Seleção de Instâncias Baseado em Aprendizado de Métricas para K Vizinhos Mais Próximos

Eduardo Zárate Guerreiro Max

SERVIÇO DE PÓS-GRADUAÇÃO DA FACOM-UFMS

Data de Depósito:

Assinatura:_____

Seleção de Instâncias Baseado em Aprendizado de Métricas para K Vizinhos Mais Próximos

Eduardo Zárate Guerreiro Max

Orientador: *Prof. Dr. Edson Takashi Matsubara*
Coorientador: *Prof. Dr. Ricardo Marcondes Marcacini*

Dissertação apresentada à Faculdade de Computação da Universidade Federal de Mato Grosso do Sul - FACOM-UFMS como parte dos requisitos necessários à obtenção do título de Mestre em Ciência da Computação.

UFMS - Campo Grande
Agosto/2016

Aos meus avós,
Joneza e Raul,

Aos meus pais,
Rosálie e Claudio,

À minha família,

À Edson Takashi Matsubara.

Agradecimentos

Agradeço aos meus pais, Claudio e Rosálie, que sempre me incentivaram e apoiaram em minhas escolhas. Agradeço a minha irmã Natalie, que mesmo em distância pude contar com ela. Agradeço a Lucelaine por ter me ajudado nos momentos difíceis e ter sido companheira durante todo processo.

Ao professor Edson Takashi pela sua orientação, pelas motivações de cada reunião e pelo conhecimento que obtive ao seu lado durante toda nossa convivência. Ao professor Ricardo Marcacini pela coorientação e seu conhecimento compartilhado nas reuniões.

Agradeço também ao pessoal do LIA, que me fizeram companhia no laboratório. Aos amigos, Anderson Bessa, Igor Visioli e Manuel Arn, pelas conversas e momentos felizes.

E por fim, agradeço a CAPES pela bolsa de mestrado.

Resumo

A seleção de instâncias, em apredizado de máquina, procura identificar instâncias relevantes e remover as instâncias que são redundantes ou prejudiciais do conjunto original. Classificadores baseados em instâncias, como o K Vizinhos Mais Próximos (*k*-NN), são fortemente beneficiados com esta seleção, podendo prover uma classificação mais rápida, uma diminuição nos requisitos de armazenamento e uma diminuição na sensibilidade ao ruído. Um fundamento essencial a esses algoritmos são as métricas de distância entre os exemplos. Nesse trabalho de mestrado, é proposto um algoritmo de seleção de instâncias com aprendizado de métricas denominado Seleção de Instância sobre Aprendizado de Métrica (*Instance Selection on Metric Learning, ISML*) para o Classificador K Vizinhos Mais Próximos. O método de aprendizado de métricas, chamado de *k-Neighborhood Components Analysis* (*kNCA*), é aplicado ao conjunto de dados para melhorar a seleção e reduzir a relação de compromisso (*trade-off*) entre número de instâncias de treino e acurácia. Foram realizados experimentos para comparar métodos tradicionais da literatura de seleção de instâncias. Os resultados são promissores principalmente em cenários de redução extrema de exemplos, redução maior que 50% dos dados originais, onde a proposta *ISML* obtém melhor ROC AUC em 11 dos 12 conjunto de dados quando comparado com outros três métodos de seleção de instância.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AM Aprendizado de Máquina

AUC *Area Under ROC Curve*

EM *Expectation Maximization*

FN Falso Negativo

FP Falso Positivo

IA Inteligência Artificial

Tr Conjunto de Treinamento

Te Conjunto de Teste

Tv Conjunto de Validação

ROC *Receiver Operating Characteristic*

TFP Taxa de Falso Positivo

TVP Taxa de Verdadeiro Positivo

VN Verdadeiro Negativo

VP Verdadeiro Positivo

k-NN k-Nearest Neighbor

NCA Neighbourhood Components Analysis

kNCA k-Neighborhood Components Analysis

ENN Edited Nearest Neighbour

CNN Condensed Nearest Neighbour

CHC Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation

RMHC Random Mutation Hill Climbing

FCNN Fast Condensed Nearest Neighbor Rule

GCNN Generalized Condensed Nearest Neighbour

ENN Edited Nearest Neighbour

IBL Instance-Based Learning

DROP Decremental Reduction Optimization Procedure

Lista de Algoritmos

CAPÍTULO

1

Introdução

Com a crescente demanda de extração e compreensão de grandes volumes de dados, a importância de Aprendizado de Máquina (AM) vem crescendo cada vez mais. Atualmente, são diversas as aplicações onde AM faz parte do estado da arte, como em reconhecimento de objetos a partir da visão computacional, diagnóstico médico, detecção de falhas, carros autônomos, locomoção de robôs e processamento de linguagem natural. Grande parte destes problemas são solucionados com algoritmos de aprendizado de máquina supervisionado que induzem um classificador. Em classificação, o objetivo é rotular exemplos não vistos durante o treinamento com classes pré-definidas (García et al., 2015).

Dentro de AM, existe a família dos métodos estatísticos, que geralmente são caracterizados pela representação do conhecimento através de modelos matemáticos. Entre os métodos estatísticos existe um grupo chamado de *Instance-Based Learning* (aprendizado baseado em instância), onde os exemplos são armazenados na íntegra, normalmente sem modificações ou generalizações; e uma função de distância determina quais instâncias do conjunto de dados são próximas ao novo exemplo a ser predito (Aha et al., 1991). A diferença entre esses algoritmos encontra-se na função de distância utilizada, número de exemplos utilizados para realizar predição, ponderação para influenciar os vizinhos e algoritmos eficientes como *KD-Tree* (García et al., 2015).

Um dos representantes mais conhecidos de algoritmos de aprendizado baseado em instância é o classificador supervisionado k-Vizinhos Mais Próximos (k-NN). O algoritmo é baseado em uma ideia simples e intuitiva: a classe da nova instância é a classe majoritária das k instâncias mais similares (Alpaydin, 2014). A popularidade do k-NN também se justifica devido à sua base

estatística forte. Cover and Hart (1967) demonstram que a taxa de erro do k-NN, denominada por R, é próxima da taxa de erro de *Bayes* R^*. A probabilidade de erro do k-NN é delimitada por $R^* \leq R \leq 2 \times R^*$, conforme o número de instâncias N e o valor k tendem ao infinito, ao considerar a condição $\frac{k}{N} \to 0$.

Além disso, diversos estudos relataram resultados promissores do k-NN em aplicações do mundo real, como estimar a biomassa de uma floresta utilizando dados de um laser (McRoberts et al., 2015) e predição de variáveis para usinas modernas de energia a carvão (Agrawal et al., 2016), assim sendo reconhecido como um dos algoritmos mais representativos de aprendizado de máquina (Wu et al., 2008; Chen et al., 2012).

Apesar do bom desempenho do k-NN, há alguns desafios e problemas em aberto. Uma desvantagem bem conhecida do k-NN é a sua dependência com a qualidade do conjunto de instâncias rotuladas (conjunto de treinamento), uma vez que o algoritmo k-NN não considera a importância de cada instância deste conjunto de treinamento (Brighton and Mellish, 2002). Assim, em cenários onde não ocorre o processo de seleção de instâncias, instâncias ruídosas e supérfluas podem ser utilizadas para definir a classe de um novo exemplo, o que tende a aumentar a taxa de erro. Nesse sentido, métodos de seleção de instâncias são úteis para identificar quais instâncias pertencentes ao conjunto de treinamento devem ser consideradas no Classificador k-Vizinhos Mais Próximos (García et al., 2015).

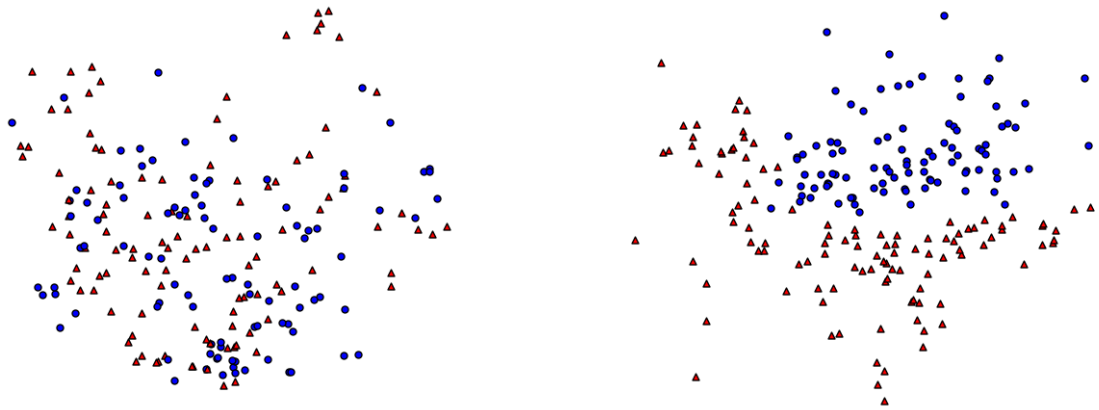
Existem diversas estratégias na literatura para seleção de instâncias, por exemplo, filtros de ruídos, métodos de condensação e métodos de seleção de protótipos (García et al., 2015). Filtros de ruídos removem instâncias cujo rótulo seja diferente dos rótulos dos seus vizinhos. Já os métodos de condensação mantêm as instâncias que estão próximas da borda de decisão e removem as instâncias internas que não influenciam na borda de decisão, trazendo pouco efeito na classificação. E finalmente, em métodos de seleção de protótipos, são métodos que esperam encontrar conjuntos de treinamento que oferecem melhores taxas de acurácia e de taxa de redução do conjunto usando classificadores que consideram uma certa semelhança ou medida de distância (García et al., 2015). As instâncias do conjunto de treinamento são organizadas em regiões de acordo com uma medida de distância. Protótipos (instâncias representativas) são selecionadas para cada região (regiões de alta densidade), assim representanto todo conjunto de treinamento com poucas instâncias.

Assim, os métodos de seleção de instâncias exploram essas estratégias, individualmente ou combinadas, para identificar um subconjunto representativo de instâncias D_s a partir de um conjunto de dados D, em que o desempenho P do classificador seja mantida, ou seja, $P(D) \cong P(D_s)$ (Reinartz, 2002;

Olvera-López et al., 2010; Leyva et al., 2013). É importante observar que em cenários em que há uma quantidade significativa de instâncias ruidosas no conjunto de treinamento, o processo de seleção de instâncias pode inclusive melhorar o desempenho do classificador.

O desempenho da maioria dos métodos de seleção de instâncias depende de uma medida de distância usada para identificar as instâncias vizinhas (Olvera-López et al., 2010; Brighton and Mellish, 2002; García et al., 2015). Ainda, a escolha da medida de distância é dependente dos dados e de um domínio particular de aplicação, sendo uma decisão não trivial (Wang and Sun, 2014). Assim, embora a definição de uma medida de distância adequada para um dado conjunto de dados pode melhorar significativamente a tarefa de seleção de instâncias, este fato não é investigado nos algoritmos existentes de seleção de instâncias, principalmente para o classificador k-NN.

Em vista disso, neste trabalho de mestrado é proposto e avaliado um método de seleção de instâncias com base em um processo de aprendizado de métrica denominado *Instance Selection on Metric Learning (ISML)*. A ideia básica do método *ISML* é que com a existência de uma medida de distância adequada para um determinado conjunto de dados, é possível identificar mais precisamente as instâncias que estão próximas das bordas de decisão entre as classes — que são as instâncias mais relevantes do problema de classificação do vizinho mais próximo.



(a) (b)

Figura 1.1: Conjunto de dados sonar (a) sem o aprendizado de métricas e (b) com o aprendizado de métricas

Para ilustrar o método *ISML*, a Figura 1.1a e 1.1b mostra o conjunto de dados "sonar" sem e com apredizado de métricas, respectivamente. No espaço original (Figura 1.1a), as instâncias das classes vermelho e azul ficam mis-

turadas. No espaço com aprendizado de métrica (Figura 1.1b), as instâncias da classe vermelha estão na região oposta à região da classe azul, facilitando assim a seperação das classes. A hipótese de pesquisa deste trabalho é fundamentada nesta observação. Com um espaço de instâncias mais "organizado" é mais provável que a borda de decisão entre as classes seja mais definida, necessitando de menos exemplos para representá-la.

1.1 Objetivos e Hipóteses

O principal objetivo desse trabalho de mestrado é propor um algoritmo de seleção de instâncias que seja agressivo na redução de exemplos, ou seja, que reduza o número de exemplos em mais de 50% do conjunto de exemplos original e que mantenha o desempenho de classificação similar ao classificador com 100% dos exemplos.

A hipótese de pesquisa considera que a junção de aprendizado de métricas com seleção de instâncias pode organizar melhor o espaço de instâncias, podendo representar o problema com menos exemplos que as abordagems tradicionais de seleção de instância.

1.2 Principais Contribuições

As principais contribuições desse trabalho são:

1. A proposta de um novo método de seleção de instâncias com aprendizado de métricas;

2. A avaliação experimental confrontando a proposta com os métodos tradicionais de seleção de instâncias;

3. A avaliação experimental do método proposto em cenários de uma seleção mais agressiva de instâncias; e

4. A ferramenta computacional desenvolvida.

Foi realizada uma extensa avaliação experimental para comparar o método proposto, *ISML*, com outros métodos tradicionais de seleção de instâncias em vários *benchmark* de conjuntos de dados. Os resultados mostram que o método *ISML* alcança redução significativa no conjunto de treinamento sem redução da acurácia do classificador em diferentes cenários. Em particular, nos cenários onde a estratégia agressiva de seleção de instâncias é exigida (ou seja, redução de mais de 50% das instâncias) o método proposto mostra o melhor resultado entre as abordagens comparadas.

1.3 Organização do Texto

O restante desta dissertação está organizada da seguinte maneira.

Capítulo 2 - Fundamentação. Nesse capítulo é apresentado conceitos e notações básicas de aprendizado de máquina. É também introduzido métricas para avaliar o desempenho dos classificadores, incluindo explicação do espaço *ROC* (*Receiver Operating Characteristic*) e curva *ROC*.

Capítulo 3 - Revisão Bibliográfica. Apresentação dos algoritmos tradicionais de seleção de instâncias.

Capítulo 4 - Proposta. Nesse capítulo é feita a descrição da proposta desse trabalho chamada de *ISML*. É apresentado o conceito de seleção de instâncias, introdução do conceito de aprendizado de métrica de distância, dos algoritmos de aprendizado de métrica *Neighbourhood Components (NCA)*, *k-Neighborhood Components Analysis (kNCA)*, por fim a proposta *ISML* e seu algoritmo.

Capítulo 5 - Experimentos. São apresentados os experimentos relacionados a seleção de instâncias utilizando medida de *Silhouette* modificada e experimentos com a medida de *AUC* e porcentagem de redução da proposta com outros métodos de seleção de instâncias da literatura.

Capítulo 6 - Conclusões. No capítulo de Conclusões são feitas as conclusões do trabalho, as contribuições e os trabalhos futuros.

CAPÍTULO

2

Fundamentação

2.1 Raciocínio Indutivo

A indução é uma forma de inferência lógica que permite obter conclusões genéricas sobre um conjunto particular de exemplos. Ela é caracterizada como o raciocínio que origina em um conceito específico e o generaliza, ou seja, da parte para o todo. Na indução, um conceito é aprendido efetuando-se inferência indutiva sobre os exemplos apresentados. Portanto, as hipóteses geradas por meio da inferência indutiva podem ou não preservar a verdade, e por isso que são chamadas de hipóteses (Rezende, 2003). Um exemplo de indução:

1. Sócrates é humano.
2. Todo ser humano é mortal.
3. Logo, Sócrates é mortal.

A maioria dos algoritmos de aprendizado de máquina utilizam a indução para resolver problemas. O aprendizado indutivo é realizado a partir de um conjunto de dados. Esse conjunto de dados é previamente entregue como informação disponível ao algoritmo de aprendizado para que ele possa gerar hipóteses. Segundo Mitchell (1997), um computador é dito que aprendeu uma experiência em relação a alguma classe de tarefas com medida de desempenho, se seu desempenho nas tarefas, melhora com as experiências. Dessa maneira, a experiência é representada pelo conjunto de dados.

Em geral para se ter um problema de aprendizagem bem definido, é necessário três caracteristicas; uma classe de tarefas, uma medida de desempenho

para melhorar e uma fonte de experiência. Por exemplo, seja o problema de classificação de pássaros; então a tarefa T é identificar pássaros, a medida de desempenho P é a porcentagem de acertos e a experiência E são pássaros já identificados.

Os algoritmos de aprendizado de máquina podem ser divididos de acordo com o uso ou não do atributo classe no conjunto de treinamento:

1. Supervisionado: No aprendizado supervisionado, as instâncias do conjunto de treinamento devem possuir um valor no atributo classe, ou também chamado, rótulo da instância;

2. Semissupervisionado: Já no aprendizado semissupervisionado, as instâncias do conjunto de treinamento além de possuir instâncias rotuladas, também devem conter instâncias não rotuladas. O número de instâncias com rótulos é normalmente inferior ao conjunto de instâncias não rotuladas; e

3. Não supervisionado: Em aprendizado não supervisionado, todos as instâncias do conjunto de treinamento não contêm rótulos;

Assim, a informação sobre o atributo classe define o tipo de aprendizado, supervisionado ou não supervisionado, a utilizar. A hierarquia do aprendizado indutivo é mostrado na Figura 2.1.

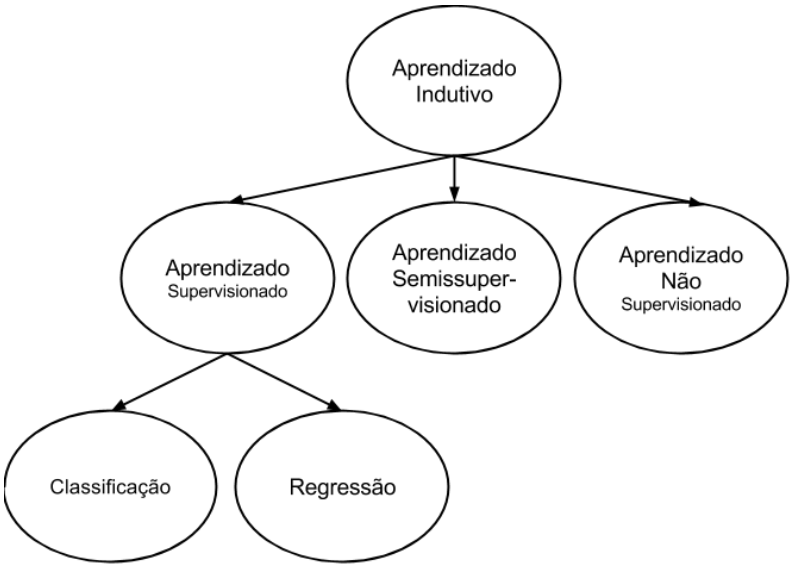


Figura 2.1: Hierarquia do aprendizado indutivo

2.1.1 Notação

Seja X definido pelo conjunto de dados e x_i, sendo $i = 1,...,n$, a representação das n instâncias. Para toda instância x_i existe a representação de um vetor

$A = (a_1, .., a_m, y)$, que a_t, sendo $t = 1, ..., m$, são os atributos e y referente ao seu atributo classe. A Tabela 2.1 ilustra a definição de um conjunto de dados.

	A_1	A_2	...	A_m	Y
x_1	a_{11}	a_{12}	...	a_{1m}	y_1
x_2	a_{21}	a_{22}	...	a_{2m}	y_2
$\vdots$	...	...	$\ddots$	...	$\vdots$
x_n	a_{n1}	a_{n2}	...	a_{nm}	y_n

Tabela 2.1: Formato de um conjunto de dados

Um exemplo de conjunto de dados é ilustrado na Tabela 2.2, contendo como atributos *Umidade*, *Temperatura* e *Chuva* como atributo classe.

X	A_1	A_2	Y	X	A_1	A_2	Y
	Umidade	Temperatura	Chuva		Umidade	Temperatura	Chuva
x_1	60%	25°C	sim	x_{11}	40%	20°C	não
x_2	65%	26°C	sim	x_{12}	43%	22°C	não
x_3	50%	24°C	sim	x_{13}	57%	24°C	não
x_4	73%	27°C	sim	x_{14}	51%	22°C	não
x_5	55%	24°C	sim	x_{15}	59%	23°C	não
x_6	52%	24°C	sim	x_{16}	48%	22°C	não
x_7	69%	27°C	sim	x_{17}	38%	19°C	não
x_8	71%	27°C	sim	x_{18}	47%	23°C	não
x_9	70%	27°C	sim	x_{19}	43%	20°C	não
x_{10}	67%	26°C	sim	x_{20}	55%	25°C	não

Tabela 2.2: Exemplo de Conjunto de dados

Em aprendizado de máquina existem três tipos de conjunto de dados que podem ser distinguidos: o conjunto de treinamento, conjunto de teste e o conjunto de validação. O conjunto de treinamento, definido por T_r, é o conjunto de dados apresentado ao algoritmo de aprendizado *a priori*, ou seja, é o conjunto de dados utilizado para gerar a hipótese do classificador. Já o conjunto de teste, definido por T_e, é utilizado para medir a qualidade da hipótese, esse conjunto de dados deve ser disjunto do conjunto de treinamento. E por fim o conjunto de validação, definido por T_v, é utilizado em métodos de estimativa de acurácia. É uma parte do conjunto de dados que não está contida no conjunto de treinamento e também não está no conjunto de teste. Alguns algoritmos geram hipótese que "decoram" o conjunto de treinamento utilizando como base o resultado da classificação no conjunto de teste e o conjunto de validação serve para validar o desempenho.

Segundo Mitchell (1997), o aprendizado ocorre na aquisição de informação a partir de um conjunto de dados, para a indução de uma hipótese representada por uma função. Em geral a função a ser aprendida é definida por f, sendo $f : X \rightarrow \{f(x)\}$. Seja f a função a ser aprendida, dado um conjunto de

treinamento T e H o conjunto de todas as possíveis hipóteses que o aprendizado deve levar em conta sobre o conceito alvo f. O objetivo do aprendizado é encontrar um $h \in H$, no qual $h(x) \cong f(x)$ para todo $x \in T$ (Mitchell, 1997). Como pode ser visto na Figura 2.2.

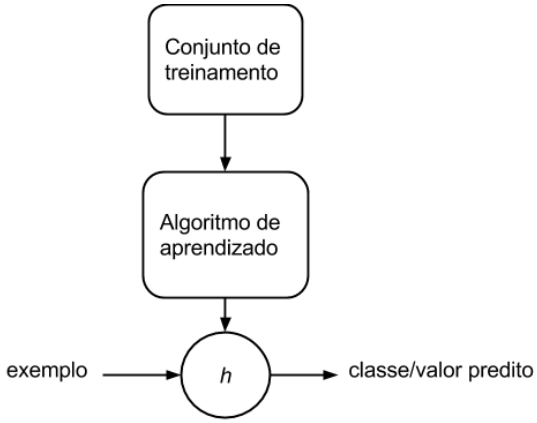


Figura 2.2: Fluxograma generalizado de um algoritmo de aprendizado

O domínio da função f é definido por y. Caso o domínio de $y \in \mathbb{R}$, o problema se torna de regressão, com $y \in \{c_1, c_2, .., c_r\}$ o problema se torna de classificação. Na Tabela 2.2 o rótulo "sim" é a ocorrência de chuva e "não" é a não ocorrência de chuva, então o problema é de classificação binária com $y \in \{\text{sim,não}\}$.

Mesmo que o objetivo do aprendizado seja determinar uma hipótese h similar a função f, a única informação sobre f está apenas disponível no conjunto de treinamento (Mitchell, 1997).

Outras definições importantes sobre o processo de aprendizado de máquina são apresentadas a seguir:

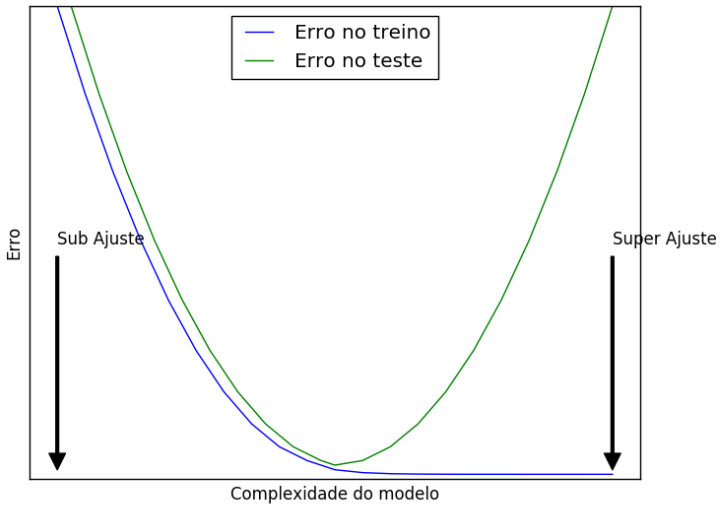


Figura 2.3: Taxa de erro de acordo com Complexidade do modelo

Super Ajuste (*Overfitting*): Ocorre quando no processo de treinamento do classificador, ou quando é induzido, o classificador acaba "decorando" o conjunto de treinamento. Ao "decorar", o classificador não consegue generalizar, fica restrito de acertar a maioria as instâncias que ocorrem naquele conjunto de treinamento e ao apresentar um conjunto de dados nunca visto, ocorre alto taxa de erro. Na Figura 2.3 mostra a região que ocorre o Super Ajuste, onde a complexidade do modelo é alta, com baixa taxa de erro no treino e alta taxa de erro no teste.

Sub Ajuste (*Underfitting*): Do contrário do que ocorre com *Overfitting*, o classificador gera uma hipótese sendo não capaz de "aprender" o próprio conjunto de treinamento. Ocorrendo alto índice de erro no conjunto de treinamento e no conjunto de teste. A Figura 2.3 mostra quando ocorre Sub Ajuste, onde a complexidade do modelo é muito baixa.

Validação Cruzada (*Cross-Validation*): É um método de amostragem para avaliação do algoritmo. Consiste em dividir o conjunto de dados em k-partições (*k-folds*), onde k é informado pelo usuário, e avaliar o algoritmo utilizando $k-1$ partições como conjunto de treinamento, e a partição remanescente como conjunto de teste. Esta avaliação é repetida k vezes, variando o conjunto de testes e consequentemente o conjunto de treinamento.

2.1.2 Aprendizado Supervisionado

Como dito anteriormente o aprendizado supervisionado utiliza instâncias do conjunto de treinamento com o atributo classe rotulado. Um exemplo de algoritmo para um algoritmo supervisionado é o K-Vizinhos Mais Próximos (*k-Nearest Neighbor*, *KNN*).

K-Vizinhos Mais Próximos

Proposto em Cover and Hart (1967), segundo Mitchell (1997) o K-Vizinhos Mais Próximos é o mais básico algoritmo baseado em instâncias. Assume que todas as instâncias estão em um plano *m-dimensional*, sendo m o número total de atributos, e utilizando uma função de distância para determinar o quão próximo uma instância está da outra, classifica as instâncias pela classe majoritária dos k vizinhos mais próximos. As instâncias armazenadas (*stored instance*), ou o conjunto de treinamento, é o conjunto de instâncias que serão comparadas para a classificação de uma nova instância. O processo de aprendizado do *KNN* é utilizar o conjunto de treinamento para comparação. A classificação da nova instância é definida pela classe majóritária dos k "vizinhos" (instâncias próximas).

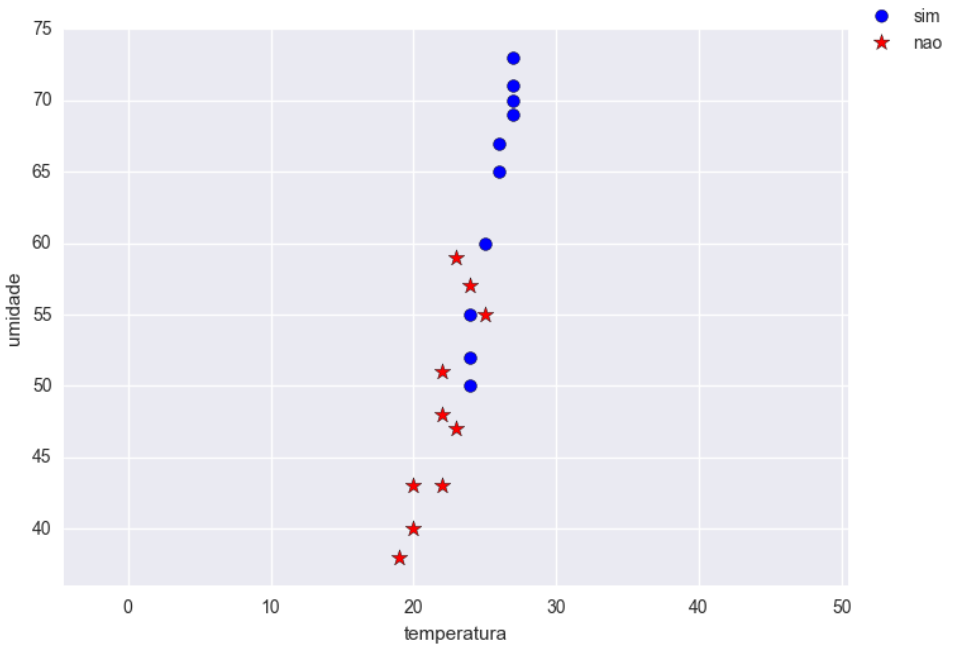


Figura 2.4: Representação do espaço 2D do Conjunto ilustrado na Tabela 2.2 do *KNN*

Utilizando o conjunto de dados da Tabela 2.2 como exemplo, a Figura 2.4 representa em um plano cartesiano as posições das instâncias, sendo o eixo x representado por *umidade* e y por *temperatura*. A distância padrão utilizada é a distância *Euclidiana* descrita na Equação 2.1, no qual mede a distância entre duas instâncias x_i e x_j,

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{m} (a_r(x_i) - a_r(x_j))^2}, \tag{2.1}$$

onde $a_r(x_i)$ devolve o atributo r de x_i. Essa distância é utilizada para determinar as k instâncias mais próximas da instância a ser classificada. Contendo o conjunto de treinamento T_r, o valor de k e a fórmula da distância, o algoritmo consegue realizar classificação de instâncias. Essa classificação é realizada calculando todas as distâncias a partir da instância a ser classificada x_q, instância *query* (instância de consulta), para todas as instâncias contidas no conjunto de treinamento, ou seja, $d(x_q, x_i)$ sendo $i = 1, .., n$ e n sendo o tamanho do conjunto de treinamento. Logo a classe majoritária, das k instâncias mais próximas da instância x_q, é atribuída como a classe da instância x_q. O algoritmo é descrito a seguir.

Exemplo

Como exemplo, seja definida a Tabela 2.2 como conjunto de treinamento do *KNN*. Como são dois atributos que compõem a representação das instâncias,

Algoritmo 1: Algoritmo de classificação do K-Vizinhos Mais Próximos

Entrada: Conjunto de Treinamento $T_r = x_1, ..., x_n$, instância de consulta x_q e constante k

Saída: Classe da instância x_q

1 **para todo** $x_j \in L_t$ **faça**

2 Calcular a distancia de x_j a x_q

3 Devolver classe majoritária entre as k instâncias mais próximas de x_q

sem incluir o atributo classe, o conjunto de treinamento pode ser visto em um plano cartesiano contendo informações de seus atributos representados pelos eixos, visto anteriormente na Figura 2.4. Seja definido x_q a instância de consulta (*query*) contendo os valores dos atributos *umidade* de 54% e *temperatura* de 26°C. A instância x_q é representada pelo ponto preto na Figura 2.5a . O resultado dos calculos estão presentes na Tabela 2.3:

Instância	Fórmula	Resultado
$d(x_{20}, x_q)$	$\sqrt{(25-26)^2+(55-54)^2}$	1.4
$d(x_5, x_q)$	$\sqrt{(24-26)^2+(55-54)^2}$	2.2
$d(x_6, x_q)$	$\sqrt{(24-26)^2+(52-54)^2}$	2.8
$d(x_{13}, x_q)$	$\sqrt{(24-26)^2+(57-54)^2}$	3.6
$d(x_3, x_q)$	$\sqrt{(24-26)^2+(50-54)^2}$	4.5
$d(x_{14}, x_q)$	$\sqrt{(22-26)^2+(51-54)^2}$	5.0
$d(x_{15}, x_q)$	$\sqrt{(23-26)^2+(59-54)^2}$	5.8
$d(x_1, x_q)$	$\sqrt{(25-26)^2+(60-54)^2}$	6.1
$d(x_{16}, x_q)$	$\sqrt{(22-26)^2+(48-54)^2}$	7.2
$d(x_{18}, x_q)$	$\sqrt{(23-26)^2+(47-54)^2}$	7.6
$d(x_2, x_q)$	$\sqrt{(26-26)^2+(65-54)^2}$	11.0
$d(x_{12}, x_q)$	$\sqrt{(22-26)^2+(43-54)^2}$	11.7
$d(x_{19}, x_q)$	$\sqrt{(20-26)^2+(43-54)^2}$	12.5
$d(x_{10}, x_q)$	$\sqrt{(26-26)^2+(67-54)^2}$	13.0
$d(x_7, x_q)$	$\sqrt{(27-26)^2+(69-54)^2}$	15.0
$d(x_{11}, x_q)$	$\sqrt{(20-26)^2+(40-54)^2}$	15.2
$d(x_9, x_q)$	$\sqrt{(27-26)^2+(70-54)^2}$	16.0
$d(x_8, x_q)$	$\sqrt{(27-26)^2+(71-54)^2}$	17.0
$d(x_{17}, x_q)$	$\sqrt{(19-26)^2+(38-54)^2}$	17.5
$d(x_4, x_q)$	$\sqrt{(27-26)^2+(73-54)^2}$	19.0

Tabela 2.3: Cálculo da distância entre a instância de consulta (x_q) e o conjunto de treinamento.

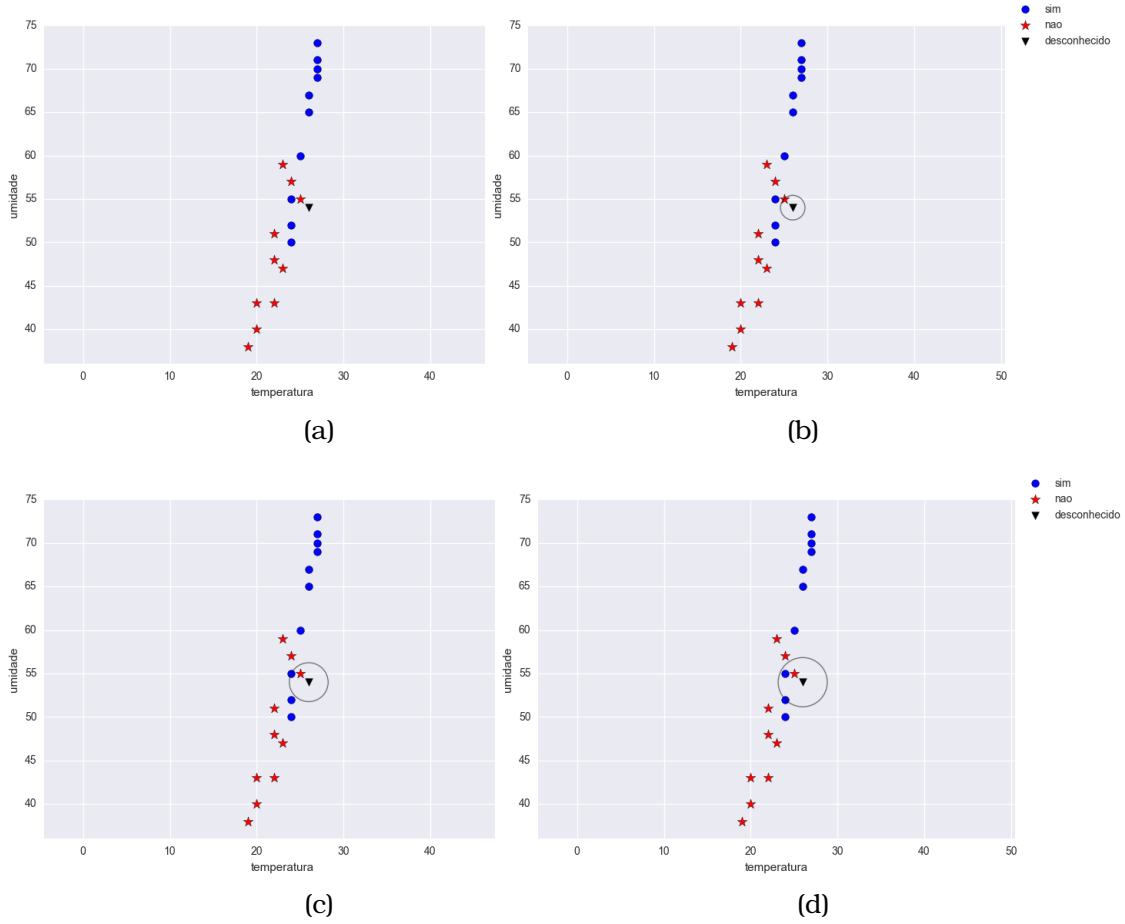


Figura 2.5: Representação do espaço 2D da Tabela 2.2 e uma instância sem classificação com k igual a (b) 1, (c) 2 e (d) 3

Suponha que o k é definido igual a 1. Nessa configuração a instância mais próxima da instância de consulta x_q é a instância de número 19, com a distância Euclidiana igual a 1.4, mostrado na Figura 2.5b. Resultando como a classificação de x_q igual a classe "não".

Já com k igual a 2, acontece um empate, as instâncias de número 19 e 4 são as mais próximas, porêm o atributo classe de ambos são diferentes, mostrado na Figura 2.5c.

E com k igual a 3, as intâncias mais próximas são a 19, 4 e 5. Nessa configuração a classe majoritária é a classe "sim", mostrado na Figura 2.5d.

2.1.3 Aprendizado Não Supervisionado

No aprendizado não supervisionado não se tem informação do atributo classe, ou rótulo, das instâncias no qual se deseja realizar o treinamento. Um exemplo de algoritmo não supervisionado é o *k-Means*.

k-Means

A principal proposta em MacQueen (1967), sobre o *k-Means*, é realizar o particionamento de uma população *m-dimensional* em *k* conjuntos. O algoritmo necessita que o usuário informe a quantidade de grupos que se deseja separar, ou seja, o valor *k* de partições. O *k-Means* utiliza uma medida de distância como medida para calcular a dissimilaridade entre os grupos. No Algoritmo 2.1.3 é apresentado o pseudo-código de classificação do *k-Means*:

Algoritmo 2: Algoritmo de classificação do *k-Means*

Entrada: Conjunto de dados $D = x_1, ..., x_n$ e número de partições k
Saída: Partições em k grupos $P = C_1, C_2, .., C_k$

```
1 Gerar k centros aleatoriamente em um conjunto C_j, sendo j = 1,..,k
2 para ∀C_j não tenha convergido faça
3     para todo x_i ∈ D faça
4         Atribuir x_i ao grupo C_i mais próximo.
5     Atualização dos centróides
```

Dado um conjunto de dados D e um valor k igual a 2, o algoritmo irá gerar k pontos que serão inicializados em locais aleatórios e serão definidos como centroides. Para cada instância do conjunto D é calculado a distância com os centroides e associado ao centroide mais próximo. Após a associação de todas as instâncias, é realizado o cálculo da nova posição dos centroides baseado nas instâncias associadas a cada um. E esse processo é repetido enquanto a posição dos centroides continuarem em mudança. Após algumas interações os centroides podem permanecer em uma única posição, ou não irá ocorrer variação de suas posições, e então o processo é finalizado.

Um dos problemas desse algoritmo são os centroides finais que nem sempre vão representar uma solução ótima. Ocorre por que a primeira posição dos centroides são gerados aleatoriamente. Uma forma para resolver esse problema é executando o algoritmo diversas vezes e selecionar o melhor resultado.

2.1.4 Aprendizado Semissupervisionado

O aprendizado semissupervisionado utiliza tanto instâncias rotuladas e não rotuladas para o seu treinamento. Um exemplo de algoritmo semissupervisionado é o *COP-k-Means*.

COP-k-Means

O *COP-k-means* é um algoritmo proposto por Wagstaff et al. (2001) no qual é uma modificação do algoritmo *k-Means*. Segundo Wagstaff et al. (2001) o

COP-k-means utiliza, no que ele diz, um conhecimento de fundo (*background knowledge*), no qual são restrições iniciais do nível de instâncias durante o processo na formação dos grupos (*clusters*). No contexto de algoritmos para particionamento, restrições de nível de instâncias são úteis ao algoritmo para decidir qual grupo a instância deve fazer parte. Wagstaff et al. (2001) definiu dois tipos de restrições de par:

- *Must-link*: Restrições que especificam que duas instâncias devem estar no mesmo grupo (*cluster*); e
- *Cannot-link*: Restrições que especificam que duas instâncias não devem estar no mesmo grupo.

A principal modificação é que ao atualizar os grupos, na atribuição, garantir que nenhuma restrição *must-link* e *cannot-link* imposta pelo usuário seja violada. A atribuição de alguma instância x é realizada ao grupo C_j mais próximo. A atribuição não ocorre apenas se alguma restrição for violada. A seguir o algoritmo é descrito:

Algoritmo 3: Algoritmo de classificação do *COP-k-Means*

Entrada: Conjunto de dados $D = x_1, ..., x_n$, seja K definido pelas restrições must-link e restrições cannot-link e seja C_i $i = 1, .., k$ os conjuntos iniciais dos centróides
Saída: Devolver o conjunto C

```
1 enquanto ∀C_i tenha convergido faça
2     para todo x_j ∈ D faça
3         Atribuir x_j ao grupo C_i mais próximo, sendo que as restrições K
          não sejam violadas.
4     Atualização dos centróides
```

2.2 Métricas de Avaliação de Classificadores

Para um estudo analítico de classificadores, é necessário realizar comparação entre classificadores em um mesmo conjunto de dados. A seguir, são introduzidas as métricas que expressam características dos classificadores.

2.2.1 Desempenho do Classificador

Dado um classificador binário e uma instância a ser classificada, existem quatro possíveis saídas de classificação. A instância que é positiva e foi classificada como positiva, é chamada de *true positive* (verdadeiro positivo), ou se foi classificada como negativa, é chamada com nome de *false negative* (falso negativo). O mesmo ocorre para as instâncias negativas. Se a instância é negativa

	preditos positivos	preditos negativos	
exemplos positivos	VP	FN	Pos
exemplos negativos	FP	VN	Neg
	PPos	PNeg	Total

Tabela 2.4: Matriz de Confusão

e foi classificada como negativa, ela é *true negative* (verdadeiro negativo), ou caso ela foi classificada como positiva, ela é *false positive* (falso positivo) (Fawcett, 2006). Com um classificador e um conjunto de instâncias, uma matriz de confusão (ou tabela de contingência) pode ser construída (ver Tabela 2.4). Nessa matriz é apresentada a contagem dos exemplos verdadeiros positivos (VP), falsos positivos (FP), verdadeiros negativos (VN) e falsos negativos (FN). Além do total de positivos (Pos), total de negativos (Neg), total preditos positivos (PPos) e total preditos negativos (PNeg). Com esses números é possível calcular:

- $precisão = \frac{VP}{PPos}$
- $acurácia = \frac{VP+VN}{Total}$
- *recall* ou *tp-rate* $= \frac{VP}{Pos}$
- *fp-rate* $= \frac{FP}{Neg}$
- *f-measure* $= \frac{2 \times precisão \times recall}{(precisão + recall)}$

Cada métrica, citada acima, expressa pontos fortes do resultado do classificador, como por exemplo:

tp-rate ou TVP: Expressa a qualidade da classificação da classe positiva.

fp-rate ou TFP: Expressa a qualidade da classificação da classe negativa como uma taxa de erro, quanto menor melhor a classificação da classe negativa.

acurácia: Representa a proporção de acertos tanto das classes positiva e negativa.

Exemplo #1 de Matriz de Confusão

Seja a Tabela 2.5 exemplo de uma matriz de confusão, onde a proporção de exemplos positivos e exemplos negativos sejam iguais.

	preditos positivos	preditos negativos	
exemplos positivos	400	100	500
exemplos negativos	200	300	500
	600	400	1000

Tabela 2.5: Exemplo de uma Matriz de Confusão de um Classificador A

- *precisão* $= \frac{400}{600} = 0.67$
- *acurácia* $= \frac{400+300}{1000} = 0.70$
- *tp-rate* $= \frac{400}{500} = 0.80$
- *fp-rate* $= \frac{200}{500} = 0.40$

Ao analisar os valores, é possível perceber que o valor de *tp-rate* é alto, mostrando que a maioria dos exemplos positivos foram classificados corretamente. Já o valor de *fp-rate* mostra que quase metade dos exemplos negativos foram classificados errados. Apesar do valor de *tp-rate* ter sido alto, a *precisão* foi baixa, ou seja, apenas 67% dos preditos positivos foram corretos.

Exemplo #2 de Matriz de Confusão

A Tabela 2.6 representa um outro exemplo de matriz de confusão com 13% de exemplos positivos e 87% de exemplos negativos.

	preditos positivos	preditos negativos	
exemplos positivos	50	50	100
exemplos negativos	100	550	650
	150	600	750

Tabela 2.6: Exemplo de uma Matriz de Confusão de um Classificador B

- *precisão* $= \frac{50}{150} = 0.33$
- *acurácia* $= \frac{50+550}{750} = 0.80$

- $tp\text{-}rate = \frac{50}{100} = 0.50$

- $fp\text{-}rate = \frac{100}{650} = 0.15$

Já no exemplo da Tabela 2.6, o motivo pelo qual a *acurácia* ter sido alta, foi a grande parte dos exemplos negativos terem sido classificados corretamente. O valor de *tp-rate* revela que metade dos exemplos positivos foram classificados errados. A *precisão* mostra que apenas 33% de todos preditos positivos foram classificado corretamente. Apesar do classificador B lidar bem com exemplos negativos, o classificador erra cerca de 50% dos exemplos positivos.

2.2.2 Espaço Receiver Operating Characteristic (ROC)

O gráfico *ROC* é um espaço bidimensional, onde no eixo y representa o valor *tp-rate* e no eixo x é o valor *fp-rate*. Na Figura 2.7 é ilustrado um gráfico *ROC*. Nos pontos (0,0) e (1,1), representam classificadores livres de treinamento, sempre negativo e sempre positivo. O ponto (0,1) representa o classificador ideal e o ponto (1,0) é um classificador com todas as classes invertidas. Os pontos que compõem (0,0) a (1,1) representam os classificadores aleatórios (Flach, 2003).

Sejam as Tabelas 2.5 e 2.6 matrizes de confusão do classificador A e classificador B respectivamente, o Gráfico 2.6 mostra a posição que ocupam no gráfico *ROC*.

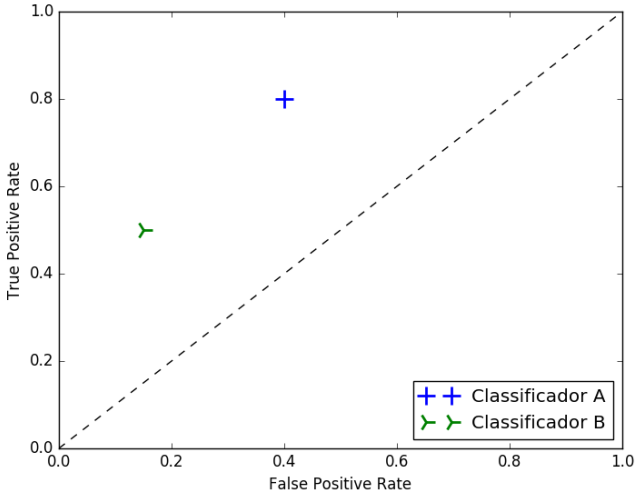


Figura 2.6: Pontos do Classificador A e Classificador B no espaço *ROC*

2.2.3 Curva ROC

Muitos classificadores produzem uma saída binária, sim ou não. E quando um classificador discreto, que atribui a classe diretamente, é apresentado a um conjunto de teste, é obtido uma matriz de confusão que corresponde a um ponto no gráfico *ROC*. Já os classificadores probabilísticos naturalmente produzem uma probabilidade ou *score*, valor numérico que representa o grau da instância pertencer a determinada classe, para cada instância (Flach, 2003).

Os classificadores probabilísticos podem utilizar um limiar (*threshold*) para produzir um classificador discreto. Ou seja, se a saída do classificador for maior do que o limiar, a classificação é da classe positiva, caso contrário é da classe negativa. Dessa forma, cada valor de limiar produz um ponto diferente no espaço *ROC*.

Seja a Tabela 2.7 resultado de um classificador probabilístico. Nela contêm 10 exemplos, sendo 5 positivos e 5 negativos, sua classe e *score* gerado por um classificador.

Instância	Classe	*Score*	Instância	Classe	*Score*
#1	y	0.993	#6	n	0.007
#2	y	0.993	#7	y	0.007
#3	y	0.993	#8	n	0.007
#4	y	0.993	#9	n	0.007
#5	n	0.57	#10	n	0.007

Tabela 2.7: Mostra *score* atribuído a cada instância do conjunto de treinamento por um classificador

Considerando o limiar igual a 0.5, é obtido uma acurácia de 80% com 8 exemplos classificados corretamente. É possível notar que esse valor de limiar obtém um valor de acurácia subótimo. Quando é utilizado um limiar igual a 0.6 é obtido uma acurácia de 90% com 9 exemplos classificados corretamente. É importante notar que a cada limiar diferente é produzido outro classificador e resultando um ponto diferente no espaço *ROC*.

Uma forma de gerar a curva *ROC* é variar o limiar de $-\infty$ até $+\infty$ e traçar a curva através do espaço *ROC*, porém é a pior forma para gerar a curva *ROC*. O Algoritmo 4 mostra uma forma eficiente de gerar curvas *ROC*.

Para gerar a curva *ROC* da Tabela 2.7 as instâncias foram ordenadas pelo seu *score*. Como nesse exemplo contém 10 instâncias da classe negativa e 10 da classe positiva a serem classificadas, o gráfico *ROC* a cada acerto e erro irá andar $\frac{1}{10} = 0.1$ no gráfico. Começando do ponto (0, 0) do gráfico *ROC*, o exemplo com o maior *score* para o menor, caso a sua classe seja positiva, o gráfico irá andar 0.1 para cima e caso contrário será 0.1 para direita.

A Figura 2.7 ilustra a curva *ROC* gerada a partir da Tabela 2.7. Como a tabela gerada é ordenada pelo *score* das instâncias, a tabela perfeita seria

Algoritmo 4: Algoritmo para a geração de pontos *ROC* (Flach, 2003)

Entrada: Conjunto de exemplos, L; probabilidade do classificador do exemplo i ser positivo, $f(i)$; número de exemplos positivos P e negativos N.

Saída: Lista de pontos da curva *ROC*, R.

```
1  L_sorted = ordena L decrescente pelas probabilidades f  FP = 0
2  TP = 0
3  R = {}
4  f_prev = −∞
5  i = 1
6  enquanto i ≤ |L_sorted| faça
7      se f(i) ≠ f_prev então
8          R = R + {FP/N, TP/P}
9          f_prev = f(i)
10     se L_sorted[i] = exemplo positivo então
11         TP = TP + 1
12     senão
13         FP = FP + 1
14     i = i + 1
15 R = R + {FP/N, TP/P}
```

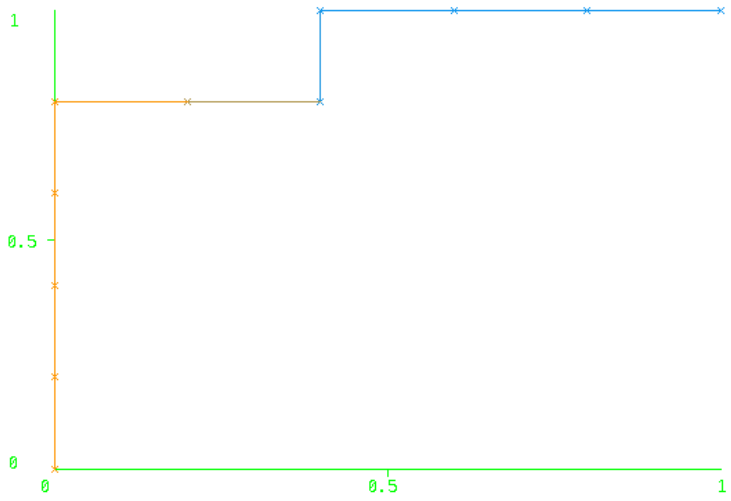


Figura 2.7: Curva *ROC* do resultado da Tabela 2.7

todas as primeiras *Pos* instâncias da classe positiva. Dessa forma a curva *ROC* não iria gerar uma concavidade próximo ao ponto (0,1). Porém o pior caso é quando a curva *ROC* é próxima da reta de (0,0) até (1,1), pois são casos em que o classificador gera resultados aleatórios.

CAPÍTULO

3

Revisão Bibliográfica

Nesse capítulo de revisão bibliográfica foi realizada a revisão de alguns dos principais algoritmos de seleção de instâncias da literatura. A seguir a Tabela 3.1 onde contêm um resumo de métodos, seguido por seu tipo (*wrapper* e *filter*), seleção baseada em alguma característica e a sua referência.

3.1 Condensed Nearest Neighbour

O *Condensed Nearest Neighbour* (*CNN*), proposto por PE (1968), é um método incremental que durante a classificação das instâncias seleciona instâncias de cada classe. Seja T o conjunto composto pelo conjunto de dados que se deseja realizar a seleção de instâncias. Seja também o conjunto S inicialmente vazio. Inicialmente é realizada a seleção aleatória de uma instância de cada classe contida em T para S. A partir dessa separação inicial, é realizada classificação de cada instância em T utilizando como conjunto de treinamento o conjunto S. Caso a classificação dessa instância seja errada, essa instância é adicionada ao conjunto S. No final as instâncias contidas em S, são as instâncias selecionadas como melhores. Ou seja, *CNN* seleciona instâncias que removidas não levam a classificação errada, consiste em achar um subconjunto mínimo de T.

3.2 Fast Condensed Nearest Neighbor Rule

Proposto por Angiulli (2005), o algoritmo *Fast Condensed Nearest Neighbor Rule* (*FCNN*) inicializa S com os centróides das classes obtidas no conjunto

Método	Tipo	Baseado em	Referência
CNN	W	Erros na classificação	PE (1968)
SNN	W	Erros na classificação	Ritter et al. (1975)
GCNN	W	Erros na classificação	Chou et al. (2006)
ENN	W	Erros na classificação	Wilson (1972)
All k-NN	W	Erros na classificação	Tomek (1976)
Multiedit	W	Erros na classificação	Devijver and Kittler (1980)
IB	W	Erros na classificação	Aha et al. (1991)
DROP	W	Associados	Wilson and Martinez (2000)
ICF	W	Acessível, Cobertura	Brighton and Mellish (2002)
SV-kNNC	W	SVM, k-means	Srisawat et al. (2006)
Evolutionary algorithms	W	Evolução Natural	Kuncheva (1995)
Memetic algorithms	W	Algoritmos evolutivos, Busca local	García et al. (2008)
CSA	W	Sistemas imunes artificiais	Garain (2008)
TS	W	Busca *Tabu*	Cerveron and Ferri (2001)
BSE, RFOS	W	Busca sequêncial	Olvera-López et al. (2005)
POP	F	Fraqueza, Instâncias de fronteira	Riquelme et al. (2003)
POC-NN	F	Média da classe oposta, Instâncias de fronteira	Raicharoen and Lursinsap (2005)
kd-trees	F	kd-trees	Narayan et al. (2006)
GCM	F	Clustering	Mollineda et al. (2002)
NSB	F	Clustering	Veenman and Reinders (2005)
CLU	F	Clustering, Instâncias interiores	Lumini and Nanni (2006)
OSC	F	Clustering, Instâncias interiores e de fronteira	Olvera-López et al. (2007)
WP	F	Peso da Instância	Paredes and Vidal (2000)
PSR	F	Relevância da Instância	Olvera-López et al. (2008)

Tabela 3.1: Características de métodos (Olvera-López et al., 2010)

Algoritmo 5: Algoritmo do *CNN*

Entrada: Conjunto de dados $T = x_1, ..., x_n$
Saída: Conjunto de dados S

```
1 S = ∅
2 S = seleciona aleatóriamente uma instância de cada classe
3 para todo x ∈ T faça
4     Gera uma nova hipótese do knn com conjunto de treinamento S
5     pred = knn(S)
6     se pred(x) ≠ classe(x) então
7         S = S + x
8         T = T − x
```

de treinamento T. Na interação do algoritmo, toda instância p contida em S representa uma célula de *Voronoi*. Para toda instância q que é próxima da célula de p, ou seja, são mais próximas de p do que outra qualquer instância em S, e de classe oposta de p são selecionadas e adicionadas em S.

Uma célula de *Voronoi* da instância p pertencente em S é o conjunto de todas as instância que são mais próximas de p do que outro p' em S. O algoritmo para de executar quando não tem mais instância para ser adicionada em S, ou seja, quando T é corretamente classificado utilizando S.

Algoritmo 6: Algoritmo do *FCNN*

Entrada: Conjunto de dados $T = x_1, ..., x_n$
Saída: Conjunto de dados S

```
1 S = ∅
2 ΔS = Centroides(T)
3 enquanto ΔS ≠ ∅ faça
4     S = S ∪ ΔS
5     ΔS = ∅
6     para todo p ∈ S faça
7         ΔS = ΔS ∪ {rep(p, Voren(p, S, T))}
```

Seja a função $Vor(p,S,T)$ definida pelo conjunto $\{q \in T \mid \forall p' \in S,\ d(p,q) \leq d(p',q)\}$, isso é o conjunto de elementos de T que são mais próximos de p do que qualquer outro elemento p' em S. E a notação de $Voren(p,S,T)$ o conjunto $\{q \in Vor(p,S,T) \mid l(q) \neq l(p)\}$, sendo $l(p)$ a denotação da classe de p. A função $Centroides(T)$ retorna o conjunto contendo os centroides de cada classe em T. A representação $rep(p,X)$ pode ser definida de diferentes maneiras. A primeira definição, chamada de *FCNN1*, variante do *FCNN*, utiliza essa definição, seleciona o vizinho mais próximo de p em X, isso é $rep(p,X) = nn(p,X)$. Sendo que, a função $nn(p,X)$ retorna o vizinho mais próximo de p em X. A segunda definição, chamada de *FCNN2*, seleciona o centróide das classes em X mais próxima de p, isso é, $rep(p,X) = nn(p,Centroides(X))$. Seja Figura 3.1 resultado da execução do *FCNN1*.

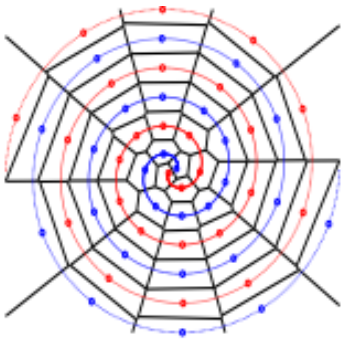


Figura 3.1: Exemplo do resultado da execução do FCNN1 (Angiulli, 2005)

3.3 Generalized Condensed Nearest Neighbour

Um outro variante do *CNN* é o *Generalized Condensed Nearest Neighbour* (GCNN) que é idêntico ao *CNN*, porém *GCNN* inclui instâncias em S caso satisfaça um critério de *absorption* (absorção) de acordo com um *threshold* (limiar). Para toda instância, a absorção é computada baseado nos vizinhos próximos e nas instâncias próximas de classe diferente da instância sendo comparada, ou da classe inimiga. Para uma instância p que pertence a T é absorvida se $|p-x|-|p-w| > sigma$ onde x e w pertencem a S, e x é o vizinho mais próximo e w é o vizinho mais próximo da classe inimiga de p, isto é, classe diferente de p.

3.4 Edited Nearest Neighbour

Proposto em Wilson (1972), é um método de seleção de instâncias *Edited Nearest Neighbour* (*ENN*), que tem o foco na remoção de instâncias com ruídos do conjunto de treinamento T. O *ENN* remove as instâncias de T quando a classe majoritária do *KNN* diverge de sua classe, *ENN* utiliza k igual a três.

3.5 Algoritmos Instance-Based Learning

A seção do algoritmo *IBL* tem como estudo o artigo Aha et al. (1991). As linhas destacadas nos algoritmos descritos significam a diferença entre os algoritmos do *IBL*.

Segundo Aha et al. (1991), os algoritmos *Intance-Based Learning* (*IBL*) são derivados a partir do classificador *KNN* Cover and Hart (1967). E os algoritmos *IBL* são muito similares ao *ENN* (que também salva e utiliza somente instâncias selecionadas para gerar predições de classificação). As diferenças entre os algoritmos *ENN* e *IBL* são que, o algoritmo *ENN* não é incremental e o seu principal objetivo é manter a consistência com conjunto de treinamento inicial. *ENN* diminui os dados, porém, não tenta maximizar a acurácia da classificação sobre novas instâncias. Comprometendo problemas do mundo real. Os algoritmos *IBL* são incrementais e um dos principais objetivos inclui maximizar a acurácia da classificação em instâncias apresentadas *a posteriori*.

A principal saída do *IBL* é a descrição do conceito (*concept description* ou *CD*). Que é uma função que dado uma instância a partir do espaço de instâncias, isso produz uma classificação, que é o valor predito do atributo classe da instância. Uma *CD* inclui um conjunto armazenado de instâncias e alguma informação sobre a performance passada durante classificação, ou seja, número correto e incorreto de previsões da classificação. Sendo que esse conjunto de instâncias pode ser mutável após cada instância do conjunto de treinamento

ser processada.

São três componentes que descrevem todos os algoritmos *IBL*, são eles:

1. Função de Similaridade: Calcula a similaridade entre a instância de treinamento i e a instância na descrição do conceito.

2. Função de Classificação: Utiliza o resultado da função de similaridade e armazena a performance da classificação da instância na descrição do conceito. Isso produz a classificação para i.

3. Atualizador da Descrição do Conceito (*Concept Description Updater*): Mantém armazenado no desempenho da classificação e escolhe qual instância é adicionada na Descrição do Conceito (CD). De entrada inclui i, os resultados da similaridade, resultado da classificação e a própria Descrição de Conceito. Como resultado, deve gerar uma nova Descrição do Conceito.

3.5.1 Algoritmo IB1

O algoritmo *IB1* é o mais simples algoritmo *IBL*. A função de similaridade, 3.1, utilizada é:

$$Similaridade(x,y) = -\sqrt{\sum_{i=1}^{n} f(x_i,y_i)} \tag{3.1}$$

Onde n é a quantidade de atributos. É definido $f(x_i,y_i) = (x_i - y_i)^2$ para atributos numéricos e $f(x_i,y_i) = (x_i \neq y_i)$ para atributos com valores boleanos e simbólicos. Os atributos sem valores assumem ter a máxima diferença do valor a ser comparado. Se ambos os atributos não contêm os seus valores, então $f(x_i,y_i) = 1$. O algoritmo *IB1* é idêntico ao algoritmo *Nearest Neighbor*, diferença apenas que *IB1* normaliza os atributos, processa as instância incrementalmente e tem política simples para tolerar falta de valores.

3.5.2 Algoritmo IB2

O algoritmo *IB2* é idêntico ao *IB1*, porém no *IB2* armazena apenas as instâncias classificadas erradas. O *IB2* é parecido com o algoritmo *CNN*, porém no *IB2* inicia com o conjunto vazio, já o *CNN* povoa com uma instância de cada classe. A ideia do *IB2* é que a maioria das instâncias classificadas erradas estão próximas da fronteira. Dessa forma o algoritmo *IB2* reduz os requisitos de armazenamento, porém é mais sensível a ruído em relação ao *IB1*, segundo Aha et al. (1991). A seguir o algoritmo *IB2*.

Como o *IB2* erra na classificação, por armazenar muitas instâncias de ruídos, houve a necessidade de se criar um novo algoritmo chamado *IB3*. O algoritmo *IB3* é descrito a seguir.

Algoritmo 7: Algoritmo *IB1*

Entrada: Conjunto de dados $T = x_1, ..., x_n$
Saída: Conjunto de dados S

```
1  S = ∅
2  para todo x ∈ T faça
3      para todo y ∈ S faça
4          Sim[y] = Similaridade(x, y)
5      y_max = y ∈ S com valor Sim[y] maximal
6      se classe(x) = classe(y_max) então
7          classificação = correto
8      senão
9          classificação = incorreto
10     S = S ∪ x
```

Algoritmo 8: Algoritmo *IB2*, diferença entre o Algoritmo *IB1* em destaque amarelo

Entrada: Conjunto de dados $T = x_1, ..., x_n$
Saída: Conjunto de dados S

```
1  S = ∅
2  para todo x ∈ T faça
3      para todo y ∈ S faça
4          Sim[y] = Similaridade(x, y)
5      y_max = y ∈ S com valor Sim[y] maximal
6      se classe(x) = classe(y_max) então
7          classificação = correto
8      senão
9          S = S ∪ x
10         classificação = incorreto
```

3.5.3 Algoritmo IB3

O algoritmo *IB3* é uma extensão do algoritmo *IB2*, onde a função de similaridade é idêntica do *IB2*. No algoritmo *IB3* é utilizado um filtro para determinar quais instâncias salvas devem ser utilizadas para a tomada de decisão do classificador. A função de classificação e função de atualização do algoritmo são diferentes, essas diferenças são descritas a seguir:

1. Função de Classificação: O algoritmo *IB3* mantém armazenado um registro de classificação de cada instância, o número correto e incorreto de classificação, de cada instância armazenada. A classificação armazenada é resumida em performance da classificação da instância no conjunto de treinamento, apresentado posteriormente, e sugere como vai ser o desempenho futuramente.

2. Função de Atualização: O *IB3* realiza um teste significância para determinar qual instância são bons classificadores e quais acreditam serem ruídos. Os primeiros são utilizados para classificar os casos apresentados posteriormente. Os últimos são descartados da Descrição do Conceito (CD).

Algoritmo 9: Algoritmo *IB3*, diferenças entre o Algoritmo *IB2* em destaque amarelo

Entrada: Conjunto de dados $T = x_1, ..., x_n$
Saída: Conjunto de dados S

```
1  S = ∅
2  para todo x ∈ T faça
3      para todo y ∈ S faça
4          Sim[y] = Similaridade(x, y)
5      se ∃y ∈ S|aceitavel(y) então
6          y_max = algum y ∈ S aceitável com Sim[y] maximal
7      senão
8          i = random(1, len(S))
9          y_max = y ∈ S com valor Sim[y] maximal
10     se classe(x) ≠ classe(y_max) então
11         classificação = correto
12     senão
13         S = S ∪ x
14         classificação = incorreto
15     para todo y ∈ S faça
16         se Sim[y] > Sim[y_max] então
17             Atualiza a classificação de y
18             se y é significamente pior então
19                 S = S − y
```

Segundo Aha et al. (1991), o algoritmo *IB3* assume que o registro de classificação vai distinguir as instâncias com ruídos das instâncias sem ruídos. As instâncias com ruídos vão ter um desempenho na classificação pior, por terem vizinhos próximos com invariavelmente outras classificações. O algoritmo *IB3* funciona bem com dados superficiais simples, consegue prevenir a maioria das instâncias com ruídos na participação de decisão na classificação. Porém, quando se utiliza um conjunto de treinamento que não contêm instâncias com ruídos, a sua performance de classificação cai levemente.

3.6 Decremental Reduction Optimization Procedure

Esta seção o algoritmo *Decremental Reduction Optimization Procedure (DROP)* tem como base o artigo Wilson and Martinez (2000).

Em Wilson and Martinez (2000) é apresentado outro método relacionado com *KNN*, chamado *Decremental Reduction Optimization Procedure* (DROP), contendo cinco métodos: *DROP1*, *DROP2*, ..., *DROP5*. As primeiras versões, *DROP1-DROP3*, foram introduzidas em Wilson and Martinez (1997) pelos nomes *RT1 - RT3* respectivamente. Em *DROP* existe a definição de *associados*. Onde todas as instâncias que tem uma certa instância p fazendo parte dos seus k vizinhos próximos, são chamadas de associados de p.

3.6.1 DROP1

A principal regra do algoritmo é: O *DROP1* descarta instância p do conjunto de treinamento T se os associados de p em S estão corretamente classificados sem p.

Para se verificar se a instância p pode ser removida utilizando essa regra, se verifica cada associado de p o efeito da remoção de p. Essencialmente, essa regra de teste verifica se a remoção de p irá piorar a acurácia generalizada de *leave-one-out* e *cross-validation*, que é uma estimativa da verdadeira capacidade de generalização do classificador resultante. E através dessa regra, *DROP1* descarta as instâncias com ruídos desde que os associados da instância com ruído possam ser corretamente classificada sem a instância com ruído. Quando os vizinhos próximos da instância com ruído são os primeiros a serem removidos, então a instância com ruído não vai ser descartada.

Para resolver esse problema, *DROP2* é similar ao *DROP1* porém os associados da instância são procurados em todo conjunto de treinamento, dessa forma, p é somente deletado se os associados em T estão classificando corretamente, sem p. *DROP3* e *DROP4* descarta as instâncias com ruído usando filtro similar do *ENN* e então aplica *DROP2*. *DROP5* é baseado no *DROP2* mas inicia descartando as instâncias de classe oposta mais próximas afim de suavizar os limites.

3.7 Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation

Segundo Eshelman (2014), *Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation* (*CHC*) é um algoritmo genético não tradicional, que difere dos algoritmos genéticos em seguintes pontos:

Algoritmo 10: Algoritmo *DROP1*

Entrada: Conjunto de dados $T = x_1, ..., x_n$
Saída: Conjunto de dados S

```
1  S = T
2  para todo p ∈ S faça
3      Procurar k+1 vizinhos mais próximos de p em S
4      Adicionar p na lista de associados de cada vizinho
5  para todo p ∈ S faça
6      Seja variável com, o número de associados de p classificados
       corretamente com p como vizinho
7      Seja variável sem, o número de associados de p classificados
       corretamente sem p
8      se sem ≥ com então
9          S = S − p
10         para todo associado A de p faça
11             Remove p da lista de vizinho mais próximo A
12             Procure um novo vizinho mais próximo para A
13             Adicione em A o novo vizinho na lista de associados
14         para todo vizinho de N ∈ p faça
15             Remove p da lista de associados de N
```

1. Seleção Elitista: Conduzido pela seleção de sobrevivência, em vez da seleção reprodutiva, ou seja, o bias a favor das estruturas com melhor desempenho .

2. Prevenção de incesto (*incest prevention*): O novo bias é introduzido contra os indivíduos que são semelhantes.

3. A operação de recombinação é uma variante da recombinação uniforme (*uniform crossover* (*HUX*)), uma forma altamente disruptiva de cruzamento (*crossover*).

4. Mutação não é realizada na fase de recombinação, a diversidade é mantida por randomização parcial da população sempre que a convergência é detectada.

O algoritmo *CHC* substitui reprodução com ênfase por sobrevivência do mais apto. Durante a seleção para reprodução, ao invés de influenciar a seleção de candidatos $C(t)$ para reprodução em favor dos membros com melhor desempenho da população parente $P(t-1)$, cada membro de $P(t-1)$ é copiado para $C(t)$, e randômicamente emparelhados para reprodução. Ou seja, $C(t)$ é idêntico ao $P(t-1)$, exceto a ordem das estruturas que foram embaralhadas. Durante a seleção de sobrevivência, ao invés de trocar a população antiga do parente $P(t-1)$ por uma população filha $C'(t)$ para formar $P(t)$, é criado um

novo filho que deve competir com os membros da população parente $P(t-1)$ por sobrevivência, a competição é *cross-generational*. Mais especificamente, os membros de $P(t-1)$ e $C'(t)$ são fundidos e ordenados de acordo com os melhores, e $P(t)$ é criado selecionando os melhores M membros da população fundida. Esse processo de manter os melhores membros do grupo fundido, da população parente e população filha, é chamada de seleção da população elitista (*population-elitist selection*). O algoritmo *CHC* é descrito em Algoritmo 3.7.

Algoritmo 11: Algoritmo do *CHC*

Entrada: Conjunto de dados T, avaliações e, população M, taxa de divergência r
Saída: Conjunto de dados S

```
1  L = tamanho(T)
2  t = 0
3  d = L/4
4  Inicializa P(0)
5  Avalia P(0)
6  enquanto t < e faça
7      t = t + 1
8      Selecione_r C(t) a partir de P(t-1)
9      Recombine as estruturas de C(t) formando C'(t)
10     Avalia C'(t)
11     Selecione_s P(t) a partir de C'(t) e P(t-1)
12     se P(t) = P(t-1) então
13         d = d - 1
14     se d < 0 então
15         Diverge P(t)
16         d = r * (1.0 - r) * L
17 S = P(t)
```

Para garantir a prevenção de incesto o algoritmo *CHC* executa durante o processo de reprodução, onde cada membro da população parente é randômicamente escolhido sem substituição e emparelhado para cruzamento. Antes do cruzamento, a distância *Hamming* entre potenciais parentes é calculado, e se metade dessa distância não exceder um limiar de diferença (*difference threshold*), eles não são cruzados e são deletados da população filha. Então apenas uma fração da população é cruzada para produzir novos descendentes em qualquer geração. Se nenhum filho for aceito dentro da população parente, o limiar de diferença é decrementado. O efeito desse processo é que somente potenciais parentes são cruzados, mas a diversidade requerida pelo limiar de diferença diminui automaticamente a medida que a população converge naturalmente (Eshelman, 2014).

O ciclo do algoritmo *CHC*, reprodução e recombinação, não utiliza muta-

ção. A utilização de *HUX* e prevenção de incesto em conjunto com o tamanho da população larga o suficiente para preservar um número de diversas estruturas, permite o *CHC* a atrasar convergência prematura. Se saindo bem sem nenhuma mutação. Mas esses mecanismos não garantem que nenhum alelo (um ou mais genes que determinam caracteres alternantes e que se localizam no mesmo lugar em cromossomos homólogos) vão prematuramente convergir. Surge a necessidade de alguma mutação (Eshelman, 2014).

A mutação é menos eficaz no algoritmo *CHC* que em comparação ao algoritmo genético tradicional. O ciclo de reprodução e recombinação do *CHC* já matêm a diversidade, e a mutação contribui muito pouco no início da busca. Segundo Eshelman (2014), uma busca tardia, quando uma população está próxima de convergir, a mutação combinada com seleção elitista não é muito eficaz para reintroduzir diversidade. E nesse estágio da busca, a mutação irá raramente produzir um indivíduo que é melhor que o pior indivíduo na população, dessa forma, poucos novos indivíduos vão ser aceitos na população.

Uma forma que o algoritmo *CHC* resolve a introdução de mutação, é quando a população tem convergido ou a busca ficou estagnada, ou seja, o limiar de diferença cai para zero e quando diversas gerações sem qualquer nova descendência é aceita na população parente. Então quando o ciclo de reprodução e recombinação atingue essa condição, a população é reinicializada e o ciclo é repetido. Sendo essa reinicialização parcial, utilizando os melhores indivíduos como modelo para criação de uma nova população (Eshelman, 2014).

3.8 Random Mutation Hill Climbing

O algoritmo *Random Mutation Hill Climbing* (*RMHC*) é um método de busca local (Skalak, 1994). O algoritmo é descrito a seguir:

Cada *bit* da *string* binária s representa o índice de instâncias armazenadas em T, onde cada *bit* com 1 significa instância contida e o *bit* com 0 a instância não contida. O método $random(t, p)$ gera uma *string* s aleatória, com porcentagem $p - 100\%$ total de *bits* igual a 1. Cada mutação do *RMHC* faz a permutação aleatória de dois *bits*, sendo que uma das duas precisam ser 1 e a outra 0. A avaliação utilizada em $avalie()$, é a acurácia do *KNN* com k igual a 1. Por fim o método $gerarDataSet()$ retorna o conjunto de dados de acordo com as posições de *bits* contido em s. Em e interações vai ser retornado a *string* com a melhor acurácia (Skalak, 1994).

Algoritmo 12: Algoritmo do *RMHC*

Entrada: Conjunto de dados T, avaliações e, porcentagem reduzida p
Saída: Conjunto de dados S

```
1  t = |T|
2  s = random(t, p)
3  melhorAvaliado = avalie(s)
4  interação = 0
5  enquanto interação < e faça
6      stringMutado = mutação(s)
7      avaliadoMutado = avalie(stringMutado)
8      se avaliadoMutado > melhorAvaliado então
9          melhorAvaliado = avaliadoMutado
10         s = stringMutado
11     interação = interação + 1
12 S = gerarDataSet(s)
```

CAPÍTULO

4

Seleção de Instância sobre Aprendizado de Métrica

Esse capítulo é realizada a introdução da proposta desse trabalho chamada de Seleção de Instância sobre Aprendizado de Métrica (*ISML*). Considerando a importância da medida de proximidade para o desempenho de seleção de instâncias, a proposta é uma abordagem unificada para seleção de instâncias baseada em aprendizado de métricas.

Na Seção 4.2 é feita a descrição do coeficiente *Silhouette* para realização da seleção da proposta *ISML*, Seção 4.3 é feita a introdução dos conceitos da área de seleção de instância, Seção 4.4 sobre aprendizado de métrica de distância, Seção 4.5 descrição do algoritmo *Neighbourhood Components Analysis* (*NCA*), Seção 4.6 descrição do algoritmo *k-Neighbourhood Components Analysis* (*kNCA*), Seção 4.7 a descrição da proposta *ISML* e por fim Seção 4.8 a descrição do algoritmo da proposta *ISML*.

4.1 Considerações Iniciais

Os principais algoritmos de seleção de instâncias existentes na literatura utilizam propriedades geométricas do conjunto de dados para definir quando uma instância é considerada relevante, por exemplo, considerando-se a proximidade da instância à borda das classes ou quando a instância é muito distante da maioria das outras instâncias (e.g. instâncias ruidosas). Essa característica pode ser observada pelo fato onde a maioria dos algoritmos existentes são baseados na técnica de vizinho mais próximo. Desse modo um importante desafio de pesquisa é investigar estratégias para definir de forma

adequada uma medida de proximidade entre as instâncias do conjunto de dados, (Olvera-López et al., 2010).

Outro aspecto importante relacionado às tarefas de seleção de instância é o custo computacional envolvido no processo. Em muitos cenários, a seleção de instâncias é empregada principalmente para diminuir o número de instâncias e, assim, reduzir o tempo de aprendizado, sem necessariamente levar a uma melhoria da acurácia do modelo.

Levando em consideração métodos de seleção de instâncias que são dependentes de uma medida de distância ou que selecionam baseando na "geometria" dos dados, a escolha de uma medida de distância para esses algoritmos acaba se tornando uma tarefa não trivial. Sendo assim, a utilização do aprendizado de métricas pode melhorar significativamente a tarefa de seleção de instâncias, principalmente para o classificador *k*-NN.

4.2 Largura de Silhouette

Proposto por Rousseeuw (1987), largura de *silhouette* (*silhouette width*) é uma apresentação gráfica para auxiliar na interpretação e validação em análises de agrupamento.

O agrupamento (*Clustering*) ocorre quando não há informações sobre a classe do conjunto de dados, mas os exemplos devem ser divididos em grupos. Ou seja, é a organização não supervisionada de exemplos baseados em seu grau de similaridade.

O valor de *silhouette* utiliza de métricas de distâncias, dissimilaridade ou similaridade, de cada instância x_i. Segundo Rousseeuw (1987), para construção é necessário a partição obtida por alguma técnica de agrupamento e a coleção de todas as aproximidades entre as instâncias.

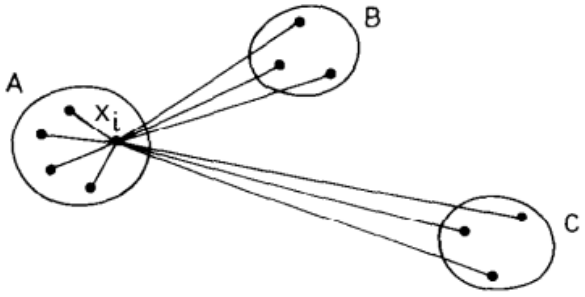


Figura 4.1: Ilustração dos elementos envolvendo $s(x_i)$, onde x_i pertence ao agrupamento A (Rousseeuw, 1987)

Ao utilizar métricas de dissimilaridade, caso da distância Euclidiana, para cada instância x_i é atribuido um valor $s(x_i)$, Figura 4.1, onde o cálculo do valor $s(x_i)$ envolve a média da distância de x_i entre todos outras instâncias do

mesmo agrupamento A de x_i, chamado de $a(x_i)$, e também o cálculo da média da distância de x_i entre todas as outras instâncias do agrupamento $B \neq A$ mais próximo de x_i , chamado de $b(x_i)$.

- $a(x_i) =$ média de distância de x_i com todos os outras instâncias do mesmo agrupamento definido por A.
- $b(x_i) =$ média de distância de x_i com todos os outras instâncias do agrupamento B.

Assim é obtido o valor $s(x_i)$, combinando $a(x_i)$ e $b(x_i)$, definido pela Equação 4.1 a seguir:

$$s(x_i) = \frac{b(x_i) - a(x_i)}{max\{a(x_i), b(x_i)\}} \tag{4.1}$$

Segundo Rousseeuw (1987), os valores de *silhouette* variam de $-1 \leq s(x_i) \leq 1$ para cada instância x_i. Quando os valores de $s(x_i)$ são próximos de 1, implica que $b(x_i)$ é muito maior que $a(x_i)$, ou seja, x_i esta mais próximo do agrupamento A. Dessa forma associado ao conjunto correto.

Na situação oposta, quando valor de $s(x_i)$ é próximo de -1, implica que $b(x_i)$ é muito menor que $a(x_i)$, ou seja, x_i está muito próximo do grupo B do que do grupo A associado. Nessa situação é entendido como classificado errado.

E por fim no caso sendo considerado "intermediário". Os valores de $s(x_i)$ próximos de zero, ocorre quando $a(x_i)$ e $b(x_i)$ são aproximadamente iguais, não é claro para qual agrupamento a instância i deve pertencer.

4.3 Seleção de Instâncias

O principal objetivo da seleção de instâncias é obter um subconjunto representativo S a partir de um conjunto de dados T_r, em que o desempenho P de um algoritmo de aprendizado de máquina seja mantida, ou seja, $P(T_r) \cong P(S)$ (Reinartz, 2002; Olvera-López et al., 2010; Leyva et al., 2013). O objetivo é reduzir o conjunto de treinamento permitindo predizer as classes com a mesma, ou maior, acurácia do conjunto original (Brighton and Mellish, 1999).

Segundo Olvera-López et al. (2010), na prática o conjunto de treinamento T_r pode conter exemplos de pouca relevância para o classificador, que podem ser ruídos ou informações redundantes.

Na Figura 4.2 representa um processo de seleção de instâncias.

Inicialmente a seleção de instâncias concentram em duas competências, a de aprimoramento e a de preservação. A competência de aprimoramento consiste em remover determinadas instâncias, que em alguns casos acarreta na melhoria da acurácia do classificador. Uma forma de se fazer é removendo ou

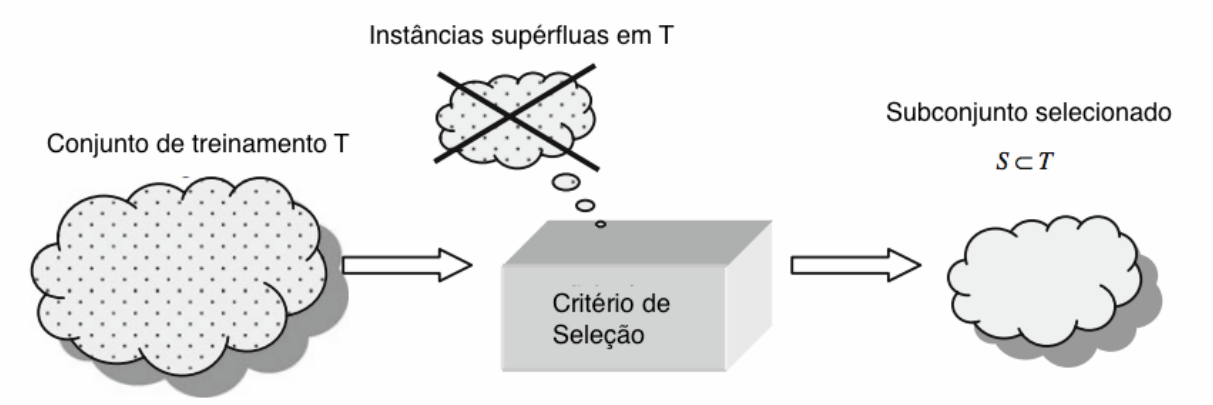


Figura 4.2: Processo de seleção de instâncias (Olvera-López et al., 2010)

isolando as instâncias com ruídos ou instâncias corrompidas. Já a competência de preservação é, quando a instância removida, não reduz a acurácia do classificador.

Utilizando seleção de instâncias, o conjunto de treinamento é reduzido, podendo reduzir o tempo gasto no processo de aprendizado. Como existem diversos algoritmos para se fazer seleção de instâncias, foi feita a divisão em dois grupos:

- *Wrapper*: O critério de seleção é baseado na acurácia do classificador, ou seja, as instâncias que não incrementam a acurácia são descartadas.

- *Filter*: O critério de seleção utiliza uma função de seleção que não é baseada no classificador.

Existem também as diversas maneiras de como se deve construir o subconjunto S: modo incremental, decremental, em lote (*batch*), misturado (*mixed*) e fixo (*fixed*).

- Incremental: No método incremental, é inicializado um subconjunto S vazio, e cada instância de T em S é adicionado de modo que satisfaça alguma métrica. Nesse caso a ordem que se incrementa as instâncias pode ser muito importante. As primeiras instâncias selecionadas vão ter probabilidades diferentes de serem incluídas em S do que seria se forem visitadas mais tardes. Em alguns algoritmos, a seleção das instâncias em T ocorre de forma aleatória (Wilson and Martinez, 2000). Uma vantagem de um método incremental é que as instâncias podem ser incorporadas em S mesmo após o treinamento do algoritmo. Outra vantagem dos algoritmos de método incremental é que podem ser rápidos e ocupar menos espaço de memória durante aprendizado quando comparados com os algoritmos não incrementais, desde que podem ignorar algumas instâncias descartadas ao acrescentar outros. A principal desvantagem do método

incremental é que são sensíveis a ordem de apresentação das instâncias e são propensos a erros até que mais informações sejam disponíveis. Segundo Wilson and Martinez (2000) alguns algoritmos adicionam as instâncias em S de forma incremental, porêm precisam analisar todas as instâncias disponíveis para decidir qual adicionar, tornando-o um método não verdadeiramente incremental.

- Decremental: No método decremental inicializa $S = T$ e então a busca de instâncias para remoção a partir de S. Como no método incremental, a ordem em que as instâncias se apresentam também é importante, porém todos exemplos do conjunto de treinamento são avaliados em qualquer tempo. Dessa forma a busca pode determinar a melhor instância a ser removida durante cada passo do algoritmo. Uma desvantagem do método decremental é pelo fato de ser mais custoso computacionalmente que o método incremental (Wilson and Martinez, 2000).

- Lote: Já no método de lote, todas as instância contidas no conjunto de treinamento que satisfaça o critério de remoção, serão removidas de uma vez só. Esse método ameniza o algoritmo que exige atualização constante da lista de vizinhos próximos quando as instâncias são removidas individualmente (Wilson and Martinez, 2000).

- Misturado (*Mixed*): Começa com um subconjunto S pré selecionado, aleatoriamente ou selecionado pelo método Incremental ou Decremental. Pode adicionar ou remover qualquer instância que atende a algum critério específico (García et al., 2015).

- Fixo (*Fixed*): Pertence a uma subfamília do método Misturado, no qual o número de adições e remoções permanece a mesma. Assim, o número de protótipos no final é determinada no início da fase do aprendizado e nunca é mudado (García et al., 2015).

Outra característica que faz a distinção entre as técnicas de seleção de instâncias é a de manter os exemplos da borda de decisão, exemplos centrais, ou outros conjuntos de exemplos:

- Condensação: Os métodos de Condensação tem o objetivo de manter os exemplos que são próximos da borda de decisão, ou exemplos de borda. A intuição de se obter os exemplos de borda é que os exemplos "internos" (exemplos que não estão próximos de vizinhos de classe oposta e que estejam próximos de exemplos da mesma classe) não interferem na borda de decisão como os exemplos da borda e podem ser removidos com pouco efeito sobre classificação.

- Edição: Já os métodos de Edição (*Edition*) removem os exemplos de borda. O objetivo é remover os exemplos que são ruídos ou que não tem a mesma classe de seus vizinhos. Também deixa a borda de decisão mais "suave".

- Híbrido: Os métodos Híbridos tentam encontrar o menor subconjunto S que mantém ou aumente a acurácia em conjunto de teste. Para isso, é utilizada a remoção de exemplos de borda e internos.

4.4 Aprendizado de Métrica de Distância

Em Wang and Sun (2014), o Aprendizado de Métricas de Distância (*Distance Metric Learning*) é definido por: "O problema de aprendizado de distância de uma função D, por um par pontos de dados x e y, é a de aprender uma função de mapeamento f, cujo $f(x)$ e $f(y)$ estejam no espaço Euclidiano e $d(x,y) = ||f(x) - f(y)||$, onde $||.||$ é l_2 norm."

Para terminologia em aprendizado de métricas, seja o mapeamento $D : X \times X \to \mathbb{R}_0^+$ sobre o espaço vetorial X, é chamado de métrica de distância (*distance metric*) se $\forall x, y, z \in X$, satisfaça as seguintes propriedades (Weinberger and Saul, 2009; Wang and Sun, 2014):

1. Desigualdade triangular: $d(x,y) + d(y+z) \geq d(x,z)$

2. Não Negatividade: $d(x,y) \geq 0$

3. Simetria: $d(x,y) = d(y,x)$

4. Distinguibilidade: $d(x,y) = 0 \iff x = y$

Se o mapeamento satisfaz as três primeiras propriedades, e não a quarta (Distinguibilidade), então é chamada de pseudo métrica (*Pseudo Metric*). A seguir alguns exemplos de métricas de distâncias:

1. Distância *Euclidiana*: $d(x,y) = \sqrt{(x-y)^\top (x-y)}$

2. Distância *Cosseno*: $d(x,y) = \sqrt{1 - \frac{x^\top y}{\|x\|\|y\|}}$

3. Distância χ^2: $d(x,y) = \sqrt{\frac{1}{2}\sum \frac{(x_i - y_i)^2}{x_i + y_i}}$

4. Distância *Mahalanobis*: $d(x,y) = \sqrt{(x-y)^\top S (x-y)}$

5. Distância *Mahalanobis* Generalizada: $d(x,y) = \sqrt{(x-y)^\top M (x-y)}$

As distâncias *Euclidiana*, *Cosseno*, χ^2 e *Mahalanobis* podem ser computadas apenas passando x e y, não necessitam do procedimento de aprendizado, e são chamadas de distâncias fixas.

A matriz S da Distância *Mahalanobis* é o inverso da matriz de covariância dos dados. Já a matriz M da Distância *Mahalanobis* Generalizada é uma matriz simétrica positiva semi-definida. A família de métricas é obtida sobre x, computando a distância *Euclidiana* após realizar uma transformação linear $x = Lx$. Essa métrica calcula o quadrado da distância (Weinberger and Saul, 2009):

$$d_L(x,y) = \| L(x-y) \|_2^2, \tag{4.2}$$

A transformação linear na Equação 4.2 é parametrizada pela matriz L. É comum expressar o quadrado da distância sob a métrica na Equação 4.2 em termos da matriz quadrada (Weinberger and Saul, 2009):

$$M = L^T L \tag{4.3}$$

Toda matriz M, formada a partir de uma matriz L de valor real, deve ser uma matriz positiva semidefinida, ou seja, não conter autovalores negativos (Weinberger and Saul, 2009). Em termos da matriz M, o quadrado da distância é denotada por:

$$d_M(x,y) = (x-y)^T M (x-y) \tag{4.4}$$

Onde a Equação 4.4 é definida pela Distância *Mahalanobis* Generalizada. A métrica *Mahalanobis* Generalizada pode ser tanto parametrizada pela matriz L ou pela matriz M. A distância *Euclidiana* padrão pode ser obtida configurando a matriz M sendo igual a matriz identidade (Weinberger and Saul, 2009).

Segundo Yang and Jin (2006), aprender uma boa medida de distância em um espaço de características é crucial em aplicações do mundo real. Para classificadores *KNN*, a medida de distância é a chave para decidir quais instâncias estão mais próximas de outras, dessa forma definindo a sua classificação.

Os algoritmos de aprendizado de métricas podem ser divididos, dependendo da disponibilidade dos dados, em duas categorias: aprendizado de métricas supervisionado e não supervisionado. Diferente das definições de aprendizado supervisionado, que o conjunto de treinamento tem a classe rotulada, o conjunto de treinamento do aprendizado de métricas supervisionada são convertidas em pares de restrições:

1. Restrições de equivalência: São pares de dados que pertencem a mesma classe.

2. Restrições de falta de equivalência: São pares de dados que não pertencem a mesma classe.

Seja $X = \{x_1, x_2, ..., x_n\}$ o conjunto de pontos, onde n é a quantidade de exemplos do conjunto. Cada $x_i \in \mathbb{R}^m$ é um vetor de dados onde m é a quantidade de características. E seja S o conjunto de restrições de equivalências denotado por $S = \{(x_i, x_j) | x_i$ e x_j pertencem a mesma classe $\}$ e D o conjunto de restrições de falta de equivalência denotado por $D = \{(x_i, x_j) | x_i$ e x_j pertencem a classe diferente $\}$ (Yang and Jin, 2006).

Em aprendizado de métricas supervisionada também pode ser dividida em duas categorias: o aprendizado de métrica de distância global onde aprende a métrica de distância, que satisfaça todos os pares de restrições simultaneamente, e o aprendizado de métricas local que aprende a métrica de distância onde satisfaça os pares de restrições localmente (Yang and Jin, 2006). É também possível categorizar os algoritmos de aprendizado de métricas em linear e não linear, com base no fato da projeção ser linear ou não linear.

Os algoritmos para aprendizado de métricas tem o objetivo de obter uma matriz M, de forma que a distância entre um par exemplos x e y seja pequena quando x e y pertencerem ao conjunto de restrições de equivalência S. De forma análoga, essa distância deve ser maior quando x e y forem de classes diferentes ou restrições de falta de equivalência D.

4.5 Neighbourhood Components Analysis

O *Neighbourhood Components Analysis* (*NCA*), proposto em Goldberger et al. (2004), é um método de aprendizado de uma medida de distância *Mahalanobis* para ser utilizado em um algoritmo *KNN*. Então se deseja uma métrica de distância que maximize a performance da classificação do *KNN*. O ideal seria otimizar a performance em dados futuros, mas como não se tem informação da verdadeira distribuição dos dados, é otimizado a performance da validação cruzada (*leave-one-out*) no conjunto de treinamento.

O aprendizado da métrica de distância *Mahalanobis* foi restrito para ser quadrático, que sempre pode ser representado por matrizes positivas semidefinidas simétricas. As métricas foram estimadas através da sua inversa da raiz quadrada, por aprendizado de uma transformação linear do espaço de entrada de tal modo que no espaço transformado, *KNN* tenha um bom desempenho. Seja a denotação da transformação pela matriz A, $Q = A^T A$ cujo $d(x,y) = (x-y)^T Q(x-y) = (Ax-Ay)^T (Ax-Ay)$.

Segundo Goldberger et al. (2004), o erro da classificação na validação cruzada do *KNN* é completamente uma função descontínua da transformação A, uma vez que qualquer mudança em A possa mudar o espaço dos vizinhos, e

dessa forma, afetando o desempenho da classificação da validação cruzada. Então foi adotada uma medida mais comportada do desempenho do vizinho mais próximo, através de uma função de custo diferenciável baseado em vizinho estocástico (*stochastic neighbour*) atribuído em um espaço transformado. Cada instância i seleciona outra instância j como seu vizinho com alguma probabilidade p_{ij} e herda a classe a partir da instância selecionado. A probabilidade p_{ij} é definida usando *softmax* por cima da distância *Euclidiana* no espaço transformado.

$$p_{ij} = \frac{\exp(-\|Ax_i - Ax_j\|^2)}{\sum_{k \neq i} \exp(-\|Ax_i - Ax_k\|^2)}, p_{ij} = 0 \tag{4.5}$$

Dessa forma é possível computar a probabilidade p_i, da instância i estar corretamente classificado.

$$p_i = \sum_{j \in C_i} p_{ij} \tag{4.6}$$

Então o objetivo é maximizar o número esperado de instâncias corretamente classificados:

$$f(A) = \sum_i \sum_{j \in C_i} p_{ij} = \sum_i p_i \tag{4.7}$$

Diferenciando f em relação a matriz de transformação A produz um gradiente que poderá ser utilizado para o aprendizado (seja $x_{ij} = x_i - x_j$).

$$\frac{\partial f}{\partial A} = -2A \sum_i \sum_{j \in C_i} p_{ij}(x_{ij}x_{ij}^T - \sum_k p_{ik}x_{ik}x_{ik}^T) \tag{4.8}$$

O algoritmo do *NCA* é maximizar a Equação 4.6 utilizando um otimizador baseado em gradiente, como *delta-bar-delta*, ou um gradiente conjugado. Desde que a função de custo não seja convexa, ainda é necessário tomar alguns cuidados para evitar máximos locais durante o treinamento. E de acordo com Goldberger et al. (2004), nos testes realizados não foi observado o efeito de super ajuste (*overfitting*) nos resultados obtidos.

4.6 k-Neighborhood Components Analysis

Segundo Tarlow et al. (2013), o *k-Neighborhood Components Analysis* (*kNCA*) é uma generalização do *NCA*. Enquanto o *NCA* tenta otimizar a métrica de distância assumindo k igual a 1, com o *kNCA* a otimização é para um determinado k. Os dois componentes a seguir,

1. A distribuição de probabilidade sobre a seleção de conjunto de k vizinhos.

2. A medida de acurácia é modificada para mostrar que o *KNN* seleciona uma classe pelo voto majoritário dos k vizinhos.

juntos geram o objetivo, a acurácia esperada do *kNCA*, definida pela Equação 4.9.

$$L_{(k)}(A) = \sum_i \sum_{s \in S_i} p_i(s \mid k; A)[Maj(s) = y_i] \tag{4.9}$$

Onde S_i é o conjunto de todos subconjuntos de vizinhos de i, $Maj(s)$ é a função de maioria e $p_i(s \mid k; A)$ é a probabiliadde de i escolher s como conjunto de vizinhos dado a escolha de k vizinhos. A função $p_i(s \mid k; A)$ é definida por:

$$p_i(s \mid k; A) = \begin{cases} \frac{1}{Z(A)} exp\{-\sum_{j \in s} d_{ij}(A)\} & \text{, se } |s| = k \\ 0 & \text{, se não} \end{cases}$$

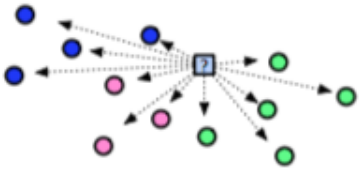
onde $d_{ij}(A) = \| Ax_i - Ax_j \|_2^2$, e a normalização $Z = Z(A)$ implica considerar todos os subconjuntos de tamanho igual a k.

A Equação 4.9 pode ser computada e otimizada eficientemente. Para descrição, a Equação 4.9, focando o objetivo em uma única instância i, pode ser reescrita como:

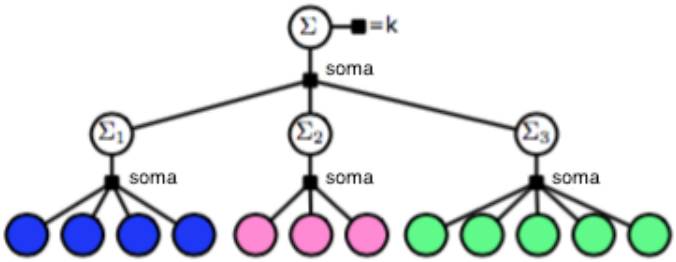
$$\frac{\sum_{s:|s|=k} exp\{-\sum_{j \in s} d_{ij}\}[Maj(s) = y_i]}{\sum_{s \in S:|s|=k} exp\{-\sum_{j \in s} d_{ij}\}}, \tag{4.10}$$

A estratégia definida por Tarlow et al. (2013), é elaborar um conjunto de gráfos de fator (*factor graphs*) para representar os componentes de tal modo que o objetivo pode ser expreso como uma soma das proporções (*ratio*) de funções de partição.

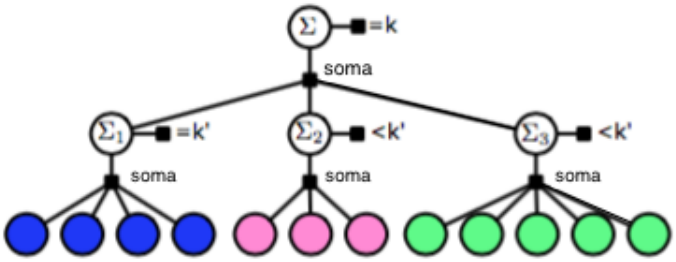
Começando com a construção do grafo de fator, para que a função de partição associada seja igual ao denominador na Equação 4.10, Figura 4.3b. Levando em consideração para uma única instância i com um conjunto de k vizinhos. Na parte inferior do grafo, existe uma variável h_j, indicador binário para cada vizinho j, indica se este é escolhido para ser vizinho de i. Potenciais unários são adicionados (*Unary potentials*). É feito o agrupamento dessas variáveis de acordo com a classe e introdução de uma variável auxiliar para cada classe c (denotado por $\sum_c$) que computa deterministicamente o número de vizinhos da classe c que foram selecionados. No nível superior do grafo, uma variável auxiliar $\sum$, é utilizada para contar o número total de vizinhos que são escolhidos através de todas as classes. Nesse ponto é possível adicionar restrições pelo número total de vizinhos igual a k, e isso é um simples potencial unário em $\sum$ que desabilita todos os valores diferentes de k. Quando essa restrição é adicionada, a função de partição do modelo resultante é igual ao denominador da Equação 4.10.



(a) Uma instância considerando vizinhança com tamanho de k



(b) Construção do Z_0^k



(c) Construção do $Z_1^{k,k'}$ para classe verdadeira sendo a cor azul

Figura 4.3: Construção do grafo de fator utilizada para computar eficientemente a acurácia esperada do *KNN* (Tarlow et al., 2013)

Para construção do numerador, $[Maj(s) = y_i]$ é verdadeiro somente se existe um k' cujo $\sum_{j \in s}[y_j = y_i] = k'$ e $\sum_{j \in s}[y_i = c] < k'$, onde $\forall c \neq y_i$. Ou seja, a soma da classe verdadeira obtem k' e a soma para as outras classes devem ser menores que k'. Essas restrições podem ser expressadas como $\sum_c$. Somando sobre a função de partição desse modelo com $k' = 1,..,k$, é possível realizar todas as possíveis formas para $Maj(s) = y_i$ ser verdadeiro. Logo, a Equação 4.10 pode ser reescrita como a seguir:

$$= \frac{\sum_{k'=1}^{k} \sum_{s:|s|=k} exp\{-\sum_{j \in s} d_{ij}\}[\phi_{k'}(s)]}{\sum_{s \in S:|s|=k} exp\{-\sum_{j \in s} d_{ij}\}} \tag{4.11}$$

$$= \frac{\sum_{k'=1}^{k} Z_1^{k,k'}}{Z_0^k} \tag{4.12}$$

$\phi_{k'}(s) = (\sum_{j \in s}[y_j = y_i] = k') \wedge (\forall c \neq y_i, \sum_{j \in s}[y_i = c] < k')$.

O algoritmo tem a complexidade em $O(Nk + Ck^2)$ para computar as probabilidades marginais, utilizado nos gradientes, e as funções de partição, para ava-

liar o objetivo do *kNCA* esperado. A ideia do Algoritmo 13, é utilizar da programação dinâmica sobre os dois níveis de cadeia de estrutura (*chain-structures*). No primeiro nível de ida (*forward*) é utilizado para calcular as probabilidades sobre o número de vizinhos escolhidos de cada classe separadamente. No segundo nível é combinado os resultados entre as classes para calcular probabilidades sobre o total de vizinhos selecionados. A volta (*backward*) propaga informação de cada classe para trás, para as classes individuais.

Algoritmo 13: Algoritmo de inferência *kNCA* para instância i, Tarlow et al. (2013)

para $c = 1,...,C$ **faça**
  $f_c^1(1) = [1, exp\{-\text{DIST}(i,c,1)\}, 0, ..., 0]$
  para $j = 2,...,J_c$ **faça**
    $f_c^1(j) = \text{Forward1}(f_c(j-1), \text{Dist}(i,c,j))$
$j^2(1) = f_c^1(J_c)$
para $c = 2,...,C$ **faça**
  $f^2(c) = \text{Forward2}(f^2(c-1), f_c^1(c)(J_c), \theta_c(\sum_c)$
$b^2(C) = \theta(\sum)$
para $c = C-1,...,1$ **faça**
  $b^2(c-1) = \text{Backward2}(b^2(c), f_c^1(c)(J_c), \theta_c(\sum_c))$
para $c = C,...,1$ **faça**
  $b_c^1(J_c) = b^2(c)$
  para $j = J_c - 1,...,2$ **faça**
    $b_c^1(j-1) = \text{Backward1}(b_c^1(j), \text{Dist}(i,c,j))$

Utilizando da programação dinâmica, o vetor $f_c(j) \in R^k + 1$ é incrementado computacionalmente onde é armazenado a probabilidade de cada $k' \in 0,..,k$ que as k' variáveis da classe c foram escolhidas como vizinhos de i a partir entre as primeiras j instâncias da classe c. A escolha independente da instância j com probabilidade $p_{ij} = \frac{exp(-d_{ij})}{1+exp(-d_{ij})}$. A função *Forward1* computa $f_c^1(j+1)$ a partir de $f_c^1(j)$ em tempo de complexidade em $O(k)$, utilizando da atualização $f_c^1(j+1)[k'] = f_c^1(j)[k'-1]p_{ij} + f_c^1(j)[k'](1-p_{ij})$.

Segundo Tarlow et al. (2013), existem duas formas para k' variáveis serem escolhidas entre as primeiras j variáveis: ou k' foi escolhido entre os primeiros $j-1$ e o j não foi utilizado, ou $k'-1$ foi escolhido entre os primeiros $j-1$, e j foi utilizado.

Já no segundo nível, o resultado do primeiro nível de *forward* é utilizado para computar os vetores $f^2(c)$, que armazena as probabilidades ascendentes (upward) de cada número possível de vizinho que foi escolhido entre as primeiras c classes. A função *Forward2* calcula essas atualizações em tempo de complexidade em $O(k^2)$ utilizando da atualização $f^2(c)[k'] = \sum_{k_c, k_{c-1}: k_c + k_{c-1} = k'} f^2(c-1)[k_{c-1}] f_c^1(J_c)[k_c] \theta_c(k_c)$. A expressão $\theta_c(k_c)$ representa as restrições dadas como

fatores unários nas variáveis $\sum_c$.

4.7 Proposta

Conforme apresentado nas seções anteriores, a seleção de instâncias representa uma tarefa relevante para a área de aprendizado de máquina. Em especial, permite a melhora do aprendizado de um modelo apresentando apenas as instâncias mais importantes, além de reduzir o tempo computacional para o aprendizado deste modelo.

A proposta chamada Seleção de Instância sobre Aprendizado de Métrica (*ISML*), tem como objetivo realizar seleção de instâcia em cima de um conjunto de dados em um novo espaço de características gerado por aprendizado de métrica. O aprendizado de métricas tem como objetivo de aprender uma medida de distância da família de *Mahalanobis* para melhorar o processo de seleção de instância. O *ISML* aprende uma métrica específica para um classificador k-NN, utilizando o *k-Neighborhood Components Analysis* (*kNCA*). Sendo o *kNCA* uma generalização do *NCA*, como já visto anteriormente, o *NCA* é um método para aprender uma distância *Mahalanobis* que maximize a acurácia do classificador k-NN em um procedimento de validação cruzada. Na Figura 4.4 é mostrado o fluxo do método *ISML*.

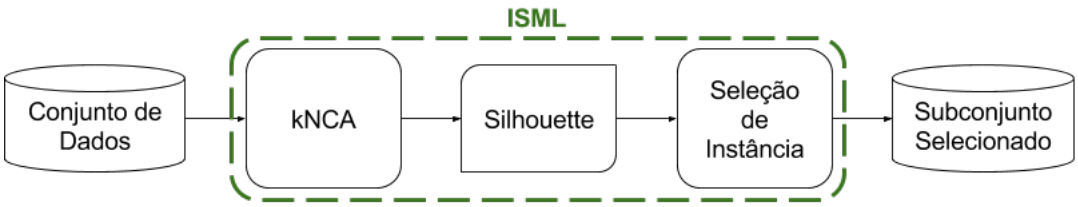


Figura 4.4: Fluxograma do método *ISML*.

Após o aprendizado de uma medida de distância apropriada, é feita a identificação das instâncias que estão próximas da borda de decisão entre as classes. O método *ISML* utiliza um critério de separação inter-classe e intra-classe para identificar esses casos. Foi adaptado um critério chamado *Silhouette*, utilizado em tarefas de agrupamento, com objetivo de separar as instâncias inter-classe e intra-classe. O critério de separação inter-classe e intra-classe é definido na Equação 4.13.

$$s(x_i) = \frac{b(x_i) - a(x_i)}{max\{a(x_i), b(x_i)\}} \tag{4.13}$$

Como já descrito na Seção 4.2, pela definição do *Silhouette*, $a(x_i)$ e $b(x_i)$ retornam a média da distância de x_i com as instâncias da classe mais próxima pertecente a mesma classe de x_i e da mesma forma para classe oposta de x_i respectivamente. A adaptação realizada na Equação 4.13 foi na quantidade

de seleção para realizar a média em a e b, fixando em selecionar o vizinho mais próximo da classe de x_i e o vizinho mais próximo da classe oposta de x_i.

O método *ISML* utiliza dos valores $s(x_i)$ de cada instância x_i do conjunto de treinamento. Se o valor $s(x_i)$ esta próxima do 0, então quer dizer que a instância x_i está próxima da borda de decisão considerando-se uma determinada medida de distância.

4.8 Algoritmo

O método proposto *ISML* é definido no Algoritmo 14 a seguir:

Algoritmo 14: Método ISML

Data: Conjunto de Treinamento T, Tamanho n
Result: Conjunto de Dados S

$S = \emptyset$
$n_0 = n_1 = 0$
$A = kNCA(S)$
$T_A = (A * T^t)^t$
$T_{A^{silhouette}} = silhouette(T_A)$
$T_{A_{ordenado}} = Ordenar(T_{A^{silhouette}})$
para $\forall x_i \in T_{A_{ordenado}}$ **faça**
    se $T_{A^{silhouette}}[x_i] \geq 0$ **então**
        $y_i = classe(x_i)$
        se $n_{y_i} \leq n$ **então**
            $n_{y_i}++$
            $S = S + T_A[i]$

No Algoritmo 14, seja A definido como a matriz de transformação do espaço métrico gerada pelo *kNCA*. Seja E T^t a transposta do conjunto de treinamento T. O conjunto T_A é o resultado do conjunto de treinamento T transformado pela matriz A. Um exemplo é mostrado na Figura 4.5, onde o conjunto de treinamento é o *dataset sonar*. O aprendizado de métricas, *kNCA*, gera uma matriz de transformação A utilizando o conjunto de treinamento T. Dessa forma, gerando o conjunto T_A.

O método $silhouette(T_A)$ realiza o cálculo de $s(x_i)$ para cada instância x_i contida em T_A, onde $i = 1,..,|T_A|$. Na Figura 4.6 são ilustradas as instâncias de T_A coloridas de acordo com valor de *silhouette* de cada instância.

A partir das instâncias em T_A e seus respectivos valores de *silhouette*, é feita a ordenação em ordem crescente dessas instâncias baseado no valor $s(x_i)$ conforme o método $Ordernar()$.

Assim, o conjunto de treinamento em um novo espaço métrico e ordenado conforme sua posição em relação à borda de decisão entre as classes, é realizada a seleção de instâncias. A ordem da seleção das instâncias é importante, pois no método proposto é possível definir um limite máximo de quantidade de instâncias a serem selecionadas.

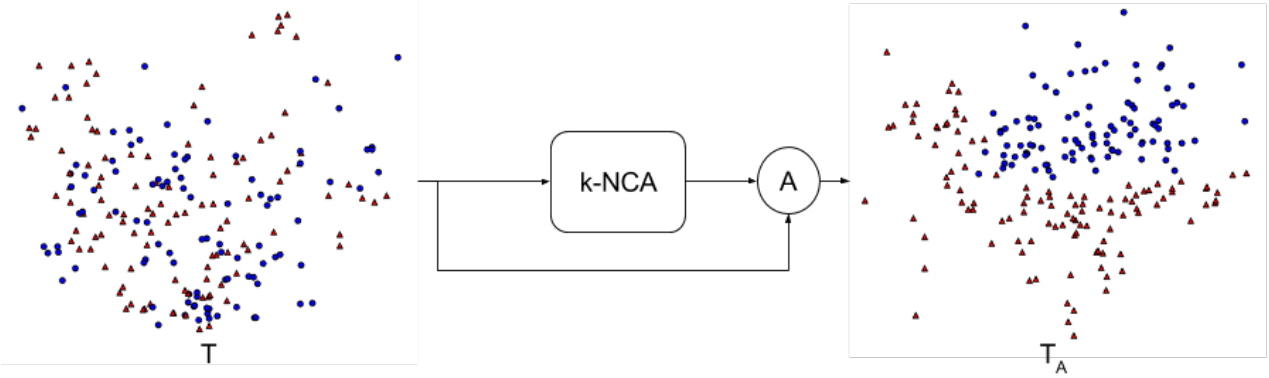


Figura 4.5: Exemplo do fluxograma para gerar o conjunto T_A, utilizando o conjunto de dados *sonar*.

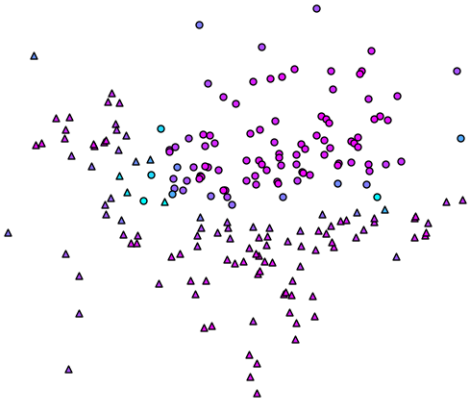


Figura 4.6: Valores de *silhouette* do conjunto T_A.

As instâncias que devem ser selecionadas são as que contêm o valor de *silhouette* maiores ou iguais a 0. Logo as primeiras instâncias que devem ser selecionadas são as que contêm valor de *silhouette* próximas de 0. A quantidade máxima é informada antes da execução, onde n é a quantidade máxima de instâncias que devem ser selecionadas por classe.

CAPÍTULO

5

Experimentos

Para comparação com o método proposto *ISML*, os algoritmos de seleção de instâncias foram implementados em *Python*, versão 2.7.10, e utilizando a biblioteca *Scikit-learn* (Pedregosa et al., 2011), versão 0.17. Os algoritmos de seleção de instâncias implementados foram *ENN*, *CNN*, *RMHC* e *CHC*. O algoritmo *CNN* é um método de Condensação e incremental; *ENN* é um método de Edição e decremental; o algoritmo *RMHC* é um método Híbrido e *Fixed+Wrapper*; e por fim o algoritmo *CHC* é um método Híbrido e *Mixed+Wrapper*

Em relação ao classificador k-NN, foi utilizado da biblioteca *Scikit-learn*. Para comparação e avaliação dos resultados, foi aplicado Validação Cruzada com um k igual a 10. Na Figura 5.1 é apresentado o fluxograma da avaliação experimental.

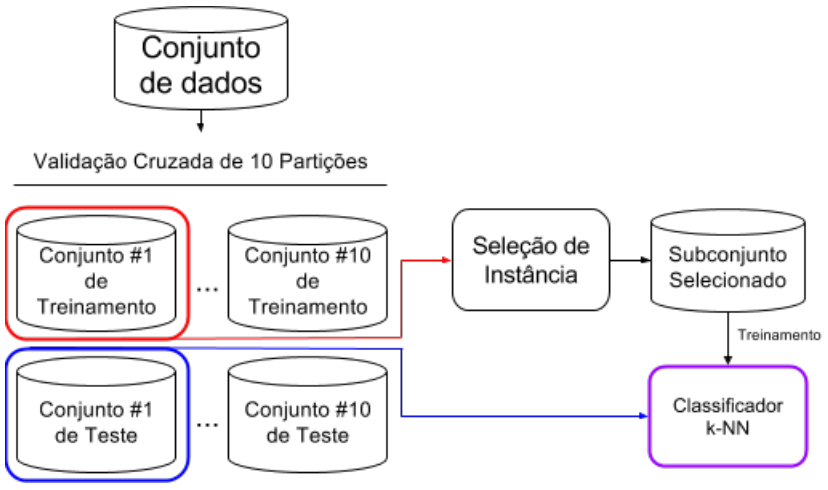


Figura 5.1: Fluxograma dos experimentos realizados nos algoritmos comparados.

Os experimentos relacionados ao método proposto *ISML* são detalhados na Figura 5.2.

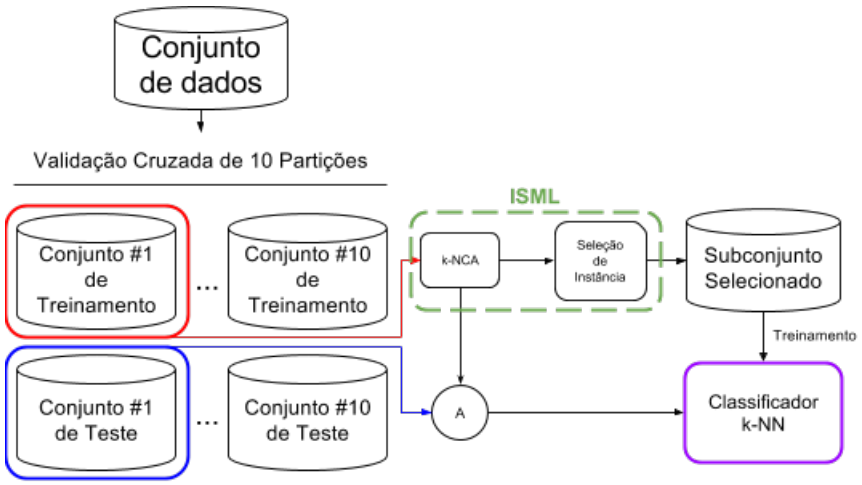


Figura 5.2: Fluxograma dos experimentos realizados na proposta *ISML*.

Em uma primeita etapa, foi realizado um procedimento de ajustes de parâmetros dos algoritmos. Para esta tarefa foi utilizado o método *RandomizedSearchCV* (Bergstra and Bengio, 2012), algoritmo para "afinar"(*fine-tune*) da biblioteca do *Scikit-learn*. Por fim, foi utilizado o software KELL [1] para análise estatística dos resultados experimentais.

Na Tabela 5.1 é apresentada uma descrição geral dos conjuntos de dados utilizados na avaliação experimental, com o rótulo da classe e o número de instâncias de cada classe. Foram utilizadas versão binárias desses conjuntos de dados para todos os métodos avaliados.

Conjunto de Dados	Classe Negativa	Classe Positiva	Total
balance-scale	*R* (288)	*L* (288)	576
diabetes	*tested_positive* (268)	*tested_negative* (500)	768
glass	*build wind float* (70)	*build wind non-float* (76)	146
heart-statlog	*present* (120)	*absent* (150)	270
ionosphere	*g* (225)	*b* (126)	351
iris	*Iris-versicolor* (50)	*Iris-virginica* (50)	100
letter	*D* (805)	*S* (748)	1553
segment	*path* (330)	*cement* (330)	660
sonar	*Rock* (97)	*Mine* (111)	208
spect train	0 (54)	1 (26)	80
vehicle	*saab* (217)	*bus* (218)	435
vowel	*hid* (90)	*hId* (90)	180

Tabela 5.1: Conjunto de Dados utilizados nos experimentos, com as classes utilizadas seguido do tamanho delas.

Os parâmetros definidos para os algoritmos de seleção de instâncias, contidos na Tabela 5.2, são parâmetros encontrados em García et al. (2008). Ape-

[1] KELL: http://www.keel.es/

nas para a variável "geração" (*generation*) do algoritmo *CHC* foi definida aleatoriamente com um valor inferior, em García et al. (2008) é definido por 10000. Devido ao grande custo computacional, neste trabalho o parâmetro "geração" foi reduzido para 100.

Algoritmo	Parâmetros
CHC	população=50 avaliações=100 r=0.5 alfa=0.5
RMHC	interações=10000 porcentagem=90%

Tabela 5.2: Parâmetros utilizados.

Os parâmetros do método ISML foram ajustados por uma técnica chamada *RandomizedSearchCV*. Usamos a medida AUC (Área abaixo a curva ROC) (Flach, 2003) para avaliar o desempenho do classificador. Além disso, também utilizamos a medida ISP (Performance da Seleção de Instância), que é uma média harmônica entre a acurácia do classificador e a taxa de redução do conjunto de treinamento. Nesse caso, quando valor de ISP é próximo a 1, significa que o método de seleção de instância foi capaz de reduzir significativamente o conjunto de dados, sem comprometer a sua acurácia.

Na Seção 5.1 são apresentados os resultados relacionados à análise do coeficiente de *Silhouette* e na Seção 5.2, os resultados de comparação da *AUC*, da porcentagem de redução dos conjuntos de dados e o tempo de execução, entre a proposta *ISML* e os algoritmos de seleção de instância existentes na literatura.

5.1 Análise do coeficiente de Silhouette

Nesta seção é apresentada uma análise do coeficiente de *silhouette*, adaptado neste trabalho, para identificar instâncias na borda de decisão de classificadores.

Foram gerados gráficos mostrando a quantidade de instâncias relacionadas ao seu valor de coeficiente de *Silhouette*. Os experimentos fornecem evidências de que ao aplicar o algoritmo de aprendizado de métricas *kNCA* no conjunto de dados, as instâncias da mesma classe ficam mais próximas e as instâncias com classe de rótulo diferente são afastadas no novo espaço métrico. Dessa forma, os valores de coeficiente de *Silhouette* tendem a 1, conforme ilustrado na Figura 5.3a para o conjunto de dados "diabetes", Figura 5.3b para o conjunto de dados "letter" e a Figura 5.3c para o conjunto de dados "sonar". Os histogramas vermelhos são relacionados aos coeficientes de *Silhouette* antes do aprendizado de métricas *kNCA* e os histogramas verdes ilustram os coeficientes após aprendizado de métricas.

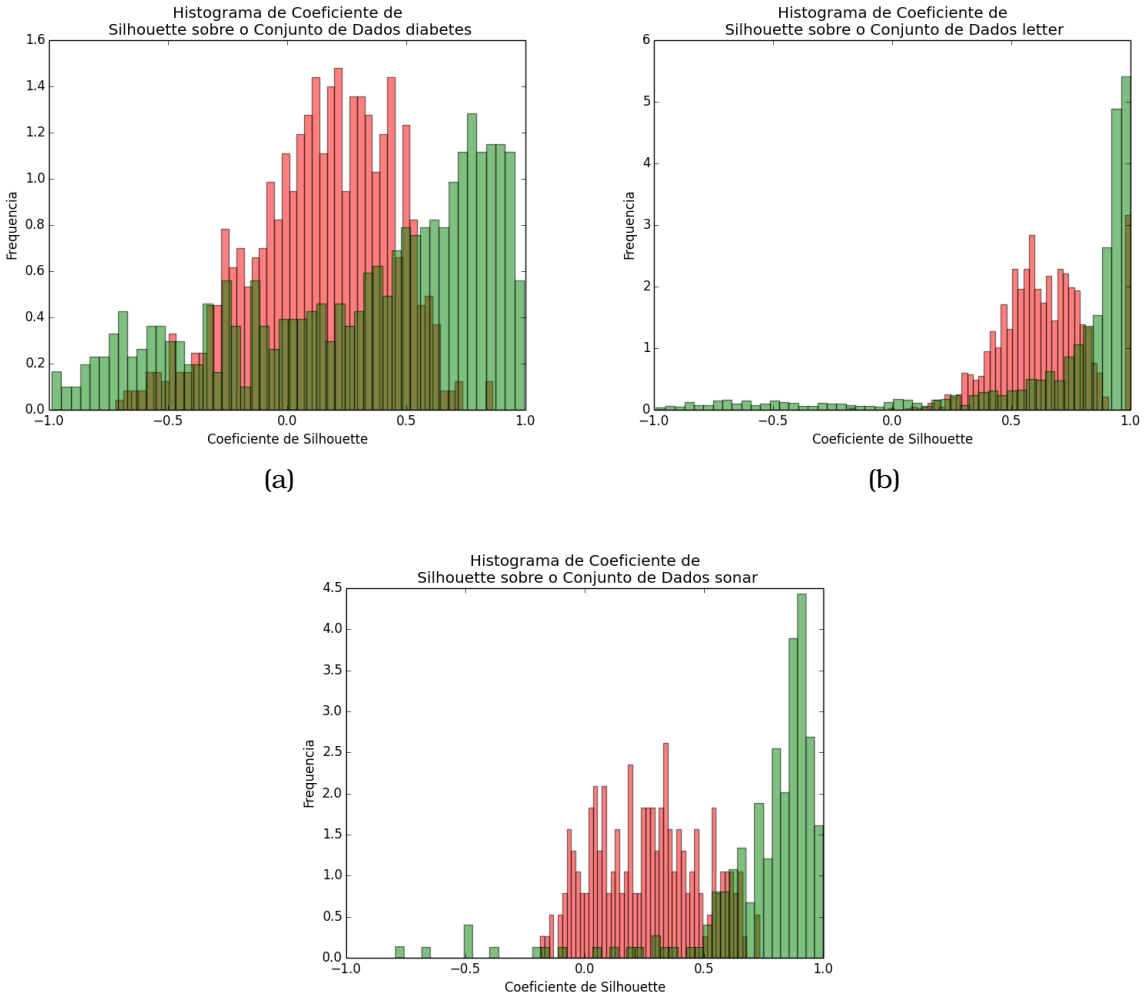
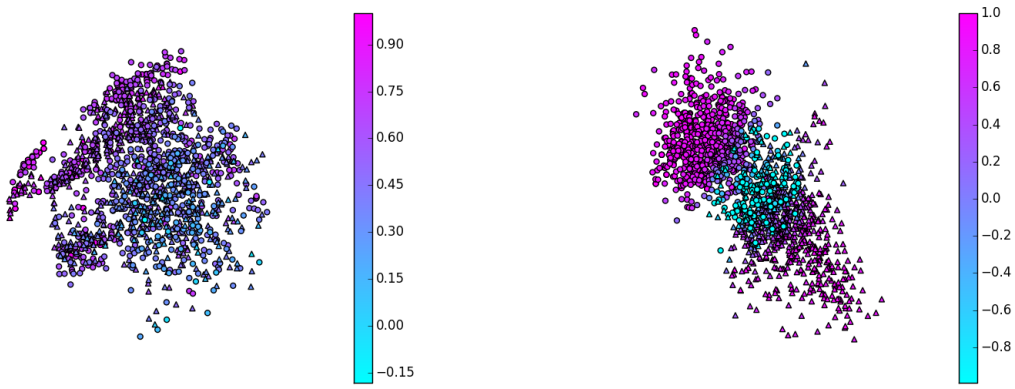


Figura 5.3: Histogramas de coeficiente de *Silhouette* dos conjunto de dados (a) diabetes, (b) letter e (c) sonar, onde as barras em vermelho são antes da aplicação do aprendizado de métricase e as verdes são depois da aplicação do aprendizado de métricas.

Para mostrar o efeito visualmente do algoritmo de métricas *kNCA* com os valores respectivos de coeficiente de *Silhouette*, foram gerados gráficos relacionados ao conjunto de dados antes e depois da aplicação do algoritmo de aprendizado de métricas *kNCA*, conforme ilustrado nas Figuras 5.4a e 5.4b.

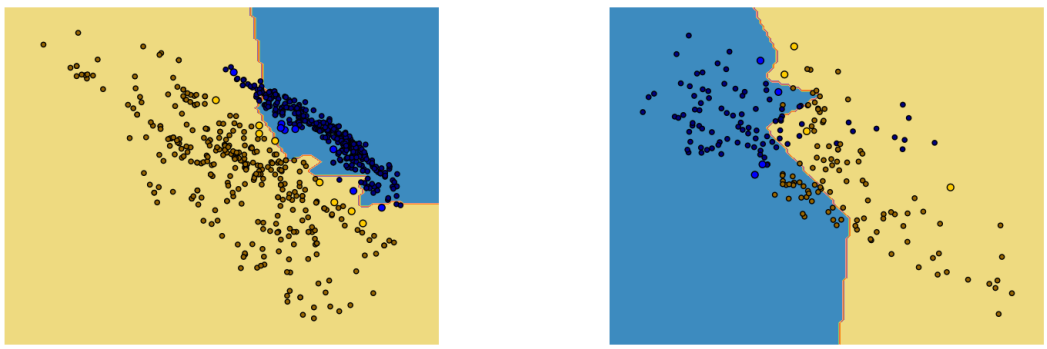


(a) (b)

Figura 5.4: Espaço do conjunto de dados letter (a) sem o aprendizado de métricas *kNCA* e (b) com o aprendizado de métricas *kNCA*, seguido dos valores de coeficiente de *Silhouette*

Nas Figuras 5.4a e 5.4b, é possível notar que no conjunto de dados *letter*, após a aplicação do algoritmo *kNCA*, a borda de decisão fica mais visível e as instâncias mais separadas de acordo com a sua classe, como mostra a Figura 5.3b. A borda de decisão contém valores de coeficiente de *Silhouette* próximos do valor 0.

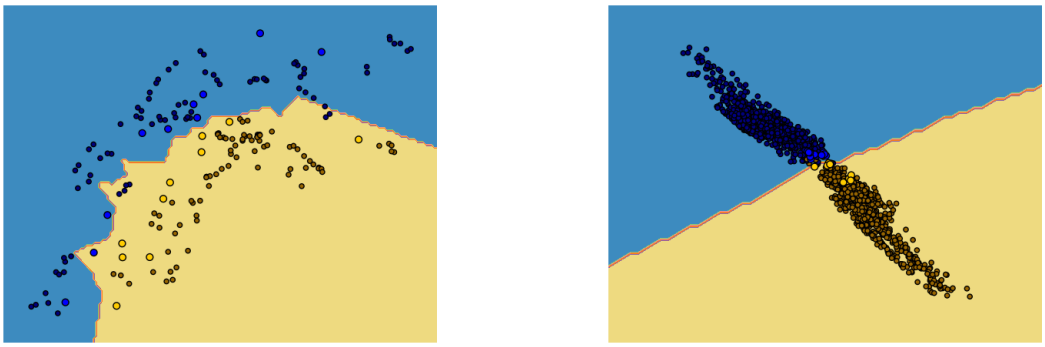
Nas Figuras 5.5a, 5.5b, 5.6a e 5.6b, a seguir, são ilustrados os conjuntos de dados reduzido pelo método *ISML*. O fundo demarca a borda de decisão do algoritmo k-NN, as instâncias selecionadas utilizadas para pertencerem ao conjunto de treinamento estão representadas com a circunferência maior e com cores claras.



(a) (b)

Figura 5.5: Espaço do conjunto de dados *segment* e *sonar* após aplicação do método *ISML* Onde o fundo demarca a classe predita e os círculos claros e maiores indicam a instância selecionada pelo algoritmo.

É possível notar que as instâncias selecionadas pelo método *ISML*, obtém



(a) (b)

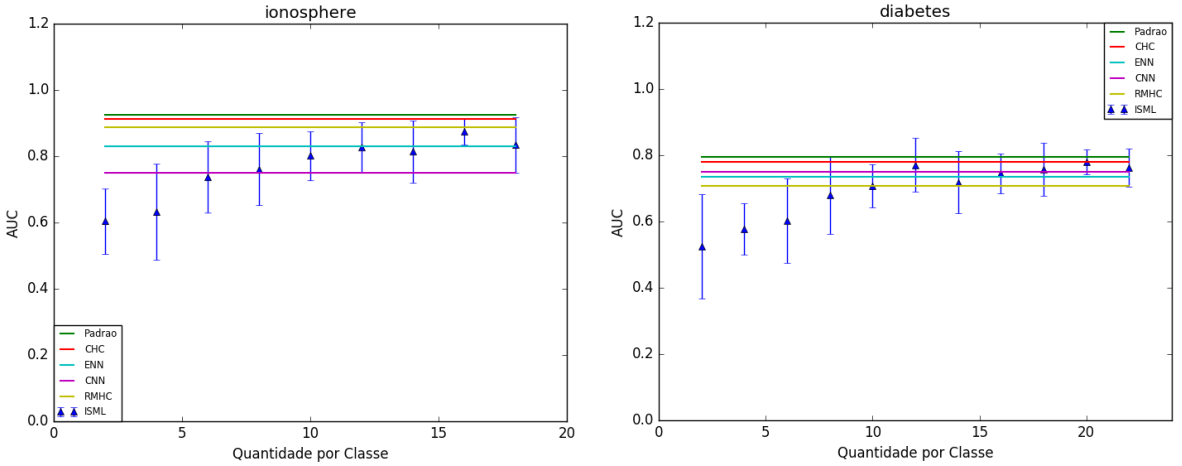
Figura 5.6: Espaço do conjunto de dados *vowel* e *letter* após aplicação do método *ISML*. Onde o fundo demarca a classe predita e os círculos claros e maiores indicam a instância selecionada pelo método.

instâncias que estão próximas ou na borda de decisão do conjunto de dados, indicando que o coeficiente de *Silhouette* no novo espaço métrico consegue identificar adequadamente esse tipo de instância.

5.2 Resultados Experimentais

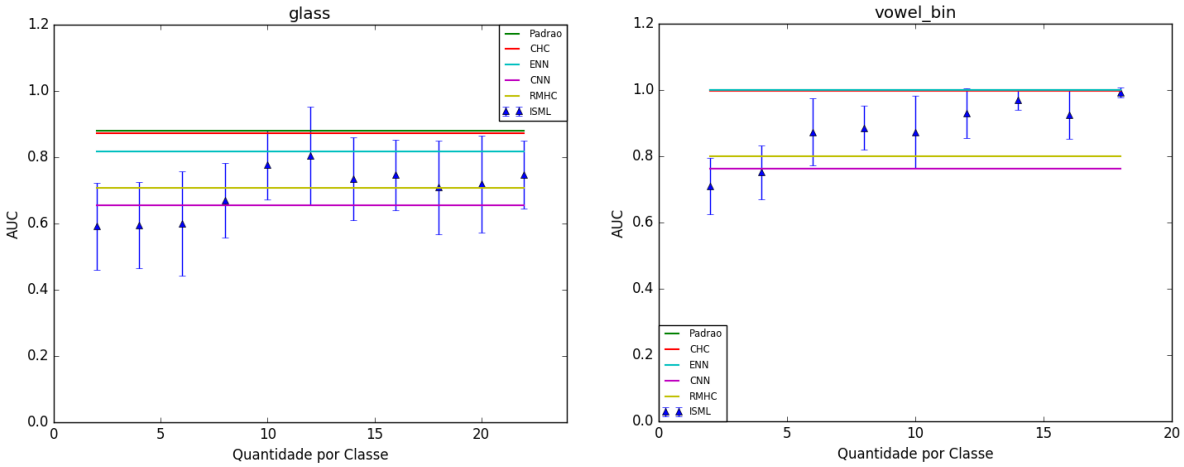
Os resultados experimentais são analisados considerando dois aspectos: desempenho (*AUC*) do classificador e capacidade de redução de instâncias. Para os experimentos do método *ISML*, foram gerados gráficos comparando a *AUC* do k-NN. O conjunto de dados sem seleção de instâncias, é apresentado com nome de "Padrão". Os algoritmos de seleção de instâncias *ENN*, *CNN*, *RMHC* e *CHC*. E por fim a proposta do método *ISML*.

A Figura 5.7a mostra os resultados de desempenho *AUC* no conjunto de dados *ionosphere*. Assim como para a Figura 5.7b para o conjunto de dados *diabetes*, Figura 5.8a para o conjunto de dados *glass*, Figura 5.8b para o conjunto de dados *vowel*, Figura 5.9a para o conjunto de dados *heart-statlog*, Figura 5.9b para o conjunto de dados *iris*, Figura 5.10a para o conjunto de dados *spect-train*, Figura 5.10b para o conjunto de dados *balance-scale*, Figura 5.11a para o conjunto de dados *letter*, Figura 5.11b para o conjunto de dados *vehicle*, Figura 5.12a para o conjunto de dados *segment* e por fim a Figura 5.12b para o conjunto de dados *sonar*.



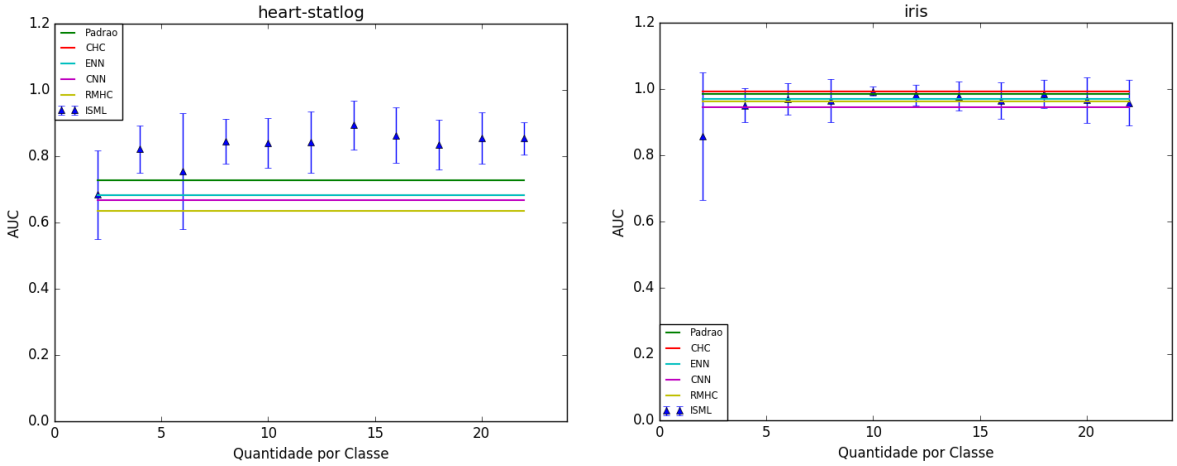
(a) (b)

Figura 5.7: Gráfico com as curvas de *AUC* do (a) conjunto de dados *ionosphere* (b) e conjunto de dados *diabetes*.



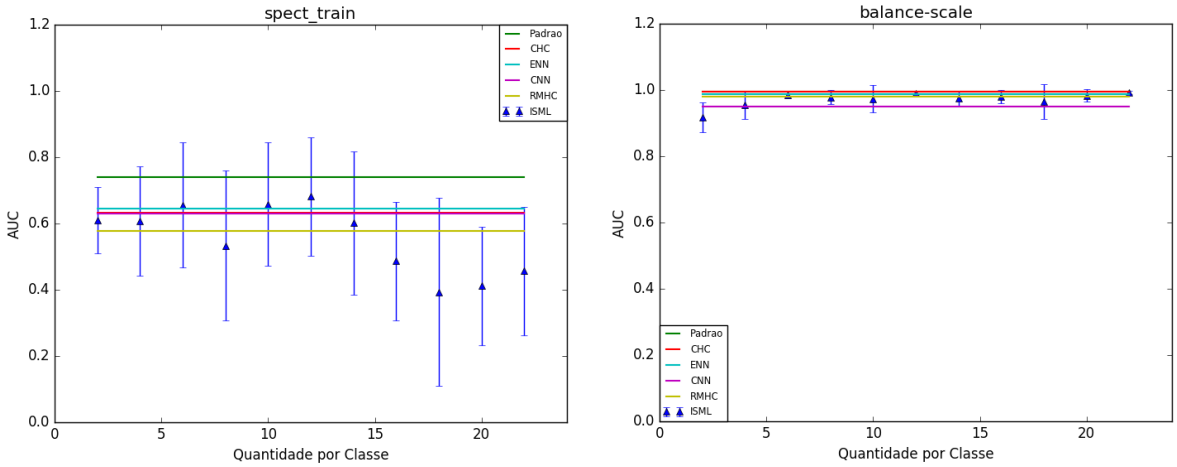
(a) (b)

Figura 5.8: Gráfico com as curvas de *AUC* do (a) conjunto de dados *glass* (b) e conjunto de dados *vowel*.



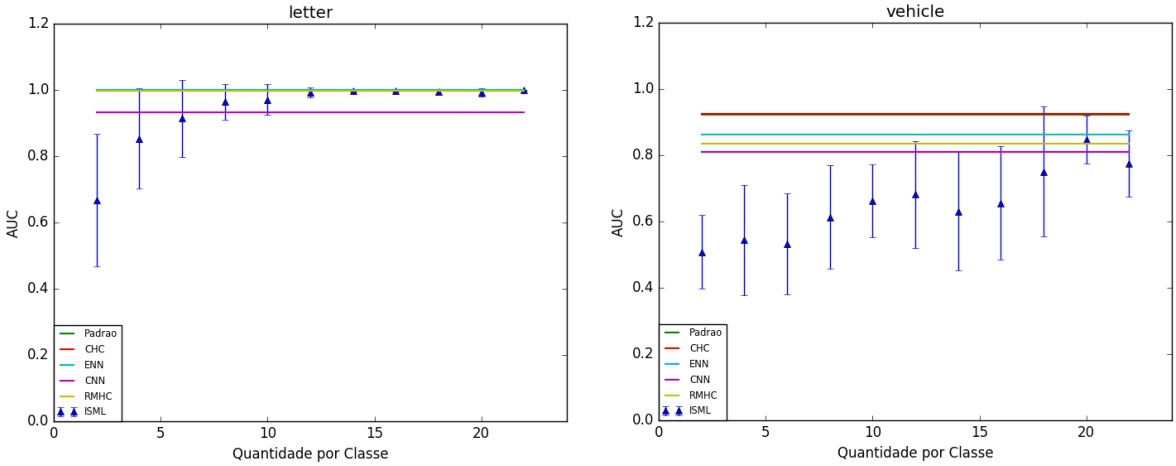
(a) (b)

Figura 5.9: Gráfico com as curvas de *AUC* do (a) conjunto de dados *heart-statlog* (b) e conjunto de dados *iris*.



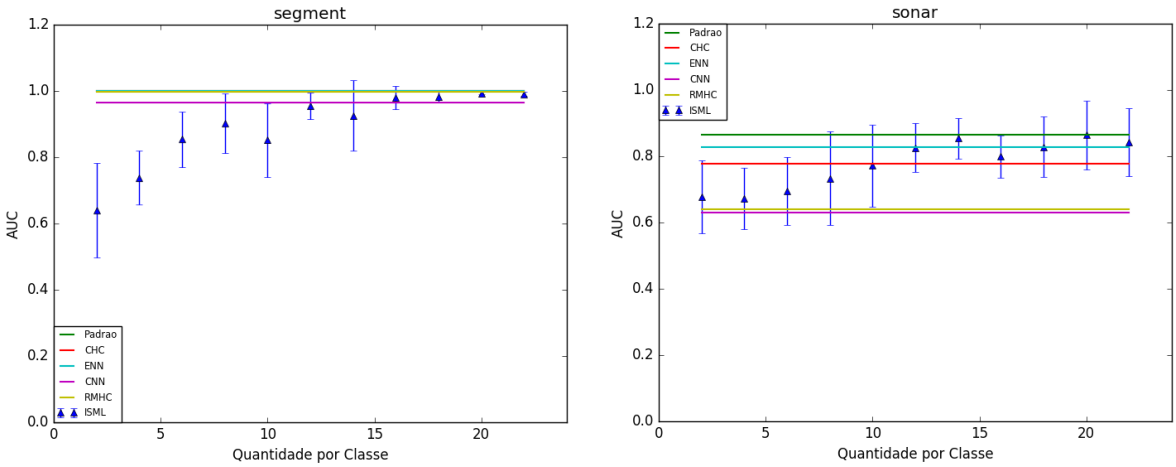
(a) (b)

Figura 5.10: Gráfico com as curvas de *AUC* do (a) conjunto de dados *spect-train* (b) e conjunto de dados *balance-scale*.



(a) (b)

Figura 5.11: Gráfico com as curvas de *AUC* do (a) conjunto de dados *letter* (b) e conjunto de dados *vehicle*.



(a) (b)

Figura 5.12: Gráfico com as curvas de *AUC* do (a) conjunto de dados *segment* (b) e conjunto de dados *sonar*.

Para comparar a taxa de redução do conjunto de dados que foi obtida por cada algoritmo é utilizado gráfico de barras, conforme ilustrado na Figura 5.13. Para o método *ISML*, foi selecionado o conjunto com a maior *AUC* dos testes para aquele conjunto de dados. E para os outros métodos em comparação, *CHC*, *ENN*, *CNN* e *RHMC*, foi selecionado o único conjunto gerado por eles.

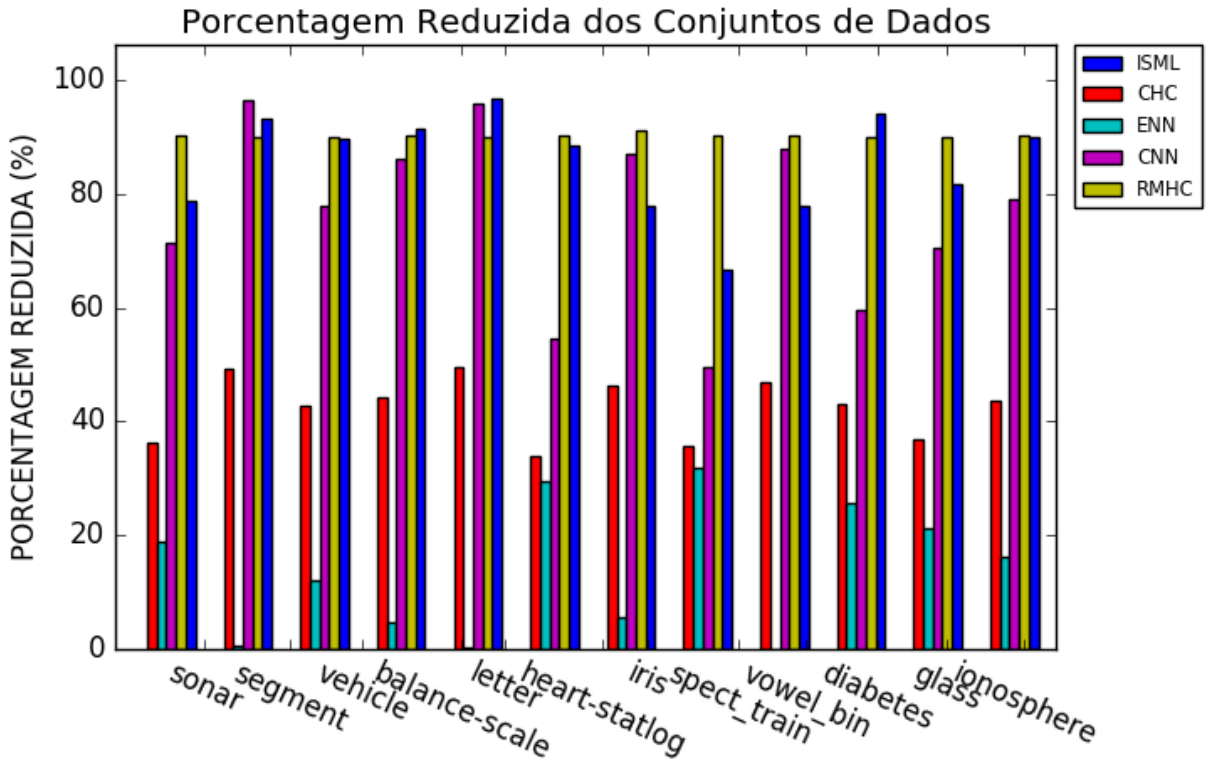


Figura 5.13: Gráfico de barras com o tamanho dos subconjuntos gerados pelos algoritmos em cada conjunto de dados

É possível notar que o algoritmo *RMHC* obtém uma significativa taxa de redução em alguns conjuntos de dados.

Em relação ao tempo de execução dos algoritmos de seleção de instâncias, na Tabela 5.3 é apresentada a comparação com o tempo de execução, no treino e teste, de todos os algoritmos.

O método *ISML* no treino é um dos algoritmos que mais consome tempo, junto com o algoritmo *CNN*. No entanto, este tempo é compensado na etapa de teste, uma vez que está diretamente relacionada ao tamanho do subconjunto obtido por cada algoritmo.

5.3 Análise Estatística

Analisamos os resultados experimentais em dois aspectos: (1) a comparação da melhor configuração para cada método com relação a acurácia do k-NN

Algoritmo	Tempo no treino	Tempo no teste
ISML 2	115.3822691	0.0004661560059
ISML 4	134.5899611	0.0005129814148
ISML 6	150.4186308	0.0005795717239
ISML 8	133.0290919	0.0006220579147
ISML 10	113.4263114	0.0006577730179
ISML 12	150.9329261	0.000723695755
ISML 14	145.510298	0.0007652759552
ISML 16	157.3240796	0.0007750272751
ISML 18	168.1625893	0.0008242845535
ISML 20	150.2946089	0.0008919715881
ISML 22	177.16086	0.0009076833725
CHC	160.5576524	0.004329276085
ENN	4.534798121	0.003200054169
CNN	0.866108799	0.001132392883
RMHC	1.622119474	0.0008306503296

Tabela 5.3: Média do tempo gasto em segundos no treinamento e no teste.

depois da seleção de instância, e (2) a comparação considerando o cenário com seleção de instância mais agressiva, ou seja, com a redução significativa do conjunto de treinamento (reduzido em 50% ou mais). Nesta análise, a taxa (%) de redução é denominada pela métrica *RDC*.

Tabela 5.4: Medida de desempenho do classificador (AUC) e a porcentagem de redução do conjunto de treinamento (RDC) para cada método de seleção de instância.

	CHC		CNN		ENN		RMHC		Silhouette		ISML	
Dataset	AUC	RDC	AUC	RDC	AUC	RDC	AUC	RDC	AUC	RDC	AUC	RDC
balance-scale	0.994 ± 0.006	44.3	0.951 ± 0.019	86.2	0.988 ± 0.012	4.9	0.980 ± 0.013	90.2	0.939 ± 0.024	91.5	0.993 ± 0.004	91.5
diabetes	0.781 ± 0.079	43.0	0.749 ± 0.056	59.7	0.734 ± 0.066	25.8	0.707 ± 0.045	90.0	0.653 ± 0.085	94.2	0.779 ± 0.037	94.2
glass	0.874 ± 0.069	36.9	0.655 ± 0.102	70.6	0.818 ± 0.095	21.2	0.708 ± 0.126	90.1	0.802 ± 0.106	66.5	0.804 ± 0.147	81.7
heart-statlog	0.683 ± 0.083	34.0	0.667 ± 0.096	54.5	0.682 ± 0.075	29.6	0.636 ± 0.084	90.1	0.711 ± 0.087	83.5	0.894 ± 0.074	88.5
ionosphere	0.914 ± 0.060	43.7	0.751 ± 0.117	79.2	0.831 ± 0.072	16.2	0.888 ± 0.069	90.2	0.720 ± 0.083	88.6	0.874 ± 0.040	89.8
iris	0.993 ± 0.021	46.2	0.944 ± 0.070	86.9	0.970 ± 0.047	5.7	0.963 ± 0.065	91.1	0.990 ± 0.030	73.3	0.994 ± 0.013	77.8
letter	1.000 ± 0.000	49.7	0.933 ± 0.060	95.8	0.999 ± 0.002	0.2	0.997 ± 0.003	90.1	0.747 ± 0.085	97.1	0.999 ± 0.002	96.9
segment	0.999 ± 0.002	49.3	0.966 ± 0.015	96.5	1.000 ± 0.000	0.5	0.997 ± 0.003	90.1	0.950 ± 0.036	92.5	0.999 ± 0.001	93.2
sonar	0.778 ± 0.154	36.2	0.629 ± 0.106	71.3	0.827 ± 0.067	19.0	0.639 ± 0.114	90.4	0.687 ± 0.102	87.1	0.864 ± 0.103	78.6
spect-train	0.633 ± 0.144	35.6	0.631 ± 0.245	49.6	0.646 ± 0.207	31.8	0.578 ± 0.149	90.3	0.660 ± 0.181	47.8	0.681 ± 0.178	66.7
vehicle	0.925 ± 0.026	42.9	0.809 ± 0.105	78.0	0.862 ± 0.060	12.1	0.836 ± 0.076	90.0	0.737 ± 0.051	88.7	0.848 ± 0.072	89.8
vowel	0.999 ± 0.004	46.8	0.763 ± 0.129	88.0	1.000 ± 0.000	0.1	0.801 ± 0.103	90.1	0.942 ± 0.058	77.7	0.991 ± 0.014	77.7

Foi adicionado na Tabela 5.4 resultados da proposta sem aprendizado de métricas denominado *Silhouette*. Em relação ao primeiro aspecto, a Tabela 5.4 mostra a medida de desempenho do classificador (*AUC*) e a porcentagem de redução do conjunto de treinamento (*RDC*) para cada método de seleção de instância. Destacamos que o método *ISML* alcança resultados competitivos nas medidas *AUC* e *RDC*.

Uma análise unificada dos métodos de seleção de instância é executada utilizando o critério *ISP (Instance Selection Performance)*, como ilustrado na Figura 5.14. Nessa análise, o método *ISML* proposto apresenta resultados similares ao método *RMHC*. Uma análise estatística indica nenhuma diferença significativa entre *ISML*, *RMHC* e *CNN* para o critério *ISP*. A Figura 5.15 ilustra

o resultado da análise estatística (*Critical Difference*), onde dois métodos são conectados por uma linha quando não existe uma diferença significativa entre eles. Usamos o teste não parâmetrico *Friedman*, com *Nemenyi's post-test* com 95% de nível de confiança.

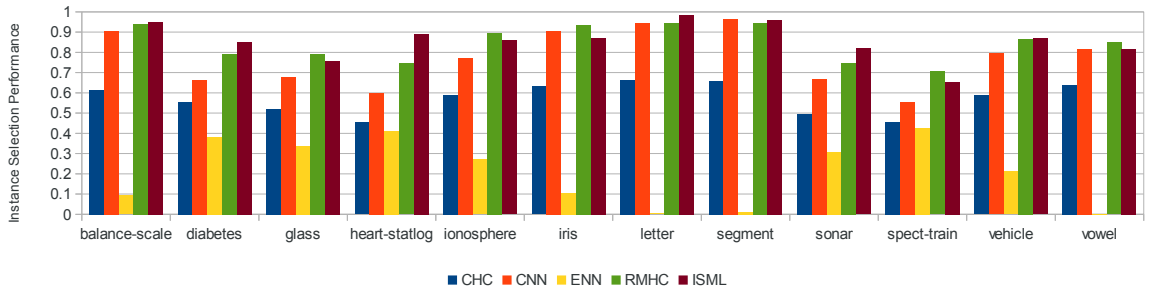


Figura 5.14: Comparação da medida *ISP* (*Instance Selection Performance*) entre os métodos de seleção de instância.

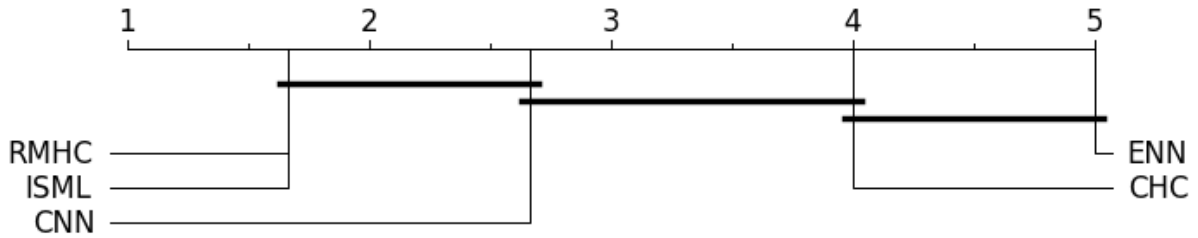


Figura 5.15: Análise estatística dos resultados experimentais, onde dois métodos são conectados por uma linha onde não existe diferença significativa entre eles. Nós usamos o teste *Friedman* não paramêtrico, com *Nemenyi's post-test* com 95% de nível de confiança.

Em relação ao segundo aspecto, a Tabela 5.5 mostra o desempenho da classificação (*AUC*) considerando uma seleção de instância agressiva, isto é, onde o método alcançou uma redução de 50% ou mais do tamanho do conjunto de treinamento. Caso o método de seleção de instância não tenha alcançado 50% de redução no conjunto de treinamento, o seu desempenho *AUC* é apresentado com um " - " na Tabela 5.5. E além disso, foi comparado a *AUC* com um método de referência, que é um classificador k-NN sem seleção de instância.

No cenário de seleção de instância agressivo, o método proposto *ISML* alcança os melhores resultados em relação ao desempenho da classificação (*AUC*), após o processo de seleção de instâncias. Por exemplo, em "heart-statlog" e "iris", o método proposto ainda levou a uma melhoria de classificação, mesmo em comparação com o classificador que utiliza todo o conjunto de treinamento. Uma análise estatística usando o teste não paramétrico de *Wilcoxon* com 95 % de confiança mostra que no cenário seleção de instância agressivo, o método proposto *ISML* é significativamente superior ao método

Tabela 5.5: Desempenho da classificação (*AUC*), considerando um cenário de seleção de instância agressivo. Destacamos que o método de seleção de instância que não alcança 50% de redução do conjunto de dados, é apresentado com a *AUC* de "-".

Dataset	CHC	CNN	ENN	RMHC	Silhouette	ISML	Baseline
balance-scale	-	0.951	-	0.980	0.939	**0.993**	0.995
diabetes	-	0.749	-	0.707	0.653	**0.779**	0.796
glass	-	0.655	-	0.708	0.802	**0.804**	0.880
heart-statlog	-	0.667	-	0.636	0.711	**0.894**	0.728
ionosphere	-	0.751	-	**0.874**	0.720	0.855	0.926
iris	-	0.944	-	0.963	0.990	**0.994**	0.986
letter	-	0.933	-	0.997	0.747	**0.999**	0.999
segment	-	0.966	-	0.997	0.950	**0.999**	1.000
sonar	-	0.629	-	0.639	0.687	**0.864**	0.865
spect-train	-	-	-	0.578	-	**0.681**	0.740
vehicle	-	0.809	-	0.836	0.737	**0.848**	0.923
vowel	-	0.763	-	0.801	0.942	**0.991**	1.000

RMHC. Este aspecto é particularmente importante para classificadores k-NN, pois o custo de classificação computacional é proporcional ao tamanho do conjunto de treinamento.

É possível notar que em conjunto de dados que tem uma *Baseline* baixa, como o conjunto de dados "diabetes", "heart-statlog" e "spect-train". A proposta *ISML* se sobressaiu em relação aos outros algoritmos de seleção de instâncias. Um possível motivo é que esses conjuntos de dados possam estar mais "bagunçados", instâncias próximas de outras instâncias da classe oposta, em relação aos outros conjuntos apresentados. Nessa situação o aprendizado de métricas consegue organizar o espaço para uma seleção melhor das instâncias.

CAPÍTULO

6

Conclusões

O estudo que foi realizado nesse trabalho de mestrado foi de propor um algoritmo de seleção de instância que utiliza de aprendizado de métricas com foco para classificador k Vizinhos Mais Próximos.

A proposta utilizou um método de aprendizado de métricas já proposto na literatura, chamado de *kNCA*, para melhorar a distribuição das instâncias pelo espaço. Para então ser realizada a seleção de instâncias.

O processo de seleção de instâncias, do método *ISML*, utiliza dos valores de *silhouette* de cada instância do conjunto de dados. Esses valores indicam se a instância está próxima de outra instância da mesma classe ou de uma instância da classe oposta. Com base nessa informação, é feita a seleção das instâncias que estão próximas da borda de decisão.

Considerando um cenário de seleção de instância agressivo, onde há uma redução de pelo menos de 50% do conjunto de dados. Nos experimentos apresentados, o método proposto *ISML* conseguiu obter uma *AUC*, na maioria dos conjuntos de dados, superior aos algoritmos comparados de seleção de instância. Em conjunto de dados que tem uma *Baseline* baixa, o método *ISML* se sobressai mais ainda em relação aos outros algoritmos de seleção de instâncias em comparação.

As contribuições alcançadas durante o estudo foram: uma proposta de um algoritmo de seleção de instâncias baseado em aprendizado de métricas e uma extensa avaliação experimental da proposta com outros métodos tradicionais.

6.1 Trabalhos Futuros

Os resultados obtidos por esse trabalho de mestrado, mostram fortes evidências que o aprendizado de métricas pode fazer parte e facilitar o processo de um método de seleção de instância. Para trabalhos futuros, podem ser realizados em:

- Novos experimentos em escolha do melhor limiar de seleção utilizado baseado no valor de *silhouette*;
- Aprofundar o estudo da aplicação de algoritmos de aprendizado de métricas em algoritmos de seleção de instância; e
- Progredir os experimentos para outros classificadores, como *k-Means* e *Cop k-Means*.

Bibliografia

Agrawal, V., Agrawal, S., Nag, S., Chakraborty, D., Panigrahi, B., e Subbarao, P. (2016). Application of k-nn regression for predicting coal mill related variables. In *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, páginas 1–9. IEEE. Citado na página 2.

Aha, D. W., Kibler, D., e Albert, M. K. (1991). Instance-based learning algorithms. *Machine learning*, 6(1):37–66. Citado nas páginas 1, 24, 26, 27, e 29.

Alpaydin, E. (2014). *Introduction to machine learning*. MIT press. Citado na página 1.

Angiulli, F. (2005). Fast condensed nearest neighbor rule. In *Proceedings of the 22nd international conference on Machine learning*, páginas 25–32. ACM. Citado nas páginas xiii, 23, e 25.

Bergstra, J. e Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305. Citado na página 52.

Brighton, H. e Mellish, C. (1999). On the consistency of information filters for lazy learning algorithms. In *Principles of Data Mining and Knowledge Discovery*, páginas 283–288. Springer. Citado na página 37.

Brighton, H. e Mellish, C. (2002). Advances in instance selection for instance-based learning algorithms. *Data mining and knowledge discovery*, 6(2):153–172. Citado nas páginas 2, 3, e 24.

Cerveron, V. e Ferri, F. J. (2001). Another move toward the minimum consistent subset: a tabu search approach to the condensed nearest neighbor rule. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 31(3):408–413. Citado na página 24.

Chen, H., Chiang, R. H., e Storey, V. C. (2012). Business intelligence and analytics: From big data to big impact. *MIS quarterly*, 36(4):1165–1188. Citado na página 2.

Chou, C.-H., Kuo, B.-H., e Chang, F. (2006). The generalized condensed nearest neighbor rule as a data reduction method. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, páginas 556–559. IEEE. Citado na página 24.

Cover, T. e Hart, P. (1967). Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27. Citado nas páginas 2, 11, e 26.

Devijver, P. A. e Kittler, J. (1980). On the edited nearest neighbor rule. In *Proc. 5th Int. Conf. on Pattern Recognition*, páginas 72–80. Citado na página 24.

Eshelman, L. J. (2014). The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. *Foundations of Genetic Algorithms 1991 (FOGA 1)*, 1:265. Citado nas páginas 30, 32, e 33.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874. Citado na página 17.

Flach, P. A. (2003). The geometry of roc space: understanding machine learning metrics through roc isometrics. In *ICML*, páginas 194–201. Citado nas páginas xix, 19, 20, 21, e 53.

Garain, U. (2008). Prototype reduction using an artificial immune model. *Pattern analysis and applications*, 11(3-4):353–363. Citado na página 24.

García, S., Cano, J. R., e Herrera, F. (2008). A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recognition*, 41(8):2693–2709. Citado nas páginas 24, 52, e 53.

García, S., Luengo, J., e Herrera, F. (2015). Instance selection. In *Data Preprocessing in Data Mining*, páginas 195–243. Springer. Citado nas páginas 1, 2, 3, e 39.

Goldberger, J., Hinton, G. E., Roweis, S. T., e Salakhutdinov, R. (2004). Neighbourhood components analysis. In *Advances in neural information processing systems*, páginas 513–520. Citado nas páginas 42 e 43.

Kuncheva, L. I. (1995). Editing for the k-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, 16(8):809–814. Citado na página 24.

Leyva, E., González, A., e Pérez, R. (2013). Knowledge-based instance selection: a compromise between efficiency and versatility. *Knowledge-Based Systems*, 47:65–76. Citado nas páginas 3 e 37.

Lumini, A. e Nanni, L. (2006). A clustering method for automatic biometric template selection. *Pattern Recognition*, 39(3):495–497. Citado na página 24.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In Cam, L. M. L. e Neyman, J., editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, páginas 281–297. University of California Press. Citado na página 15.

McRoberts, R. E., Næsset, E., e Gobakken, T. (2015). Optimizing the k-nearest neighbors technique for estimating forest aboveground biomass using airborne laser scanning data. *Remote Sensing of Environment*, 163:13–22. Citado na página 2.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition. Citado nas páginas 7, 9, 10, e 11.

Mollineda, R. A., Ferri, F. J., e Vidal, E. (2002). An efficient prototype merging strategy for the condensed 1-nn rule through class-conditional hierarchical clustering. *Pattern Recognition*, 35(12):2771–2782. Citado na página 24.

Narayan, B. L., Murthy, C., e Pal, S. K. (2006). Maxdiff kd-trees for data condensation. *Pattern recognition letters*, 27(3):187–200. Citado na página 24.

Olvera-López, J. A., Carrasco-Ochoa, J. A., e Martínez-Trinidad, J. F. (2005). Sequential search for decremental edition. In *Intelligent Data Engineering and Automated Learning-IDEAL 2005*, páginas 280–285. Springer. Citado na página 24.

Olvera-López, J. A., Carrasco-Ochoa, J. A., e Martínez-Trinidad, J. F. (2008). Prototype selection via prototype relevance. In *Progress in Pattern Recognition, Image Analysis and Applications*, páginas 153–160. Springer. Citado na página 24.

Olvera-López, J. A., Carrasco-Ochoa, J. A., Martínez-Trinidad, J. F., e Kittler, J. (2010). A review of instance selection methods. *Artificial Intelligence Review*, 34(2):133–143. Citado nas páginas xiii, xv, 3, 24, 36, 37, e 38.

Olvera-López, J. A., Martínez-Trinidad, J. F., e Carrasco-Ochoa, J. A. (2007). Restricted sequential floating search applied to object selection. In *Machine Learning and Data Mining in Pattern Recognition*, páginas 694–702. Springer. Citado na página 24.

Paredes, R. e Vidal, E. (2000). Weighting prototypes-a new editing approach. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, páginas 25–28. IEEE. Citado na página 24.

PE, H. (1968). The condensed nearest neighbor rule. *IEEE Trans Inf Theory*, 14:515 – 516. Citado nas páginas 23 e 24.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830. Citado na página 51.

Raicharoen, T. e Lursinsap, C. (2005). A divide-and-conquer approach to the pairwise opposite class-nearest neighbor (poc-nn) algorithm. *Pattern recognition letters*, 26(10):1554–1567. Citado na página 24.

Reinartz, T. (2002). A unifying view on instance selection. *Data Mining and Knowledge Discovery*, 6(2):191–210. Citado nas páginas 2 e 37.

Rezende, S. O. (2003). *Sistemas inteligentes: fundamentos e aplicações*. Editora Manole Ltda. Citado na página 7.

Riquelme, J. C., Aguilar-Ruiz, J. S., e Toro, M. (2003). Finding representative patterns with ordered projections. *Pattern Recognition*, 36(4):1009–1018. Citado na página 24.

Ritter, G., Woodruff, H., Lowry, S., e Isenhour, T. (1975). An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 21(6):665–669. Citado na página 24.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65. Citado nas páginas xiii, 36, e 37.

Skalak, D. B. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the eleventh international conference on machine learning*, páginas 293–301. Citado na página 33.

Srisawat, A., Phienthrakul, T., e Kijsirikul, B. (2006). Sv-knnc: an algorithm for improving the efficiency of k-nearest neighbor. In *PRICAI 2006: Trends in Artificial Intelligence*, páginas 975–979. Springer. Citado na página 24.

Tarlow, D., Swersky, K., Charlin, L., Sutskever, I., e Zemel, R. (2013). Stochastic k-neighborhood selection for supervised and unsupervised learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, páginas 199–207. Citado nas páginas xiii, xix, 43, 44, 45, e 46.

Tomek, I. (1976). An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, (6):448–452. Citado na página 24.

Veenman, C. J. e Reinders, M. J. (2005). The nearest subclass classifier: A compromise between the nearest mean and nearest neighbor classifier. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(9):1417–1429. Citado na página 24.

Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S., et al. (2001). Constrained k-means clustering with background knowledge. In *ICML*, volume 1, páginas 577–584. Citado nas páginas 15 e 16.

Wang, F. e Sun, J. (2014). Survey on distance metric learning and dimensionality reduction in data mining. *Data Mining and Knowledge Discovery*, 29(2):534–564. Citado nas páginas 3 e 40.

Weinberger, K. Q. e Saul, L. K. (2009). Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research*, 10:207–244. Citado nas páginas 40 e 41.

Wilson, D. L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *Systems, Man and Cybernetics, IEEE Transactions on*, (3):408–421. Citado nas páginas 24 e 26.

Wilson, D. R. e Martinez, T. R. (1997). Instance pruning techniques. In *ICML*, volume 97, páginas 403–411. Citado na página 30.

Wilson, D. R. e Martinez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine learning*, 38(3):257–286. Citado nas páginas 24, 30, 38, e 39.

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Philip, S. Y., et al. (2008). Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37. Citado na página 2.

Yang, L. e Jin, R. (2006). Distance metric learning: A comprehensive survey. *Michigan State Universiy*, 2. Citado nas páginas 41 e 42.