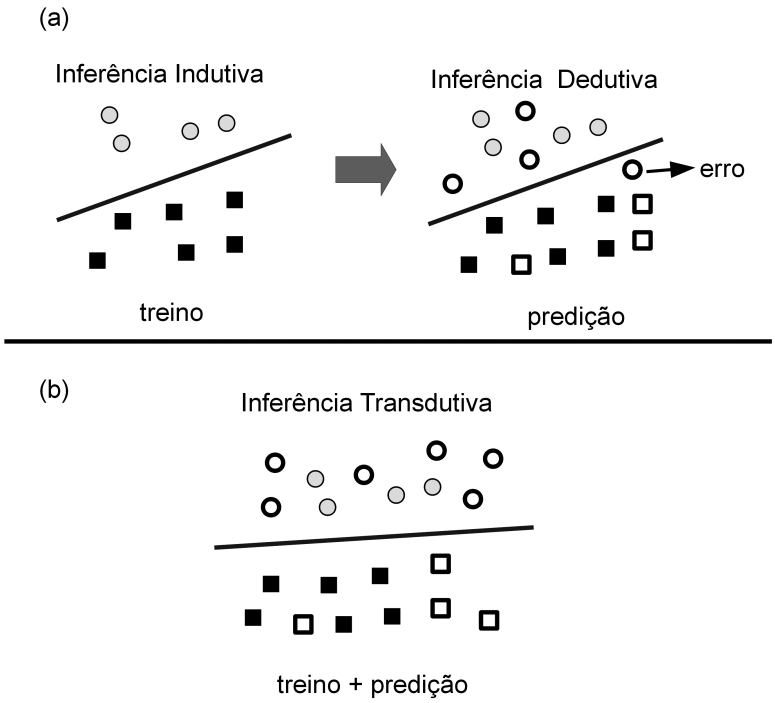


Figura 3.13: Aprendizado Indutivo versus Transdutivo

A principal vantagem com a utilização do método transdutivo é o aumento de informações disponíveis para o treinamento do algoritmo e com isso uma possível melhora na qualidade da imputação. No aprendizado indutivo tradicional, o passo indutivo visa encontrar a dependência funcional no conjunto de treino; o passo dedutivo utiliza essa dependência para classificar os exemplos de teste como ilustrado na Figura 3.13.a. Com isso, pode-se dizer que no método indutivo tradicional, o hiperplano é construído para separar as duas classes (bolas e quadrados), utilizando apenas o conjunto de treinamento. Posteriormente, num segundo passo, este hiperplano é utilizado para deduzir a classificação das amostras do conjunto predição que são adicionadas como bolas e quadrados vazados, uma bola é incorretamente classificada gerando um erro de classificação. Já na inferência transdutiva (Figura 3.13.b), o hiperplano é encontrado utilizando os dois conjuntos de dados, ou seja, treinamento e predição, observando-se que os dados de predição ainda não se encontram classificados. Ainda que, eventualmente, possam existir erros de classificação, agregando-se os dados do conjunto de predição ao treinamento do algoritmo, possívelmente deve-se obter um melhor desempenho. Quando se analisa o problema da imputação de dados, em conjuntos de dados estáticos (em que não se altera o número de exemplos), nota-se um problema de natureza intrinsecamente transdutiva, pois o objetivo é simplesmente completar a base de dados. Não é necessário a inferência de uma regra geral, apenas

de uma regra local que abranja o conjunto de treino e predição, pois não existirão novos exemplos para serem rotulados, somente os exemplos que já se encontram no conjunto de dados. Devido a isso, um dos classificadores escolhidos para a tarefa de imputação nesta monografia foi o TSVM (Trasdutive Support Vectors Machine) que será visto em detalhes.

3.7.1 TSVM (Transductive Support Vector Machines)

A TSVM (Máquinas de Vetores de Suporte Transdutiva) é uma versão da SVM que implementa a inferência transdutiva. Não difere muito da SVM a não ser que em seu treinamento além do conjunto de treino também é utilizado o conjunto de predição que contém os exemplos que se deseja rotular.

Na Teoria do Aprendizado Estatístico existe um limite denominado esquema de minimização do risco total estrutural (Vapnik, 1998), neste limite que se optou apenas por fazer citação, devido a sua complexidade. Vapnik demonstra que a inferência transdutiva obtém limitantes para o erro de predição melhores que os limitantes obtidos utilizando princípios da inferência indutiva e com isso é possível um método transdutivo obter uma capacidade de generalização maior que sua contraparte indutiva. A principal diferença na formulação da TSVM são os problemas de otimização dual e primal. Diferentemente da SVM, cujo problema de otimização é um problema de programação quadrática, o treinamento da TSVM conduz a um problema de otimização do tipo programação quadrática inteira mista (*mixed integer quadratic program*).

Geralmente, este problema não é convexo e o número de variáveis inteiras é proporcional ao número de exemplos do conjunto de predição. Assim, encontrar o ótimo global com os métodos de otimização padrão é possível somente para um número pequeno de amostras de predição. Porém, a solução exata, requer uma busca exaustiva sobre todas as combinações possíveis de classificação do conjunto de predição, e isso só pode na prática ser feito para umas poucas instâncias. No caso de um conjunto de predição maior, recomenda-se usar alguma heurística para encontrar uma solução satisfatória.

Conforme Vapnik (1998), a inferência transdutiva é um dos rumos de pesquisa mais promissores no desenvolvimento da teoria do aprendizado estatístico. Podendo assim, ter grande influência não apenas nas discussões técnicas dos métodos de generalização, mas também, no entendimento das formas de inferência adotadas. Diversos experimentos, principalmente com pequenos conjuntos de dados de treinamento, demonstraram a vantagem em se utilizar a inferência transdutiva, uma vez que houve uma significativa redução no número de erros de classificação no conjunto de predição (Demiriz e Bennett, 2001; Joachims, 1999b). Todavia, ele ainda esclarece que mesmo apesar disso, desde que a inferência transdutiva foi discutida em 1974 (Vapnik e Chervonenkis, 1974), poucos artigos foram publicados nessa área. Devido a esses fatos a inferência transdutiva ainda é uma aréa com muitas pesquisas pela frente, mas com uma perspectiva promissora.

3.7.2 TRI-TRAINING

O algoritmo TRI-TRAINING (Zhou e Li, 2005) é um algoritmo multivisão (*multiview*), inspirado no algoritmo CO-TRAINING (Blum e Mitchell, 1998) que foi utilizado com sucesso para imputação semissupervisionada (Matsubara et al., 2008). O algoritmo CO-TRAINING utiliza duas visões diferentes do conjunto de exemplos e dois classificadores para efetuar a classificação dos dados. A necessidade de duas visões é o inconveniente do algoritmo CO-TRAINING pois o conjunto de exemplos deve ser particionado em dois conjuntos que sejam suficientes para o aprendizado e condicionalmente independentes. É importante salientar que o termo multivisão atualmente não se refere somente a múltiplas visões dos dados, também é utilizado para se referir os paradigmas de aprendizado que utilizam o acordo entre diferentes classificadores. O algortimo TRI-TRAINING não necessita de múltiplas visões do conjunto de dados como o algoritmo CO-TRAINING, ele utiliza três classificadores induzidos através de amostragem bootstrap (Efron e Tibshirani, 1993). A amostragem é necessária para que os três classificadores sejam distintos, caso contrário o TRI-TRAINING degenera para autoaprendizado.

O idéia central do TRI-TRAINING consiste em induzir três classificadores h_1, h_2 e h_3 utilizando três diferentes amostras do conjunto de exemplos rotulados L. Isso é feito apenas no começo do algoritmo. Durante as iterações do algoritmo, esses três classificadores são induzidos da seguinte maneira: utilizando os classificadores h_2 e h_3 é construído um conjunto de exemplos rotulados para a indução de h_1. Da mesma maneira, para a construção do conjunto de exemplos rotulados para a indução de h_2, o algoritmo utiliza os outros dois classificadores h_1 e h_3 para rotular exemplos. Finalmente, para induzir h_3 é utilizado h_1 e h_2. Para não sobrecarregar a notação, são definidos um a, um b e um c, que podem assumir valores em $\{1,2,3\}$ tal que $b \neq a$ e $c \neq a$. Desse modo, se $a = 1$ então $b = 2$ e $c = 3$ ou $b = 3$ e $c = 2$. Utilizando a notação a, b e c, pode-se dizer que enquanto o algoritmo constrói o conjunto de exemplos rotulados para a indução de h_a, ele utiliza os outros dois classificadores h_b e h_c para rotular exemplos.

A idéia consiste em criar um conjunto de exemplos rotulados L_a, sempre utilizando exemplos que são rotulados com a mesma classe pelos classificadores h_b e h_c, para induzir o classificador h_a. Desse modo, é feito um rodízio para a construção dos conjuntos L_1, L_2 e L_3, denominada neste trabalho de *Parte 1*, e a indução dos classificadores h_1, h_2 e h_3, denominada *Parte 2*.

Durante as iterações do algoritmo é necessário ter algumas informações da iteração anterior. Para facilitar a compreensão do algoritmo considere todos os conjuntos e variáveis nomeadas com " $'$ " como o mesmo conjunto ou variável da iteração anterior, ou seja, L'_a representa o conjunto L_a da iteração anterior, a variável e'_a é a variável e_a da iteração anterior, e assim por diante.

- *Parte 1*: o objetivo da primeira parte é construir um conjunto de exemplos rotulados L_a que possua um número de exemplos rotulados errados menor que L'_a. Assim, para cada $a \in \{1,2,3\}$ o algoritmo executa os seguintes passos:

  1. cálculo de e_a: o conjunto L é rotulado pelos classificadores h_b e h_c e os exemplos $\mathbf{x}_i$

rotulados com a mesma classe, *i.e.*, $h_b(\mathbf{x}_i) = h_c(\mathbf{x})$ são adicionados ao conjunto R_a. Como o conjunto L é rotulado, é possível computar a porcentagem de exemplos rotulados errados em R_a. Essa porcentagem é denominada de e_a. Se essa porcentagem for menor que e'_a (e_a da iteração anterior) o algoritmo vai para o próximo passo.

2. criar o conjunto L_a: o conjunto de exemplos não rotulados U também é submetido aos classificadores h_b e h_c e os exemplos $\mathbf{x}_i$ rotulados com a mesma classe são adicionados ao conjunto L_a. Se $|L'_a| < |L_a|$ o algoritmo vai para o próximo passo. Vale ressaltar que a cada iteração é construído um L_a totalmente novo, não sendo acumulados os exemplos da iteração anterior.

3. verificar a *melhoria*: mesmo que $e_a < e'_a$ e $|L_a| < |L'_a|$ pode acontecer que $e_a.|L_a| < e'_a.|L'_a|$ não seja verdade. Assim, o algoritmo não tem controle direto sobre e_a, mas pode controlar o tamanho do conjunto L_a. Desse modo, caso $e_a.|L_a| < e'_a.|L'_a|$ então é atribuído verdadeiro para a variável $atualizar_a$, caso contrário, se e_a é significativamente menor que e'_a, pode-se construir um L_a menor tal que $e_a|L_a| < e'_a|L'_a|$ seja verdadeiro, caso essa construção for possível, $atualizar_a$ recebe verdadeiro e esse subconjunto de L_a é construído, caso contrário $atualizar_a$ recebe falso.

- *Parte 2*: para cada $a \in \{1,2,3\}$, caso a variável $atualiza_a$ for verdadeiro, induzir um h_a utilizando o $L \cup L_a$

De modo resumido, o algoritmo consiste em construir um L_a utilizando os classificadores h_b e h_c. Quando é garantido que $e_a|L_a| < e'_a|L'_a|$ então é atribuído verdadeiro a variável $atualizar_a$, que, por sua vez, aciona o algoritmo de aprendizado para induzir h_a utilizando $L \cup L_a$.

Como mencionado, a construção do L_a é feita utilizando os outros dois classificadores, h_b e h_c. Essa abordagem é semelhante ao utilizada por CO-EM, no qual o conjunto de treinamento de um classificador é construído utilizando os rótulos obtidos pelo outro classificador. Desse modo, o algoritmo herda algumas características do CO-EM como a iteratividade sob o conjunto de exemplos não rotulados.

O cálculo de e_a é de grande importância para o algoritmo decidir se ele vai ou não utilizar L_a para induzir h_a. Entretanto, e_a é calculado utilizando apenas o conjunto inicial de exemplos rotulados, que normalmente é muito pequeno, e que em aplicações de aprendizado semisupervisionado pode conter menos que 10 exemplos. TRI-TRAINING precisa que esse conjunto não seja muito pequeno, pois caso a estimativa de e_a não for boa, o algoritmo não gera um bom classificador.

3.7.3 SSLVote

O algoritmo SSLVote é um algoritmo semisupervisionado que utiliza aprendizado multivisão. Para selecionar e rotular os exemplos não rotulados, o SSLVote realiza uma votação ponderada entre os n classificadores que o constituem, para cada exemplo não rotulado é calculado um *score*, se o *score* do exemplo atingir um limiar predefinido o exemplo é adicionado

ao conjunto de treino. Na escolha do classificador final é calculado um *score* de acordo com o número de exemplos rotulados que foram adicionados ao conjunto de treino. O algoritmo do SSLVote segue os seguintes passos:

1. No inicio cada classificador é treinado com os exemplos rotulados do conjunto de treino e através de uma validação cruzada obtêm-se o valor da AUC de cada um desses classificadores no conjunto de treino denominado de confiança do classificador (cc).

2. Cada um dos n classificadores rotula um exemplo não rotulado. Para um dado classificador $h \in n$ é calculado uma razão de igualdade (ri) que consiste do número de classificadores que rotularam o exemplo com o mesmo rotulo do classificador h dividido por n. Para cada classificador além da predição também é utilizada a força de classificação do rotulo (fc), o score final (sf) do exemplo é dado por $sf = cc + fc + ri/3$ que resulta em um valor entre 0 e 1. Se o valor do *score* for superior a um limite predefinido o exemplo é adicionado ao conjunto de rotulados e o score do classificador é incrementado em uma unidade, caso contrário o exemplo é descartado.

3. O passo 2 é repetido para cada exemplo até o fim do conjunto de dados não rotulado. Ao final o classificador com maior *score* é treinado com o conjunto de treino original mais o novo conjunto de dados rotulados.

Este classificador tenta minimizar o impacto negativo dos dados não rotulados usando a votação ponderada, mas não é um exemplo de classificador semissupervisionado "seguro", pois pode ter seu desempenho degradado por exemplos erroneamente rotulados que forem adicionados ao conjunto de treino.

Nesse Capítulo foram apresentados os conceitos de valores ausentes e os principais algoritmos utilizados em imputação. No próximo, são apresentados os experimentos de imputação de valores ausentes utilizando aprendizado semissupervisionado.

CAPÍTULO

4

Análise Experimental

Neste capítulo são apresentados os experimentos de imputação semissupervisionada. Na Seção 4.1 são apresentadas as técnicas e algoritmos utilizados nos experimentos, na Seção 4.2 são descritas as bases de dados utilizadas, na Seção 4.3 é descrita a metodologia usada para criar os valores ausentes artificiais que são usados nos experimentos. Na Seção 4.4 é descrito o procedimento experimental bem como os experimentos e a análise dos resultados, na Seção 4.5 são feitas as considerações finais dos resultados dos experimentos.

4.1 Técnicas e Algoritmos

Para realização dos experimentos foram necessários utilizar vários algoritmos e ferramentas, sendo que alguns precisaram ser adaptados ou construídos. Esta Seção descreve os softwares utilizados, construídos ou modificados para a realização dos experimentos contidos neste Capítulo.

O pacote de software Weka (Ambiente Waikaito para Análise de Conhecimento) é uma ferramenta de Mineração de Dados de código fonte aberto, distribuído sobre licença GPL[1] (*General Public License*). Seu desenvolvimento se iniciou em 1993, na Universidade de Waikato, na Nova Zelândia, usando a linguagem de programação Java e desde então tornou-se uma ferramenta amplamente utilizada no meio acâdemico. Sua escolha foi motivada pela facilidade de uso devido a interface gráfica com o usuário e por ser implementado em linguagem Java, permitindo a portabilidade entre os diversos sistemas operacionais.

A ferramenta Weka é formada por um conjunto de pacotes que realizam as seguintes funções: *attribute selection* (seleção de atributos de uma base de dados), *clustering* (implementação de algoritmos de aprendizado não supervisionado), *associate* (regras de associação), *classifiers*

[1]GPL: GNU *General Public License*. para mais informações: http://www.gnu.org/copyleft/gpl.html.

(implementação de algoritmos de aprendizado supervisionado), *filters* (processamento de instâncias ou atributos de uma base) e *visualize* (para visualização de dados). Além disso, possui aplicações para facilitar a execução de experimentos e processamento dos dados.

Para este trabalho, foram selecionados quatro algoritmos de classificação do Weka escolhidos por utilizarem paradigmas de aprendizado diferentes, cada um desses algoritmos possui uma forma de extrair conhecimento da base de dados, esse fato pode posteriormente ajudar na análise dos resultados. Os algoritmos selecionados foram : J48, IBK (k-NN), naive Bayes e SMO (SVM), cuja breve descrição se faz abaixo e uma descrição detalhada de cada algoritmo pode ser encontrada no Capítulo 3.6:

J48: O algoritmo J48 é uma implementação do algoritmo C4.5 release 8 que, gera árvores de decisão. Ele constrói um modelo de árvore de decisão utilizado para classificar as instâncias dos conjuntos de teste, baseando-se num conjunto de dados de treinamento. O J48 foi utilizado com os valores padrões do Weka.

IBK: Uma implementação do popular algoritmo dos k-vizinhos mais próximos (k-NN). Permite o uso de várias métricas de distância, distância ponderada, cross-validação para encontrar o valor ótimo de k, além de outras funcionalidades. O k-NN teve seu parâmetro k fixado em 10 conforme o recomendado na literatura (Troyanskaya et al., 2001).

Naive Bayes: Implementação do algoritmo naive Bayes baseada em (John e Langley, 1995), com discretização não supervisionada e estimadores usando *Kernel*. O naive Bayes foi utilizado com os valores padrões do Weka.

SMO: Implementação da SVM utilizando o método de otimização sequencial mínima (Platt, 1999), é capaz de trabalhar com múltiplas classes ao contrário de muitas implementações da SVM. A SVM utilizou *kernel* polinomial de grau 3, além do grau do kernel foi alterado também o parâmetro C.

O Weka não possui nativamente algoritmos de aprendizado transdutivos ou semissupervisionados que são o foco deste trabalho. Devido a isso, foi necessário a confecção ou modificação de classificadores do Weka que são descritos a seguir:

TSVM: O SVM^{light} é uma implementação do SVM para solução de problemas como classificação, regressão e ranking. Os algoritmos de otimização utilizados no SVM^{light} (Joachims, 2002, 1999a) são escaláveis em requerimento de memória, possuem estratégias para redução do número de variáveis e podem manipular eficientemente milhares de vetores de suporte. SVM^{light} já foi utilizada numa ampla gama de problemas que incluem classificação de textos (Joachims, 1999c, 1998), reconhecimento de imagem (Osuna et al., 1997) e bioinformática (Yue et al., 2007). Sua escolha foi motivada por possuir uma implementação da TSVM e interfaces para várias linguagens de programação, entre elas, a linguagem Java que é a linguagem utilizada no Weka. Foi construído um classificador *wrapper* para o Weka, para dar suporte a TSVM do SVM^{light}, este novo classificador funciona tanto

na interface gráfica do Weka, como ilustrado na Figura 4.1, quanto em modo linha de comando. Para sua confecção foi utilizada a interface nativa Java criada por Martin Theobald[2]. O *wrapper* permite acesso a todos os parâmetros de configuração do SVM^{light} e funciona tanto em modo supervisionado como semissupervisionado.

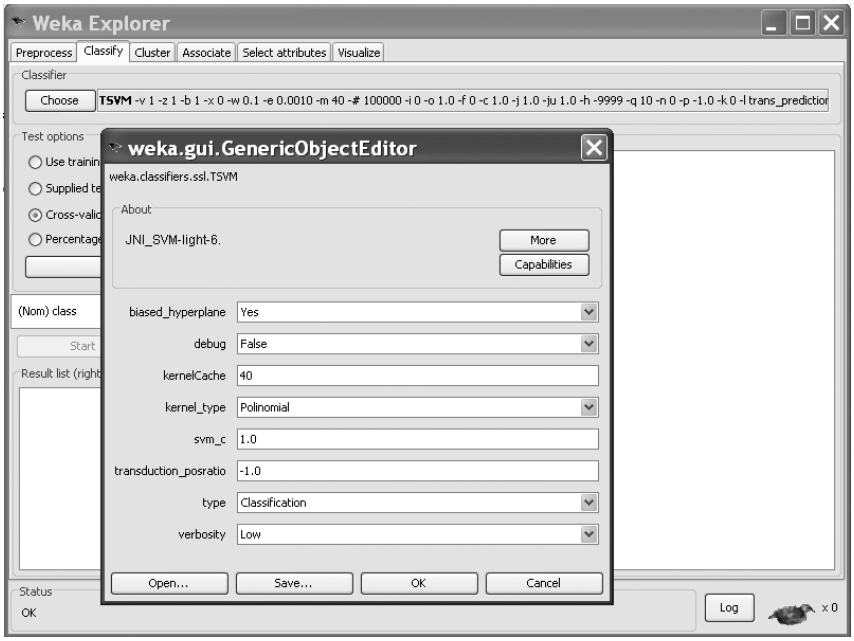


Figura 4.1: Interface gráfica do classificador TSVM no Weka

SSMultiClassClassifier: O TSVM somente trabalha com classes binárias, por isso é necessário utilizar um metaclassificador (classificador *wrapper* que funciona utilizando-se de outros classificadores do Weka) denominado MultiClassClassifier, que permite que classificadores binários trabalhem com múltiplas classes utilizando matrizes ECOC (Dietterich e Bakiri, 1995). O MultiClassClassifier não dá suporte ao aprendizado semisupervisionado. Por isso, foi necessário compreender e modificar seu código fonte, cujo resultado foi o novo metaclassificador SSMultiClassClassifier que funciona no ambiente gráfico do Weka (Figura 4.2) ou modo linha de comando. O SSMultiClassClassifier suporta tanto o modo supervisionado quanto semissupervisionado.

SSLTriTrain-AS: A partir da implementação original do algoritmo TRI-TRAINING foi criada uma versão capaz de selecionar automaticamente o classificador base do algoritmo utilizando validação cruzada no conjunto de treino com o objetivo de selecionar o melhor classificador, entre J48, NB, IBk e SMO, para o conjunto de teste. Para cada um dos classificadores supervisionados utilizados neste experimento, o TriTrain-AS realiza uma validação cruzada em 10 partições no conjunto de treino e seleciona o classificador que resultar na maior AUC para utilizar como classificador base.

SSLVote: O SSLVote é um classificador multi-descrição que utiliza todos os classificadores supervisionados utilizados neste experimento. O treinamento é dividido em duas fases: na

[2]http://www.mpi-inf.mpg.de/mtb/

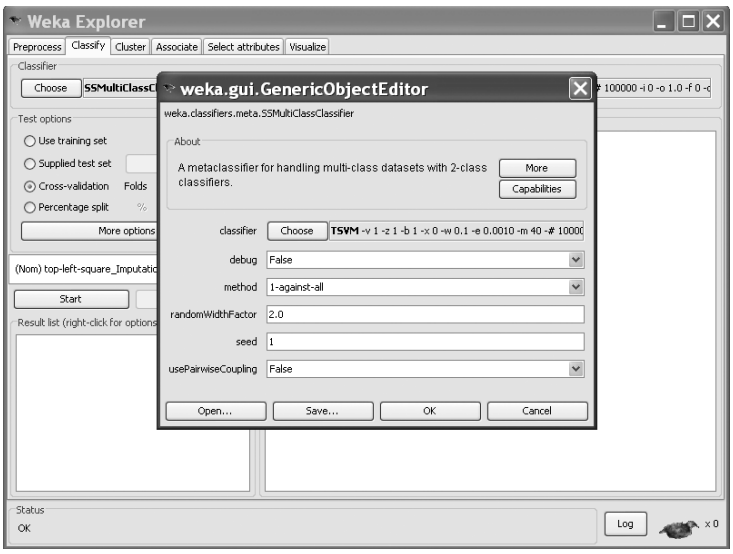


Figura 4.2: Interface gráfica do metaclassificador SSMultiClassClassifier no Weka

primeira é utilizado autoaprendizado para que todos os classificadores rotulem os exemplos não rotulados através de uma votação ponderada, apenas os exemplos que alcançam um determinado *score* na votação são adicionados ao conjunto de treino original. Na segunda fase um classificador é selecionado utilizando validação cruzada de 10 partições no conjunto de treino original e posteriormente treinado com o novo conjunto de treino que inclui os dados não rotulados selecionados e rotulados através da votação.

Também foi criado uma aplicação em linha de comando em Java utilizando os classificadores do Weka para automatizar os testes com as bases de dados e os classificadores.

4.2 Bases de Dados

Para realizar os experimentos de imputação foram selecionadas quatro bases de dados do repositório da UCI (Frank e Asuncion, 2010). O critério de seleção das bases foi escolher bases que contivessem apenas atributos nominais ou binários, para facilitar a geração dos valores ausentes e permitir a utilização de classificadores semissupervisionados pois geralmente apenas trabalham com atributos de classe binários ou nominais. Foram selecionadas as seguintes bases:

primary-tumor
: Esta base de dados contém 339 instâncias com 18 atributos nominais. As informações consistem em dados sobre tumores primários que foram obtidas no Centro Médico Universitário e no Instituto de Oncologia de Ljublja, na Yugoslavia. O atributo classe possui 21 valores distintos, referente a localização do tumor no corpo do paciente. Contém 207 instâncias com valores ausentes, restando 132 instâncias após a remoção das instâncias com atributos ausentes.

soybean A base contém dados sobre doenças que atacam os grãos de soja. Possui 683 instâncias com 36 atributos nominais. O atributo classe possui 19 valores distintos, cada um é um

tipo de doença que ataca os grãos de soja. Essa base possui 121 instâncias com valores ausentes, restando 562 instâncias após a remoção das instâncias com atributos ausentes.

tic-tac-toe Consiste em finais de jogo-da-velha, representando todas as possibilidades de configurações entre *X* (xis), *O* (bola) e *B* (em branco) utilizados em um jogo-da-velha, onde assume-se que *X* joga primeiro e o objetivo é a sua vitória. A base é composta de 958 instâncias com 9 atributos nominais, sendo que o atributo classe é binário. Não possui instâncias com valores ausentes.

solar-flare 2 Base de dados sobre explosões solares. Possui 1066 instâncias, com 13 atributos nominais e contém três potênciais atributos classe. Foi utilizado o atributo "X-class-_flares_production_by_this_region" como atributo classe por ser o último atributo. Este atributo possui três valores distintos que indicam o número de vezes que ocorreram explosões solares do tipo *X* nas últimas 24h. Não possui instâncias com valores ausentes.

Na Tabela 4.1 é ilustrada de forma resumida as características das bases de dados utilizadas: **#exemplos:** o número de exemplos existentes na base de dados; **#ausentes:** é o número de exemplos que possuem valores ausentes; **#sem ausentes:** número de exemplos após a remoção dos exemplos com valores ausentes; **#atributos:** número de atributos; e **#distintos classe:** número de valores distintos do atributo classe.

Tabela 4.1: Descrição das bases de dados utilizadas

base	#exemplos	#ausentes	#sem ausentes	#atributos	#distintos classe
primary-tumor	339	207	132	18	21
soybean	683	121	562	36	19
tic-tac-toe	958	0	958	9	2
solar-flare 2	1066	0	1066	13	3

Para realização dos experimento as instâncias com atributos ausentes foram removidas, restando apenas as instâncias completas. Esse procedimento foi adotado para se obter controle sobre os valores ausentes nas bases de dados. Os valores ausentes já existentes nas bases são devidos a mecanismos de ausência desconhecidos o que prejudicaria os experimentos.

4.3 Simulação de Valores Ausentes

O mecanismo de ausência MCAR, conforme discutido no Capítulo 3, consiste na de valores ausentes aleatóriamente, sua escolha se deve a maior facilidade na geração dos valores ausentes. Para realizar a simulacão de valores ausentes foi utilizado um procedimento de selecão de atributos, com o intuito de simular valores ausentes, de forma controlada, em atributos que são significativos.

A Metodologia adotada para o mecânismo MCAR, baseia-se na inserção controlada de valores ausentes nos atributos. Esses atributos são os atributos geralmente utilizados pelos al-

goritmos de aprendizado para a construção de hipótese. Dessa forma, é possível verificar o comportamento dos algoritmos de imputacão no processo de aprendizagem do classificador.

Para escolha do atributo foi utilizado a estratégia *wrapper* do avaliador de atributos *WrapperSubSetEval* (Kohavi e John, 1997) do software Weka com o método de busca *BestFirst.* O classificador escolhido foi o SMO com kernel polinomial de grau 3 e parâmetro C=1,0. A medida de avaliação escolhida no uso do *WrapperSubSetEval* foi a AUC (*Area Under the ROC Curve*) (Fawcett, 2006). Essa medida foi utilizada por apresentar melhores resultados na presença de classes desbalanceadas (classes com grandes diferenças na proporção do número de exemplos), que pode ser muito comum em imputação, já que a imputação pode ocorrer sobre um atributo qualquer. Quanto mais próximo de 1.0 (um) estiver o valor da AUC, melhor é a qualidade da predição.

Na seleção de atributos foi utilizada uma validação cruzada em 10 partições. Os atributos selecionados pela estratégia *wrapper* em cada base de dados foram os seguintes:

Tabela 4.2: Atributos selecionados

base	atributo	rotulo	#distintos
primary-tumor	2	sex	2
soybean	1	date	7
tic-tac-toe	1	top-left-square	3
solar-flare 2	1	class	6

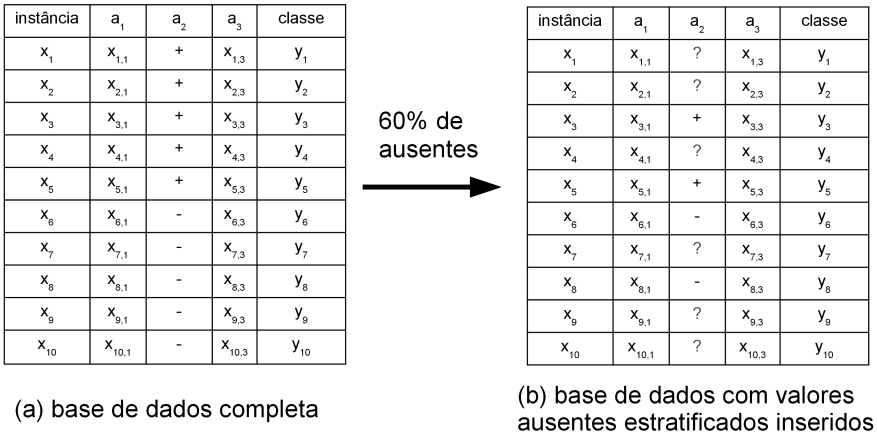
O procedimento de simulação dos valores ausentes com mecanismo MCAR foi realizado da seguinte forma:

1. Selecionar os atributos utilizando o avaliador de atributos WrapperSubSetEval com o algoritmo de classificação SMO, utilizando validação cruzada em 10 partições.

2. Ordenar cada atributo selecionado pela relevância, de forma que o atributo mais frequentemente selecionado em 10 partições seja o atributo mais relevante.

3. Selecionar aleatoriamente instâncias da base de dados e em seguida gerar valores ausentes no atributo selecionado anteriormente de forma estratificada, ou seja, os valores ausentes são gerados de forma a manter a proporção das classes originais no atributo a ser imputado.

A geração estratificada dos valores ausentes é ilustrada na Figura 4.3. Na base de dados do exemplo, o atributo a_2 foi selecionado como atributo mais relevante. Os valores ausentes são gerados aleatoriamente, mantendo-se todavia, as proporções entre as classes. Nesta Figura o atributo a_2 é binário e os 10 exemplos estão divididos em 50% para cada classe. Para introduzir 60% de valores ausentes, 6 exemplos devem ser marcados como ausentes. Para manter a proporção de 50% positivo e 50 % negativo, introduz-se 3 exemplos positivos e 3 negativos. Isso é feito para impedir que em bases com classes desbalanceadas ou minoritárias ocorra de todos os exemplos de uma determinada classe serem transformados em atributos ausentes.



(a) base de dados completa

(b) base de dados com valores ausentes estratificados inseridos

Figura 4.3: Geração de valores ausentes.

4.4 Avaliação Experimental

A validação cruzada é utilizada para algoritmos indutivos e deste modo o conjunto de treino e teste é sempre disjunto e a avalição é sobre a qualidade preditiva no conjunto de teste. Neste trabalho o problema de imputação é considerado um problema de transdução. Nesse caso, a qualidade preditiva é sobre o conjunto de exemplos não rotulados fornecidos durante o treinamento. Note que deste modo o treino e teste não são disjuntos.

Neste trabalho é proposto a utilização uma metodologia diferente por considerar que a tarefa de imputação é de natureza intrinsecamente transdutiva e o classificador apenas necessita rotular os exemplos com atributos ausentes (denominado aqui neste trabalho de conjunto de predição). Assim, não é necessário um classificador seja genérico para prever exemplos não vistos durante o treinamento. Desse modo, optou-se por utilizar um conjunto de treino (atributos completos) e predição (atributos com valores ausentes) no lugar da validação cruzada. Todos os resultados reportados são avaliados utilizando AUC. AUC como já explanado, evita problemas de utilização de classificadores com limiares de decisão na posição incorreta além de ser mais robusto para problemas de classes desbalanceadas.

Para realização dos experimentos foi implementada uma aplicação em Java, utilizando classes e métodos a biblioteca do Weka para realizar os experimentos de imputação. Essa aplicação realiza a geração dos valores ausentes conforme descrito na Seção 4.3, o treinamento dos classificadores tanto supervisionados, quanto semissupervisionados, testa predição dos classsificadores, controla o número de iterações do experimento e gera as médias e desvios padrões dos resultados em AUC.

A avaliação experimental segue o seguinte procedimento:

1. No primeiro laço, os percentuais de valores ausentes (20%, 40%, 60%, 80%, 90%, 95%, 98%) são artificialmente inseridos no conjunto de dados, gerando um conjunto de dados com valores ausentes do tipo MAR como descrito na Seção 4.3 na página 55 ;

2. Posteriormente, esse conjunto é dividido em duas partes: uma para o conjunto de treino (66%) e outra para o conjunto de predição (33%);

3. Cada um dos classificadores supervisionados descritos na Seção 4.1, são treinados usando apenas o conjunto de treino e avaliado utilizando o conjunto de predição. Os classificadores semissupervisionado o conjunto de treinamento consiste no conjunto de treino (rotulados) e predição (não rotulados) e são avaliados utilizando o conjunto de predição;

Esse procedimento foi repetido 30 vezes produzindo como resultado a média e o desvio padrão das AUC.

Além desses passos, para que o processo de imputação seja realizado é necessário que após a geração de valores ausentes, o atributo com valores ausentes se torne o atributo classe, pois os classificadores só predizem sobre esse atributo. Na Figura 4.4 é ilustrada a geração dos conjuntos de treino e teste a partir da base de dados com os valores ausentes já inseridos e o processo de treino e imputação. A base da dados de treino (Figura 4.4.a) é dividida em dois conjuntos de dados. O de treino como mostrado na Figura 4.4.b, que possui apenas exemplos completos (rotulados), e o de predição (Figura 4.4.c), que possui os exemplos não rotulados. Em ambos os conjuntos, o atributo a ser imputado é movido para o lugar do atributo classe e este é transformado em um atributo comum.

O procedimento de treino do indutor é ilustrado na Figura 4.4.d, cujo indutor recebe o conjunto de treino e induz um classificador (Figura 4.4.e) que será usado no processo de imputação para predizer o valor dos atributos ausentes. Após essa etapa, os dados são usados para restaurar a base de dados original. Para o aprendizado semissupervisionado, o processo é o ilustrado na Figura 4.5, sendo a única diferença no treinamento (Figura 4.5.d), onde o indutor utiliza os dados de treino e os de predição.

O objetivo dos experimentos foi verificar se o aprendizado semissupervisionado pode ter um desempenho melhor que o supervisionado na tarefa de preenchimento de valores ausentes. Os experimentos utilizam a metodologia mencionada na Seção 4.4. Os classificadores usaram os parâmetros citados na Seção 4.1. SMO e TSVM são sensíveis a escolha do parâmetro C. Assim foi realizada para cada base de dados uma validação cruzada de 10 partições utilizando os valores de $C = \{0.01, 0.1, 1.0, 10\}$, para cada um dos algoritmos o valor de C que apresentou o melhor resultado é utilizado nos experimentos.

De maneira mais específica, a avaliação experimental foi conduzida no intuito de responder as seguintes perguntas:

1. *Em problemas com grande quantidade de dados rotulados, os algoritmos de aprendizado supervisinado tem melhor desempenho que os semissupervisionados?*

2. *Independente da proporção de exemplos rotulados e não rotulados, qual possui melhor desempenho, algoritmos supervisionados ou semissupervisionados?*

3. *A literatura reporta um bom desempenho de semissupervisionado principalmente em bases de textos e em situações onde a quantidade de exemplos não rotulados é muito maior*

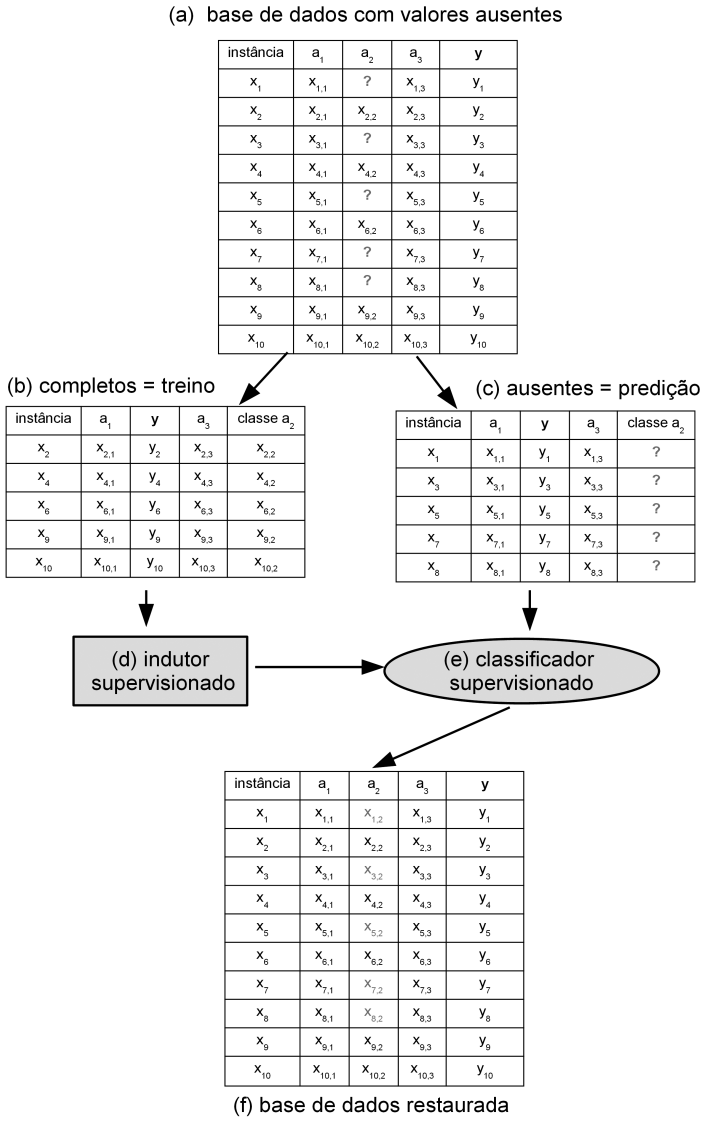


Figura 4.4: Processo de imputação utilizando classificador supervisionado

que a quantidade de exemplos rotulados. Como é o comportamento do semissupervisionado em problemas de imputação com a característica de grande desproporção de não rotulados e rotulados?

4.4.1 Avaliação Experimental 1

Nesta primeira avaliação experimental foram conduzidos experimentos com o objetivo de responder a seguinte pergunta:

Em problemas com grande quantidade de dados rotulados, os algoritmos de aprendizado supervisionado tem melhor desempenho que os semissupervisionados?

Como essa grande quantidade de exemplos rotulados pode variar de problema a problema, na avaliação experimental foram inseridos artificialmente 20%, 40% ,60%, 80%, 90%, 95% e

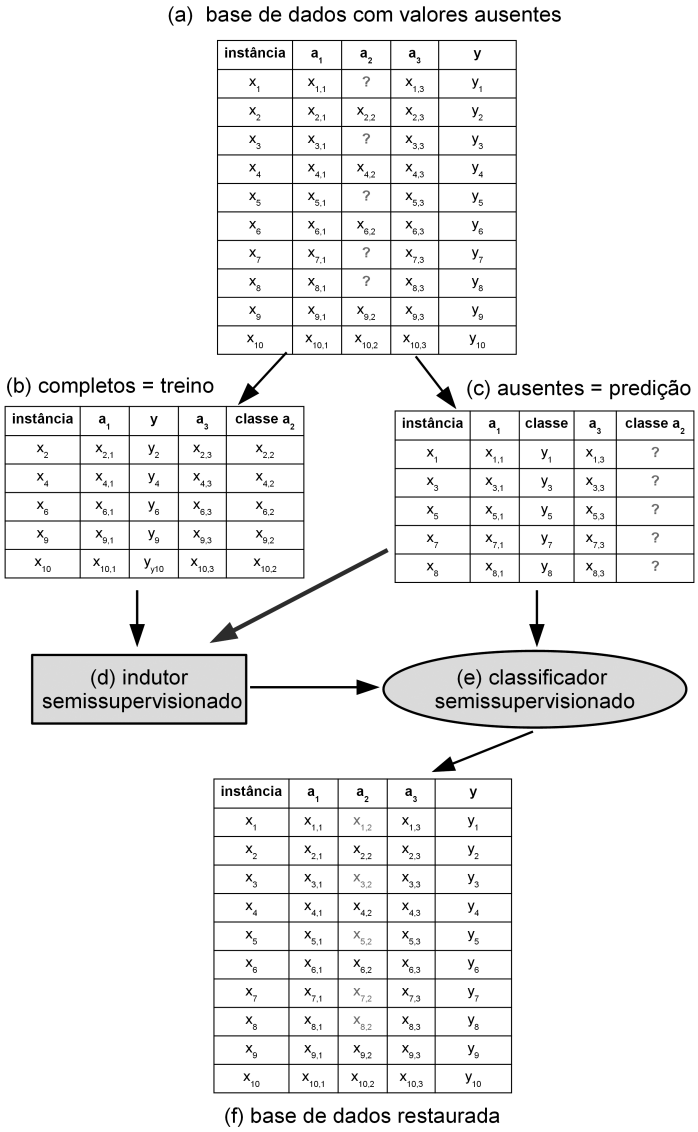


Figura 4.5: Processo de imputação utilizando classificador semissupervisionado

98% de exemplos não rotulados no atributo a ser imputado. O classificador TRI-TRAINING foi executado com o algoritmo supervisionado que apresentou o melhor resultado como classificador base. Os resultados dos classificadores foram avalidados utilizando AUC e os valores apresentados são os valores obtidos da média e desvio padrão de 30 execuções. Toda vez que o texto mencionar *diferença estatística significativa* significa que foi utilizado o teste t-pareado com um intervalo de confiança de 95% e que houve diferença segundo o teste (rejeição de hipótese nula).

primary-tumor: para a base de dados *primary-tumor* foram obtidos os resultados descritos na Tabela 4.3, cujos melhores valores de cada percentual de ausentes estão em negrito.

Os valores de 95% e 98% desta base não foram calculados para esta base pois com 90% de ausência restaram apenas 13 exemplos rotulados. Os melhores valores de C para o SMO e TSVM são respectivamente 0.01 e 0.1.

Tabela 4.3: Resultados da base *primary-tumor*

Algoritmo	20%	40%	60%	80%	90%
SMO	0.634 (0.100)	0.631 (0.056)	0.618 (0.048)	0.613 (0.051)	0.577 (0.054)
J48	0.695 (0.107)	0.651 (0.069)	0.633 (0.052)	0.617 (0.052)	0.552 (0.057)
NB	0.765 (0.086)	0.742 (0.054)	0.721 (0.034)	0.703 (0.042)	0.643 (0.054)
10-NN	0.715 (0.083)	0.697 (0.053)	0.680 (0.038)	0.646 (0.058)	0.545 (0.075)
SSLVote	**0.768 (0.087)**	0.733 (0.061)	0.683 (0.043)	0.653 (0.044)	0.608 (0.061)
TriTrain-NB	0.766 (0.087)	**0.743 (0.055)**	**0.723 (0.034)**	0.701 (0.044)	**0.654 (0.060)**
TriTrain-AS	0.766 (0.086)	0.725 (0.067)	0.712 (0.043)	**0.712 (0.043)**	0.611 (0.070)
TSVM-0.1	0.691 (0.094)	0.697 (0.061)	0.692 (0.048)	0.681 (0.062)	—

Na Figura 4.6 pode-se observar que o classificador supervisionado que obteve a melhor medida de AUC foi o naive Bayes. Os classificadores SSLVote, TriTrain-NB e TriTrain-AS não apresentaram diferenças estatisticamente significativas utilizando teste t-pareado em relação ao naive Bayes, ou seja, os dados não rotulados não melhoraram o desempenho dos classificadores semissupervisionados. O TriTrain-AS que possui autoseleção do classificador base conseguiu manter seu desempenho em todos os percentuais de ausência sem diferenças significativas em relação ao melhor classificador supervisionado. O TSVM apresentou desempenho inferior ao naive Bayes com diferença estatisticamente significativa em relação a esse, com 20% a 60% de dados ausentes, sendo que a partir de 80% a diferença não é mais significativa. TSVM não terminou a execução com 90% no prazo que os experimentos executaram e por esse motivo o TSVM não é representado no grático desta base.

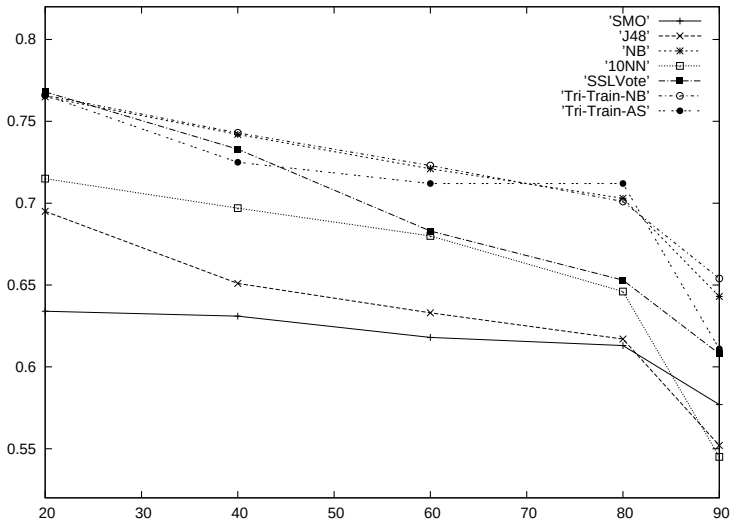


Figura 4.6: Resultados da base *primary-tumor*

solar-flare 2: na base *solar-flare 2*, o classificador supervisionado naive Bayes reportou melhor

desempenho em termos de AUC como ilustrado na Tabela 4.4.

Tabela 4.4: Resultados da base *solar-flare 2*

Algoritmo	20%	40%	60%	80%	90%	95%	98%
SMO	0.891 (0.009)	0.891 (0.007)	0.887 (0.007)	0.881 (0.010)	0.876 (0.010)	0.866 (0.011)	0.828 (0.024)
J48	0.922 (0.007)	0.918 (0.009)	0.912 (0.010)	0.901 (0.010)	0.888 (0.013)	0.877 (0.020)	0.843 (0.032
NB	**0.939 (0.007)**	**0.939 (0.005)**	**0.937 (0.003)**	**0.934 (0.003)**	**0.927 (0.005)**	**0.919 (0.007)**	**0.893 (0.018)**
10-NN	0.929 (0.007)	0.927 (0.006)	0.924 (0.004)	0.918 (0.004)	0.901 (0.007)	0.879 (0.010)	0.804 (0.023)
SSLVote	0.935 (0.008)	0.932 (0.006)	0.928 (0.005)	0.921 (0.008)	0.911 (0.011)	0.893 (0.020)	0.845 (0.018)
TriTrain-NB	**0.939 (0.007)**	**0.939 (0.005)**	**0.937 (0.003)**	0.933 (0.003)	**0.927 (0.005)**	**0.919 (0.007)**	0.890 (0.021)
TriTrain-AS	**0.939 (0.007)**	**0.939 (0.005)**	**0.937 (0.003)**	0.933 (0.003)	**0.927 (0.005)**	0.915 (0.014)	0.875 (0.029)
TSVM	0.901 (0.012)	0.903 (0.013)	0.867 (0.012)	—	—	—	0.525 (0.029)

O classificador semissupervisionado TriTrain-NB apresentou resultados semelhantes ao naive Bayes, demonstrando que o algoritmo não utilizou o potencial dos dados não rotulados. O TriTrain-AS conseguiu manter-se sem diferença estatisticamente significativa em relação ao naive Bayes, até 95% de exemplos rotulados; com 98% houve diferença significativa estatisticamente, isso se deve a pequena quantidade de exemplos rotulados que possivelmente causou falhas na seleção do classificador base, fazendo com que outros classificadores que não fossem o naive Bayes fossem selecionados em uma ou mais das 30 iterações.

Um resultado que chama a atenção são os classificadores supervisionados manterem um bom desempenho com até 80% de valores ausentes nesta base de dados como ilustrado na Figura 4.7. O TSVM só conseguiu se equiparar ao SMO e mesmo assim com 98% de exemplos rotulados foi o classificador com pior desempenho. O valor de C para o SMO foi de 0.01 e para o TSVM foi 0.1. Os gráficos excluem o TSVM por não ter sido possível obter resultados para todos os percentuais de ausência.

soybean: na base de dados *soybean*, Tabela 4.5, o melhor classificador supervisionado foi o SMO com $C = 0.1$ até o percentual de ausentes atingir 95% Figura 4.8. Quando esse valor foi atingido ocorreu um empate com o naive Bayes e posteriormente com 98% de percentual de ausentes o naive Bayes teve melhor desempenho que o SMO obtendo diferença estatística significativa. Esse fato pode ser explicado por classificadores discriminativos, como o SMO, terem melhor desempenho com um maior número de exemplos rotulados. Quando se tem poucos exemplos rotulados, os gerativos como naive Bayes, tem um melhor desempenho (Jordan, 2002).

Nos algoritmos semissupervisionados o algoritmo SSLVote teve uma grande degradação no desempenho indicando que os dados não rotulados o afetaram de forma negativa. Já o TriTrain-SMO, apesar dos resultados ligeiramente melhores que o seu classificador base, o SMO, não obteve diferença significativa com um intervalo de confiança de 95%. O mesmo ocorreu com o TriTrain-AS que não obteve diferença significativa do SMO. Novamente a seleção automática de classificador base funcionou até o percentual de ausentes

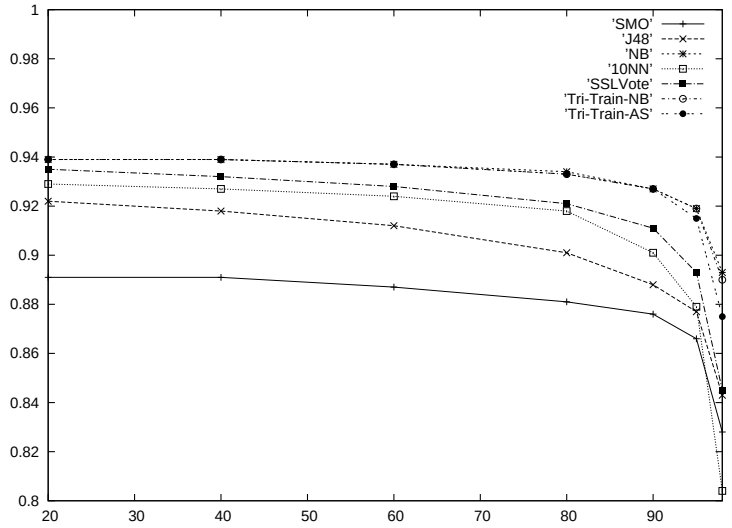


Figura 4.7: Resultados da base *solar-flare 2*

Tabela 4.5: Resultados da base *soybean*

Algoritmo	20%	40%	60%	80%	90%	95%	98%
SMO	0.760 (0.019)	0.745 (0.016)	0.719 (0.013)	0.676 (0.016)	0.635 (0.022)	0.586 (0.024)	0.533 (0.021)
J48	0.738 (0.029)	0.729 (0.019)	0.698 (0.019)	0.638 (0.025)	0.613 (0.028)	0.575 (0.041)	0.517 (0.022)
NB	0.691 (0.027)	0.684 (0.016)	0.668 (0.016)	0.642 (0.017)	0.614 (0.023)	0.587 (0.024)	**0.546 (0.023)**
10-NN	0.746 (0.026)	0.730 (0.014)	0.699 (0.016)	0.647 (0.019)	0.591 (0.025)	0.537 (0.014)	0.500 (0.014)
SSLVote2	0.747 (0.016)	0.717 (0.021)	0.670 (0.021)	0.619 (0.018)	0.596 (0.019)	0.574 (0.024)	0.536 (0.024)
TriTrain-SMO	**0.763 (0.020)**	**0.749 (0.015)**	**0.724 (0.015)**	**0.681 (0.014)**	**0.642 (0.025)**	**0.592 (0.025)**	0.532 (0.023)
TriTrain-AS	0.752 (0.024)	0.745 (0.016)	0.711 (0.018)	0.675 (0.024)	0.627 (0.030)	0.584 (0.030)	0.527 (0.031)

de 95%; com 98% o classificador base com melhor desempenho, o naive Bayes, não foi selecionado.

tic-tac-toe: na base *tic-tac-toe*, cujos resultados são ilustrados na Tabela 4.6, o SMO foi o melhor classificador supervisionado com o valor de $C = 0.1$, até 98% de percentual de ausentes. Nesse valor o naive Bayes conseguiu empatar com o SMO. O semissupervisionado TriTrain-SMO novamente degenerou para o classificador base e não obteve significância estatística sobre esse em nenhum momento.

Tabela 4.6: Resultados da base *tic-tac-toe*

Algoritmo	20%	40%	60%	80%	90%	95%	98%
SMO	0.994 (0.005)	0.976 (0.008)	0.912 (0.015)	0.794 (0.016)	0.689 (0.022)	0.624 (0.016)	0.575 (0.016)
J48	0.641 (0.033)	0.631 (0.018)	0.628 (0.020)	0.607 (0.023)	0.586 (0.029)	0.572 (0.031)	0.535 (0.034)
NB	0.746 (0.029)	0.737 (0.025)	0.715 (0.024)	0.677 (0.020)	0.635 (0.025)	0.613 (0.025)	0.571 (0.025)
10-NN	0.941 (0.011)	0.907 (0.016)	0.811 (0.015)	0.656 (0.014)	0.640 (0.016)	0.598 (0.020)	0.538 (0.019)
SSLVote	**0.998 (0.002)**	**0.996 (0.004)**	**0.990 (0.006)**	**0.927 (0.040)**	**0.792 (0.053)**	**0.719 (0.059)**	**0.598 (0.044)**
TriTrain-SMO	0.993 (0.006)	0.976 (0.011)	0.917 (0.020)	0.796 (0.018)	0.681 (0.025)	0.615 (0.019)	0.568 (0.021)
TriTrain-AS	0.993 (0.007)	0.978 (0.010)	0.921 (0.019)	0.796 (0.016)	0.667 (0.038)	0.609 (0.024)	0.562 (0.029)

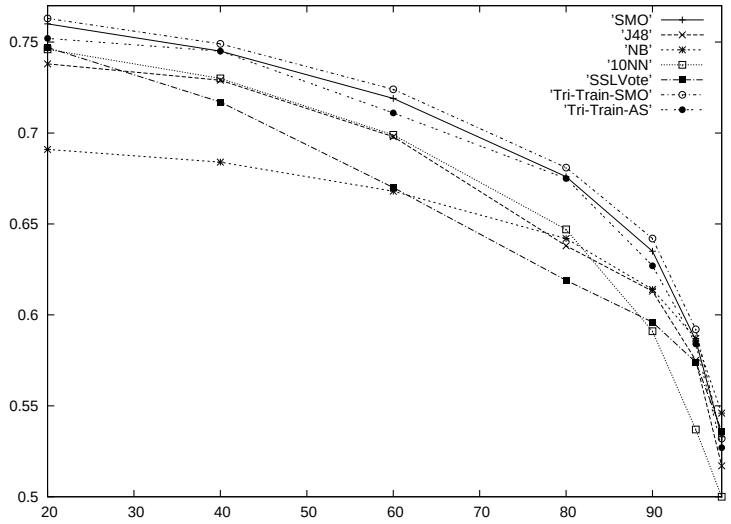


Figura 4.8: Resultados da base *soybean*

O TriTrain-AS conseguiu se manter sem diferença estatística significativa do SMO e TriTrain-SMO com até 80% de valores ausentes. Acima desse valor teve desempenho levemente inferior aos dois algoritmos citados. Já o SSLVote conseguiu utilizar o potencial dos dados não rotulados e foi estatisticamente melhor que o SMO e todos os percentuais de valores ausentes; com até 60% de valores ausentes seu desempenho varia muito pouco.

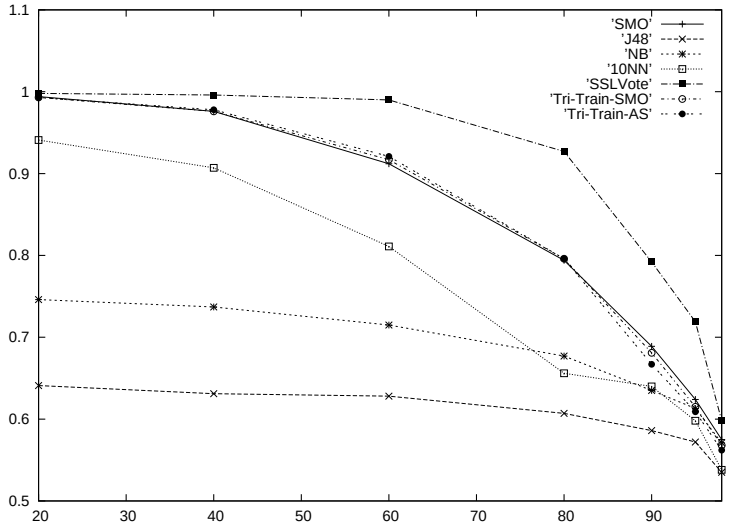


Figura 4.9: Resultados da base *tic-tac-toe*

Como pode ser verificado, TSVM apresentou problemas de execução não possibilitando reportar o seu desempenho de maneira adequada. Em diversas situações o algoritmo ficou

executando por semanas não finalizando a execução.

> Respondendo a pergunta: *Em problemas com grande quantidade de dados rotulados, os algoritmos de aprendizado supervisionado tem melhor desempenho que os semissupervisionados?*
>
> RESP: Não signicativamente com algoritmos semisupervisionados tipo *wrapper* (SSLVote, TriTrain-NB, TriTrain-AS). Os algoritmos supervisionados, quando apresentados com uma grande quantidade de exemplos rotulados não apresentam diferença estatística significativa com os algoritmos SSLVote, TriTrain-NB e TriTrain-AS. Isso deve-se principalmente devido aos algoritmos utilizados serem *Wrappers* sobre os algoritmos supervisionados.

Para os algoritmos não *wrapper* os resultados são inconclusivos. O algoritmo TSVM que e apresenta diferença estatística significativa para os algoritmos supervisionados para as bases *primary-tumor* e *solar-flare 2* independente da proporção sendo pior em todos os casos. A literatura reporta que TSVM é muito instável (Zhu, 2011). Futuramente serão testados outros algoritmos não *wrapper* para uma melhor avaliação.

4.4.2 Avaliação Experimental 2

Na Avaliação Experimental 1 apesar de não apresentar diferença estatística quando olhados separadamente por conjunto de dados, os algoritmos semissupervisionado, na maioria das vezes, reportam um maior valor médio de AUC. Para uma melhor análise dessa nesta Seção foi conduzido um teste não paramétrico conhecido como teste de Friedman e o teste pos-hoc Nemenyi (Demsar, 2006).

Na Tabela 4.7 são apresentados as informações de ranking de cada algoritmo para cada conjunto de dados, entre parênteses. O número 1 entre parênteses indica o melhor resultado, número 2 entre parênteses o segundo melhor resultado e assim por diante. Quando reportado valores intermediários como 2.5 indicam empates entre 2 e 3. O ranking médio é apresentado no final da tabela onde os menores valores indicam os melhores resultados. Observe que os três algoritmos semisupervisionados, TriTrain, TriTrain-AS e SSLVote, são os algoritmos semisupervisionados apresentados com menor ranking médio com 2.115, 2.981 e 3.385 respectivamente. Os algoritmos supervisionados são os quatro últimos colocados.

A estatística F calculada é 17.80. O valor crítico para a estatística F com 6 e 150 graus de liberdade com p-valor de 5% (grau de confiança de 95%) é de 2,16. Assim com o valor crítico obtido pode-se rejeitar a hipótese nula de que os algoritmos possuem desempenho similar. Para verificar se existe diferença significativa entre os algoritmos é calculado o teste post-hoc Nemenyi.

De acordo com a estatística Nemenyi, o valor crítico para comparação do ranking médio de dois diferentes algoritmos com p-valor de 5% é 1.77. Diferença entre algoritmos com ranking médio superior a 1.77 indica diferença estatística significativa. Na Figura 4.10 é apresentado o gráfico de diferença crítica. O eixo principal que varia de 1 a 7 e o ranking médio de cada

Tabela 4.7: Resultados do ranking médio para realizar o teste de Friedman

base	%	TriTrain	TriTrain-AS	SSLVote	NB	SMO	10-NN	J48
	20%	0.766(2.5)	0.766(2.5)	0.768(1.0)	0.765(4.0)	0.634(7.0)	0.715(5.0)	0.695(6.0)
	40%	0.743(1.0)	0.725(4.0)	0.733(3.0)	0.742(2.0)	0.631(7.0)	0.697(5.0)	0.651(6.0)
primary-tumor	60%	0.723(1.0)	0.712(3.0)	0.683(4.0)	0.721(2.0)	0.618(7.0)	0.680(5.0)	0.633(6.0)
	80%	0.701(3.0)	0.712(1.0)	0.653(4.0)	0.703(2.0)	0.613(7.0)	0.646(5.0)	0.617(6.0)
	90%	0.654(1.0)	0.611(3.0)	0.608(4.0)	0.643(2.0)	0.577(5.0)	0.545(7.0)	0.552(6.0)
	20%	0.939(2.0)	0.939(2.0)	0.935(4.0)	0.939(2.0)	0.891(7.0)	0.929(5.0)	0.922(6.0)
	40%	0.939(2.0)	0.939(2.0)	0.932(4.0)	0.939(2.0)	0.891(7.0)	0.927(5.0)	0.918(6.0)
	60%	0.937(2.0)	0.937(2.0)	0.928(4.0)	0.937(2.0)	0.887(7.0)	0.924(5.0)	0.912(6.0)
solar-flare 2	80%	0.933(2.5)	0.933(2.5)	0.921(4.0)	0.934(1.0)	0.881(7.0)	0.918(5.0)	0.901(6.0)
	90%	0.927(2.0)	0.927(2.0)	0.911(4.0)	0.927(2.0)	0.876(7.0)	0.901(5.0)	0.888(6.0)
	95%	0.919(1.5)	0.915(3.0)	0.893(4.0)	0.919(1.5)	0.866(7.0)	0.879(5.0)	0.877(6.0)
	98%	0.890(2.0)	0.875(3.0)	0.845(4.0)	0.893(1.0)	0.828(6.0)	0.804(7.0)	0.843(5.0)
	20%	0.763(1.0)	0.752(3.0)	0.747(4.0)	0.691(7.0)	0.760(2.0)	0.746(5.0)	0.738(6.0)
	40%	0.749(1.0)	0.745(2.5)	0.717(6.0)	0.684(7.0)	0.745(2.5)	0.730(4.0)	0.729(5.0)
	60%	0.724(1.0)	0.711(3.0)	0.670(6.0)	0.668(7.0)	0.719(2.0)	0.699(4.0)	0.698(5.0)
soybean	80%	0.681(1.0)	0.675(3.0)	0.619(7.0)	0.642(5.0)	0.676(2.0)	0.647(4.0)	0.638(6.0)
	90%	0.642(1.0)	0.627(3.0)	0.596(6.0)	0.614(4.0)	0.635(2.0)	0.591(7.0)	0.613(5.0)
	95%	0.592(1.0)	0.584(4.0)	0.574(6.0)	0.587(2.0)	0.586(3.0)	0.537(7.0)	0.575(5.0)
	98%	0.532(4.0)	0.527(5.0)	0.536(2.0)	0.546(1.0)	0.533(3.0)	0.500(7.0)	0.517(6.0)
	20%	0.993(3.5)	0.993(3.5)	0.998(1.0)	0.746(6.0)	0.994(2.0)	0.941(5.0)	0.641(7.0)
	40%	0.976(3.5)	0.978(2.0)	0.996(1.0)	0.737(6.0)	0.976(3.5)	0.907(5.0)	0.631(7.0)
	60%	0.917(3.0)	0.921(2.0)	0.990(1.0)	0.715(6.0)	0.912(4.0)	0.811(5.0)	0.628(7.0)
tic-tac-toe	80%	0.796(2.5)	0.796(2.5)	0.927(1.0)	0.677(5.0)	0.794(4.0)	0.656(6.0)	0.607(7.0)
	90%	0.681(3.0)	0.667(4.0)	0.792(1.0)	0.635(6.0)	0.689(2.0)	0.640(5.0)	0.586(7.0)
	95%	0.615(3.0)	0.609(5.0)	0.719(1.0)	0.613(4.0)	0.624(2.0)	0.598(6.0)	0.572(7.0)
	98%	0.568(4.0)	0.562(5.0)	0.598(1.0)	0.571(3.0)	0.575(2.0)	0.538(6.0)	0.535(7.0)
	rank médio	2.115	2.981	3.385	3.558	4.500	5.385	6.077

algoritmo é posicionado neste eixo. Por exemplo, o ranking médio de TriTrain é 2.11 e portanto no gráfico o algoritmo TriTrain está ligado com o eixo principal a posição 2.11. Com um valor crítico de 1.77 TriTrain não tem diferença significativa com todos os algoritmos que possuem ranking médio menor que $2.11 + 1.77 = 3.77$. Assim os algoritmos TriTrain-AS (2.98), SSL-Vote (3.38) e NB (3.55) que possuem ranking médio menor que 3.77 estão ligados por uma linha horizontal logo abaixo do eixo principal que indica que esses algoritmos não possuem diferença significativa com TriTrain. Assim, os algoritmos que estão ligados por essas linhas significam que não possuem diferença significativa entre si. TriTrain não está ligado com SMO, 10NN e J48, isto significa que existe diferença significativa entre eles. Como ranking médio de TriTrain é menor que esses três algoritmos supervisionados pode-se dizer que TriTrain possue melhor desempenho que SMO, 10NN e J48 com diferença significativa.

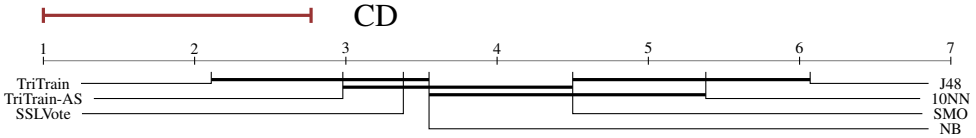


Figura 4.10: Gráfico de diferença crítica. Algoritmos conectados por linhas significam que não possuem diferença significativa

Com esses valores é possível responder a pergunta:

Independente da proporção de exemplos rotulados e não rotulados, qual possui

melhor desempenho, algoritmos supervisionados ou semissupervisionados?

RESP: Para os algoritmos testados, o algoritmo semissupervisionado TriTrain reporta melhores resultados que os algoritmos supervisionados SMO, 10-NN e J48 com 95% graus de confiança, e por muito pouco ($3.558 - 2.115 = 1.447 < 1.77$) não apresenta diferença significativa em relação ao NB. Assim pode-se concluir que TriTrain reporta resultados melhores para três dos quatro algoritmos supervisionados apresentados.

4.4.3 Avaliação Experimental 3

A literatura reporta um bom desempenho de semissupervisionado principalmente em bases de textos em situações onde a quantidade de exemplos não rotulados é muito maior que a quantidade de exemplos rotulados (Ghani, 2001; Blum e Mitchell, 1998). A pergunta que dirige a análise desta Seção consiste em responder:

Como é o comportamento do semissupervisionado em problemas de imputação com a característica de grande desproporção de não rotulados e rotulados?

O atributo imputado para as bases de dados escolhidas possuem 2, 7, 3 e 6 valores distintos para as bases *primary-tumor*, *soybean*, *tic-tac-toe* e *solar-flare 2* respectivamente. Os experimentos conduzidos até então produziam diferentes proporções de exemplos não rotulados de maneira estratificada para essas quantidade de classes. Para simular uma grande desproporção de não-rotulados e rotulados foi preparado um experimento que utiliza apenas 5 exemplos rotulados de cada classe. O conjunto de dados formado por esses exemplos rotulados é considerado o conjunto de treino e os resultados são reportados na Tabela 4.8. Os valores em negrito indicam os melhores resultados supervisionados e semissupervisionados. Como pode-se observar os valores médios dos semissupervisionados são sempre superiores aos supervisionados.

Tabela 4.8: Resultados as bases *#5 rotulados*

base	SMO	J48	NB	10-NN	SSLVote	TriTrain	TriTrain-AS	dif. significativa
primary-tumor	0.569 (0.059)	0.543 (0.066)	**0.624 (0.064)**	0.500 (0.000)	0.603 (0.084)	0.600 (0.066)	**0.636 (0.058)**	-
solar-flare2	0.847 (0.020)	0.835 (0.044)	**0.888 (0.016)**	0.816 (0.027)	0.852 (0.023)	**0.895 (0.017)**	0.865 (0.038)	-
soybean	**0.606 (0.018)**	0.596 (0.029)	0.590 (0.020)	0.553 (0.015)	0.578 (0.017)	**0.609 (0.022)**	0.604 (0.025)	-
tictactoe	**0.560 (0.017)**	0.535 (0.038)	0.560 (0.022)	0.531 (0.019)	**0.569 (0.036)**	0.556 (0.023)	0.544 (0.028)	-

A última coluna indica o resultado do teste t-pareado e o sinal negativo "-" indica que não houve diferença significativa e o sinal positivo "+" caso exista diferença significativa. Como pode-se observar não houve diferença significativa para nenhuma das comparações apesar do valor médio ser sempre maior (melhor) para o semissupervisionado.

4.5 Considerações Finais

Os resultados experimentais apresentam resultados promissores para o uso de aprendizado semissupervisionado em problemas de valores ausentes. Nas três avaliações experimentais, o

algoritmo semissupervisionado apresentou resultados melhores que o supervisionado, porém apenas na segunda avaliação experimental foi detectado diferença significativa. De maneira resumida obteve-se os seguintes resultados:

- Na primeira avaliação foi possível verificar se os algoritmos de aprendizado supervisionado tem desempenho superior aos algoritmos semissupervisionado quando existem grandes quantidades de exemplos rotulados disponíveis. Os resultados demonstram que quando se trata de um algoritmo semissupervisionado tipo *wrapper* não existe diferença significativa.

- Na segunda avaliação o algoritmo semissupervisionado, utilizando diversas proporções de valores ausentes, apresenta melhor desempenho que o aprendizado supervisionado para os algoritmos avaliados com diferença significativa.

- Na terceira avaliação pode-se verificar se, mesmo com cinco exemplos de cada classe, o algoritmo semissupervisionado pode ter bom desempenho para o problema de imputação. Os resultados indicam valores levemente superiores para semissupervisionado mas novamente não existe diferença significativa entre semissupervisionado e supervisionado.

CAPÍTULO

5

Conclusões

Neste capítulo são apresentadas as conclusões deste trabalho. Na Seção 5.1 é realizado um resumo dos objetivos nesta disseratção. Na Seção 5.2 são mencionadas as contribuições deste trabalho. Na Seção 5.3 são citadas as limitações e na Seção 5.4 são apresentadas algumas direções de trabalhos futuros.

5.1 Resumo dos Objetivos e Principais Resultados

O escopo deste trabalho foi utilizar algoritmos e ferramentas disponíveis livremente, para verificar se aprendizado semissupervisionado pode ser usado para tarefas de imputação de dados. Assim, de posse desses resultados as questões inicialmente propostas podem ser respondidas:

Os dados não rotulados podem ajudar a melhorar a qualidade da imputação?

Na avaliação experimental 1 e 2 os resultados indicam que os dados não rotulados ajudam os classificadores semissupervisionados. Na avaliação experimental 1 não houve diferença estatística significativa utilizando teste t pareado entre o melhor supervisionado e o melhor semissupervisionado, apesar do semissupervisionado apresentar valores ligeiramente superiores. Já na avaliação experimental 2 os algoritmos semissupervisionados apresentam melhores resultados que os supervisionados com diferença significativa. Assim pode-se concluir que os dados não rotulados ajudam na melhoria da qualidade de imputação.

Vale ressaltar que a imputação de valores ausentes neste trabalho é tratado como uma tarefa de natureza transdutiva, conforme já explicado no Capítulo 3.6. Desta forma, os dados não rotulados devem aumentar a qualidade da imputação, desde que se utilize um algoritmo seguro (Goldberg, 2010), ou seja, um algoritmo semissupervisionado que não sofra degradação quando os pressupostos do aprendizado semissupervisionado falham em uma base de dados.

Qual algoritmo semissupervisionado seria o melhor para tarefas de imputação?

Os algoritmos do tipo *wrapper*, como TRI-TRAINING, oferecem uma boa alternativa para tarefas de imputação. Como é um algoritmo que é executado sobre um algoritmo de aprendizado supervisionado, nos experimentos apresentados neste trabalho, os algoritmos semissupervisionados apresentaram valores de AUC superior ao supervisionado. Normalmente o desempenho desses algoritmos *wrapper* tem como limite inferior (*lower bound*) o próprio algoritmo supervisionado utilizado como algoritmo base. Os algoritmos não *wrapper* não foram testados de maneira extensiva e a implementação de TSVM apresentou diversas instabilidades não possibilitando obter o resultado final. Assim, com os resultados obtidos, os algoritmos do tipo *wrapper* são recomendados para tarefas de imputação e os não *wrapper* ainda são inconclusivos.

Um outro trabalho futuro, além de investigar algoritmos não *wrapper* seria interessante investigar o fSSL (*Formulative Semi-Supervised Learning*), onde ao invés de rotular um atributo de classe, são criados atributos artificiais a partir do conhecimento extraído dos dados não rotulados. O fSSL tem uma menor propagação de erro, pois algoritmos de aprendizado toleram mais ruído em atributos comuns do que em atributos de classe (Zhu e Wu, 2004).

Quando usar ou não usar o aprendizado semissupervisionado para imputação?

Na avaliação experimental 1 apesar do semissupervisionado ter valores de AUC ligeiramente superiores quando comparados ao supervisionado, não houve diferença significativa entre o melhor algoritmo supervisionado e o melhor algoritmo semissupervisionado. Na avaliação experimental 2 mostrou-se que os semissupervisionados são superiores aos supervisionados. Assim pode-se concluir que o uso de semissupervisionado pode produzir resultados muito similares aos resultados do aprendizado supervisionado e na maioria das vezes apresenta desempenho melhor. Assim fica a recomendação do uso dos algoritmos de aprendizado semissupervisionado apresentados neste trabalho, mas sempre utilizando com cautela pois nem sempre existe diferença significativa.

Durante a investigação a esta pergunta observou-se que os algoritmos semissupervisionados disponíveis atualmente são em sua maioria criados para tarefas onde se tem pouquíssimos exemplos rotulados e grande quantidade de dados não rotulado e a propagação de erro não é tão crítica quanto na imputação. Naturalmente, isso leva a uma outra questão: Será que faz sentido imputar um atributo com muitos valores ausentes (não rotulados), mais de 90% de exemplos não rotulados, como os utilizados em semissupervisionados tradicional?

Se os valores imputados forem confiáveis, acreditamos que sim. No entando, para verificar a confiabilidade de um algoritmo de aprendizado semissupervisionado é necessário um conjunto de teste que verifica se o algoritmo não degrada durante as iterações. Assim para responder a essa outra pergunta seria necessário um outro conjunto de experimentos que ficará como trabalho futuro. A terceira avaliação experimental (Seção 4.4.3 na página 67) responde em parte a esta pergunta pois verifica o quão robusto é um algoritmo semissupervisionado para

o problema de imputação com apenas 5 exemplos de cada classe. Os resultados não mostraram diferença significativa com os algoritmos supervisionados, apesar dos valores médios de AUC dos semissupervisionados serem superiores que os supervisionados em todos os quatro conjuntos testados. Assim considera-se necessário uma melhor exploração deste problema para resultados mais conclusivos. Os resultados iniciais ao uso de muitos exemplos não rotulados e poucos exemplos rotulados apontam fracamente que semissupervisionado pode ser útil também em situações extremas como esta.

> *É possível selecionar o melhor classificador para imputação em uma determinada base de dados?*

Pelos experimentos realizados, é possivel recomendar bons algoritmos mas sempre é difícil apontar o "melhor". O TRI-TRAINING modificado para autoseleção do classificador conseguiu na maioria das vezes selecionar o algoritmo supervisionado de base que teve o melhor desempenho. Trabalhos futuros podem verificar qual o percentual de rotulados necessários para realizar esta seleção de forma confiável com um minimo de sobrecarga.

5.2 Contribuições

Este trabalho possibilitou as seguintes contribuições:

1. Esclarece que a tarefa de imputação de valores ausentes para base de dados estáticas é de natureza transdutiva.
2. Demonstra por meio da avaliação experimental que em situações onde o algoritmo de aprendizado consegue fazer uso dos exemplos não rotulados, o desempenho de tarefas de imputação tende a ser melhor que os métodos que utilizam apenas os dados rotulados para a indução de hipóteses.
3. Implementa o suporte de classificadores semissupervisionados ao programa de mineração de dados Weka, com a confecção de dois *wrappers* de classificadores semissupervisionados.
4. Propõem o algoritmo SSLVote.
5. Propõem uma modificação no algoritmo TRI-TRAINING que possibilita selecionar o melhor classificador para tarefas de imputação.

5.3 Limitações

Este trabalho tem as seguintes limitações:

1. As bases de dados usadas são relativamente pequenas e possuem apenas atributos nominais e/ou binários.

2. A forma que os atributos foram selecionados utilizando o *wrapper* pode ter prejudicado o algoritmo SVM, por ter sido utilizado como algoritmo base do *wrapper* o atributo encontrado é o atributo mais significativo para o SVM.

3. A quantidade de bases de dados utilizadas é relativamentepequena.

4. O mecanismo de aleatóriaedade utilizado foi do tipo MCAR, o ideal seria se utilizar MAR por ser o mais comum em dados reais.

5. Só foi trabalhado o problema de imputação univariada.

5.4 Trabalhos Futuros

No decorrer deste trabalho surgiram novos questionamentos e novas idéias que são apresentadas a seguir:

1. Teste e modificação de algoritmos de aprendizado semissupervisionados que tratam do problema de regressão para o tratamento de atributos ausentes com valores numéricos.

2. Teste do SSLVote variando os seus parâmetros.

3. Implementação do algoritmo fSSL para imputação de dados. O fSSL é uma área relativamente nova no aprendizado semissupervisionado e aparentemente muito promissora.

4. Pelos resultados obtidos se pode ver que não existe um algoritmo de imputação que seja ótimo em cada bases de dados, seria interessante encontrar uma forma de selecionar o algoritmo mais apropriado para cada base de dados.

5. Avaliação experimental extensiva com algoritmos semissupervisionado não *wrappers*.

6. Tratamento de imputação múltipla utilizando algoritmos semissupervisionados.

Referências Bibliográficas

Abayomi, K., Gelman, A., e Levy, M. (2008). Diagnostics for multivariate imputations. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 57(3):273–291. Citado na página 31.

Acuna, E. e Rodriguez, C. (2004). The treatment of missing values and its effect in the classifier accuracy. In *Proceedings of the Meeting of the International Federation of Classification Societies : Classification, Clustering and Data Mining Applications*, páginas 639–648. Springer. Citado na página 33.

Afifi, A. e Elashoff, M. (1966). Missing observations in multivariate statistics I: Review of the literature. *Journal of the American Statistical Association*, 61(315):595–604. Citado na página 3.

Anderson, T. W. (1957). Maximum likelihood estimates for a multivariate normal distribution when some observations are missing. *Journal of the American Statistical Association*, 52(278):200–203. Citado na página 2.

Bair, E. e Tibshirani, R. (2004). Semi-supervised methods to predict patient survival from gene expression data. *PLoS biology*, 2(4):511–522. Citado na página 3.

Batista, G. E. A. P. A. (2003). Pré-processamento de dados em aprendizado de máquina supervisionado. Dissertação de Doutorado, ICMC–USP. Citado nas páginas 19, 23, e 24.

Bennett, K. P. e Demiriz, A. (1998). Semi-supervised support vector machines. In *Advances in Neural Information Processing Systems*, páginas 368–374. MIT Press. Citado na página 5.

Blum, A. e Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, COLT' 98, páginas 92–100, New York, NY, USA. ACM. Citado nas páginas 4, 5, 48, e 67.

Boser, B. E., Guyon, I. M., e Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, páginas 144–152, New York, NY, USA. ACM. Citado na página 43.

Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Jackel, L. D., Le Cun, Y., Muller, U. A., Säckinger, E., Simard, P., e Vapnik, V. (1994). Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Conference B: Computer Vision & Image Processing.*, volume 2, páginas 77–82, Jerusalem. IEEE. Citado na página 43.

Braga, I. A. (2010). Aprendizado semissupervisionado multidescrição em classificação de textos. Dissertação de Mestrado, ICMC–USP. Citado na página 5.

Buck, S. F. (1960). A method of estimation of missing values in multivariate data suitable for use with an electronic computer. *Journal of the Royal Statistical Society Series B (Statistical Methodology)*, 22(2):302–306. Citado na página 3.

Castaneda, R., Ferlin, C., Goldschmidt, R. R., de Abreu Soares, J., de Carvalho, L. A. V., e Choren, R. (2008). Aprimorando processos de imputação multivariada de dados com workflows. In *XXIII Simpósio Brasileiro de Banco de Dados*, páginas 238–252. SBC. Citado nas páginas xiii, 30, e 31.

Castaneda, R., Goldschmidt, R., e Choren, R. (2009). Um ambiente orientado a agentes para experimentação em imputação sequencial. In *V Workshop on Software Engineering for Agent-oriented Systems*, páginas 47–58. Citado nas páginas xiii, 3, 32, e 45.

Cayton, L. (2005). Algorithms for manifold learning. Relatório técnico, UCSD tech report CS2008-0923. Citado nas páginas xiii e 19.

Chapelle, O., Schölkopf, B., e Zien, A., editors (2006). *Semi-Supervised Learning*. MIT Press, Cambridge, MA. Citado nas páginas 14, 18, e 19.

Cozman, F. e Cohen, I. (2006). Risks of semi-supervised learning. In *Semi-Supervised Learning*, páginas 55–68. MIT Press. Citado na página 18.

Craven, M. W. e Shavlik, J. W. (1995). Extracting comprehensible concept representations from trained neural networks. In *Working Notes on the IJCAI95 Workshop on Comprehensibility in Machine Learning*, páginas 61–75. Citado na página 22.

Demiriz, A. e Bennett, K. P. (2001). Optimization approaches to semi-supervised learning. In *Complementarity: Applications, Algorithms and Extensions*, páginas 121–141. Kluwer Academic. Citado nas páginas 5 e 47.

Demsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30. Citado na página 65.

Dietterich, T. G. e Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286. Citado na página 53.

Efron, B. e Tibshirani, R. J. (1993). *An Introduction to the Bootstrap*. Chapman & Hall, New York. Citado na página 48.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874. Citado na página 56.

Fayyad, U., Fayyad, U., Piatetsky-shapiro, G., Piatetsky-shapiro, G., Smyth, P., e Smyth, P. (1996). Knowledge discovery and data mining: Towards a unifying framework. In *Procedings of the Second International Conference on Knowledge Discorevy and Data Mining (KDD-96)*, páginas 82–88. AAAI Press. Citado nas páginas xiii, 20, e 21.

Fisher, L. e Ness, J. W. V. (1971). Admissible clustering procedures. *Biometrika*, 59(1):91–104. Citado na página 15.

Frank, A. e Asuncion, A. (2010). UCI machine learning repository. Citado na página 54.

Fung, G. e Mangasarian, O. L. (2001). Semi-supervised support vector machines for unlabeled data classification. *Optimization Methods and Software*, 15(1):29–44. Citado na página 5.

Gelman, A. e Raghunathan, T. (2001). Using conditional distributions for missing-data imputation. *discussion of Conditionally Specified Distributions by Arnold et al Statistical Science*, 3:268–269. Citado nas páginas 30 e 31.

Ghani, R. (2001). Combining labeled and unlabeled data for text classification with a large number of categories. In *ICDM*, páginas 597–598. Citado na página 67.

Goldberg, A. (2010). New directions in semi-supervised learning. Dissertação de Doutorado, University of Wisconsin. Citado na página 69.

Goldberg, A. B. e Zhu, X. (2009). Keepin' it real: semi-supervised learning with realistic tuning. In *Proceedings of the NAACL HLT 2009 Workshop on Semi-Supervised Learning for Natural Language Processing*, SemiSupLearn '09, páginas 19–27, Stroudsburg, PA, USA. Association for Computational Linguistics. Citado na página 4.

Gould, A. (1980). A new approach to the analysis of clinical drug trials with withdrawals. *Biometrics*, 36(4):721–727. Citado na página 3.

Haffari, G. e Sarkar, A. (2007). Analysis of semi-supervised learning with the yarowsky algorithm. Technical Report TR 2007-07, Simon Fraser University, 8888 University Drive, Burnaby, B.C. Canada V5A 1S6. Citado na página 4.

Jerez, J. M., Molina, I., García-Laencina, P. J., Alba, E., Ribelles, N., Martín, M., e Franco, L. (2010). Missing data imputation using statistical and machine learning methods in a real breast cancer problem. *Artificial intelligence in medicine*, 50(2):105–115. Citado na página 3.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning (ECML)*, páginas 137–142, Berlin. Springer. Citado na página 52.

Joachims, T. (1999a). Making large-scale SVM learning practical. In *Advances in Kernel Methods - Support Vector Learning*, chapter 11, páginas 169–184. MIT Press, Cambridge, MA. Citado na página 52.

Joachims, T. (1999b). Transductive Inference for Text Classification using Support Vector Machines. In *Proceedings of ICML-99, 16th International Conference on Machine Learning*, páginas 200–209, Bled, SL. Morgan Kaufmann Publishers, San Francisco, US. Citado nas páginas 5 e 47.

Joachims, T. (1999c). Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning (ICML)*, páginas 200–209, Bled, Slowenien. Citado na página 52.

Joachims, T. (2002). *Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms*. Kluwer/Springer. Citado na página 52.

John, G. H. e Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. In Besnard, P. e Hanks, S., editors, *UAI*, páginas 338–345. Morgan Kaufmann. Citado na página 52.

Jordan, A. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems 14*, volume 2, pagina 841. Citado na página 62.

Kimball, R. e Ross, M. (2002). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (Second Edition)*. Wiley, 2 edition. Citado na página 21.

Kohavi, R. e John, G. H. (1997). Wrappers for feature subset selection. *ARTIFICIAL INTELLIGENCE*, 97(1):273–324. Citado na página 56.

Lakshminarayan, K., Harp, S. A., e Samad, T. (1999). Imputation of missing data in industrial databases. *Applied Intelligence*, 11(3):259–275. Citado na página 3.

Lavori, P. W. (1992). Clinical trials in psychiatry: should protocol deviation censor patient data. *Neurosychompharmacology*, 6(1):39–48. Citado na página 3.

Liang, P. e Jordan, M. I. (2008). An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, páginas 584–591, New York, NY, USA. ACM. Citado na página 14.

Little, R. J. A. e Rubin, D. B. (2002). *Statistical Analysis With Missing Data*. John Wiley and Sons, 2nd edition edition. Citado nas páginas 24, 27, 30, e 31.

Lorena, A. C. e de Carvalho, A. C. P. L. F. (2007). Uma introdução às support vector machines. *RITA*, 14(2):43–67. Citado nas páginas xiii e 45.

Maeireizo, B., Litman, D., e Hwa, R. (2004). Co-training for predicting emotions with spoken dialogue data. In *The Companion Volume to the Proceedings of 42st Annual Meeting of the Association for Computational Linguistics*, páginas 202–205, Barcelona, Spain. Association for Computational Linguistics. Citado na página 5.

Martins, C. A. (2003). Uma abordagem para pré-processamento de dados textuais em algoritmos de aprendizado. Dissertação de Doutorado, ICMC–USP. Citado nas páginas xiii, 16, e 17.

Matsubara, E. T., Prati, R. C., Batista, G. E., e Monard, M. C. (2008). Missing value imputation using a semi-supervised rank aggregation approach. In *Proceedings of the 19th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligence*, SBIA '08, páginas 217–226, Berlin, Heidelberg. Springer-Verlag. Citado nas páginas 4 e 48.

Merz, C. J., Clair, D. S., e Bond, W. E. (1992). Semi-supervised adaptive resonance theory (smart2). In *International Joint Conference on Neural Networks*, volume 3, páginas 851–856. Institute of Electrical and Electronics Engineers. Citado na página 5.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York. Citado nas páginas 9 e 34.

Monard, M. C., de Almeida Prado Alves Batista, G. E., Kawamoto, S., e Pugliesi, J. B. (1997). Uma introdução ao aprendizado simbólico de máquina por exemplos. Didactical Notes 29, ICMC-USP, São Carlos - SP. Citado na página 9.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117. Citado na página 1.

Nordheim, E. (1984). Inference from nonrandomly missing categorical data: an example from a genetic study of Turner's syndrome. *Journal of the American Statistical Association*, 79(388):772–780. Citado na página 3.

Osuna, E., Freund, R., e Girosi, F. (1997). Training support vector machines: an application to face detection. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, volume 0, páginas 130–136, Los Alamitos, CA, USA. IEEE Computer Society. Citado na página 52.

Platt, J. C. (1999). Advances in kernel methods. chapter Fast training of support vector machines using sequential minimal optimization, páginas 185–208. MIT Press, Cambridge, MA, USA. Citado na página 52.

Pyle, D. (1999). *Data preparation for data mining*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. Citado na página 21.

Rosenberg, C., Hebert, M., e Schneiderman, H. (2005). Semi-supervised self-training of object detection models. In *Seventh IEEE Workshop on Applications of Computer Vision*, páginas 29–36. Citado na página 5.

Rubin, D. B. (1987). *Multiple imputation for nonresponse in surveys*. New York: Wiley. Citado nas páginas 3 e 29.

Russell, S. J. e Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education. Citado na página 7.

Ryszard S. Michalski, Jaime G. Carbonell, T. M. M. (1986). *Machine Learning: An Artificial Intelligence Approach*, volume II. Morgan Kaufmann. Citado nas páginas 8, 9, e 10.

Sande, I. G. (1983). Hot-deck imputation procedures. In *In Incomplete Data in Sample Surveys*, páginas 339–349. Academic Press. Citado nas páginas 2 e 27.

Schafer, J. L. (1997). *Analysis Of Incomplete Multivariate Data*. Chapman and Hall/Crc. Citado na página 2.

Shannon, C. (1949). Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715. Citado na página 36.

Simon, H. A. (1983). Search and reasoning in problem solving. *Artificial Intelligence*, 21(1–2):7–29. Citado nas páginas 8 e 9.

Singh, A., Nowak, R. D., e Zhu, X. (2008). Unlabeled data: Now it helps, now it doesn't. In *NIPS'08*, páginas 1513–1520. Citado na página 18.

Tenenbaum, J. B., Silva, V., e Langford, J. C. (2000). A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323. Citado na página 18.

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., e Altman, R. B. (2001). Missing value estimation methods for dna microarrays. *Bioinformatics (Oxford, England)*, 17(6):520–525. Citado nas páginas 34 e 52.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460. Citado na página 9.

Tuzhilin, A. (1995). On subjective measures of interestingness in knowledge discovery. In *In Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, páginas 275–281. AAAI Press. Citado na página 22.

van Buuren, S., Brand, J. P. L., Oudshoorn, C. G. M. G., e Rubin, D. B. (2006). Fully conditional specification in multivariate imputation. *Journal of Statistical Computation and Simulation*, 76(12):1049–1064. Citado nas páginas 3, 29, e 30.

van Buuren, S. e Oudshoorn, K. (1999). Flexible multivariate imputation by mice. Relatório técnico, TNO Prevention and Health. Citado na página 3.

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley-Interscience. Citado nas páginas 5 e 47.

Vapnik, V. N. e Chervonenkis, A. Y. (1974). *Theory of Pattern Recognition [in Russian]*. Nauka, USSR. Citado nas páginas 18, 43, e 47.

Weiss, S. M. e Kulikowski, C. A. (1991). *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA. Citado na página 10.

Weston, J. (2008). Large scale semi-supervised learning. In *Proceedings of NATO Advanced Study Institute on Mining Massive Data Sets for Security*, páginas 62–75. IOS Press. Citado na página 4.

Williams, D., Liao, X., Xue, Y., Carin, L., e Krishnapuram, B. (2007). On classification with incomplete data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):427–436. Citado na página 4.

Wu, X. e Barbará, D. (2002). Modeling and imputation of large incomplete multidimensional datasets. In *DaWaK*, páginas 286–295. Citado na página 3.

Yajima, Y. e Hoshiba, T. (2005). Optimization approaches for semi-supervised learning. In *Proceedings of the Fourth International Conference on Machine Learning and Applications*, ICMLA '05, páginas 247–252, Washington, DC, USA. IEEE Computer Society. Citado na página 5.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, páginas 189–196. Citado na página 5.

Yue, Y., Finley, T., Radlinski, F., e Joachims, T. (2007). A support vector method for optimizing average precision. In *ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, páginas 271–278. Citado na página 52.

Zhou, Z.-H. e Li, M. (2005). Tri-Training: Exploiting Unlabeled Data Using Three Classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1529–1541. Citado na página 48.

Zhu, X. (2011). Cross-domain semi-supervised learning using feature formulation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 41(6):1627–1638. Citado nas páginas 4 e 65.

Zhu, X. e Goldberg, A. B. (2009). *Introduction to Semi-Supervised Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers. Citado nas páginas 7, 12, 14, e 17.

Zhu, X. e Wu, X. (2004). Class noise vs. attribute noise: a quantitative study of their impacts. *Artif. Intell. Rev.*, 22(3):177–210. Citado nas páginas 5 e 70.