Capítulo 5 - Máquina de Vetores de Suporte ou *Support Vector Machine*

Introdução

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

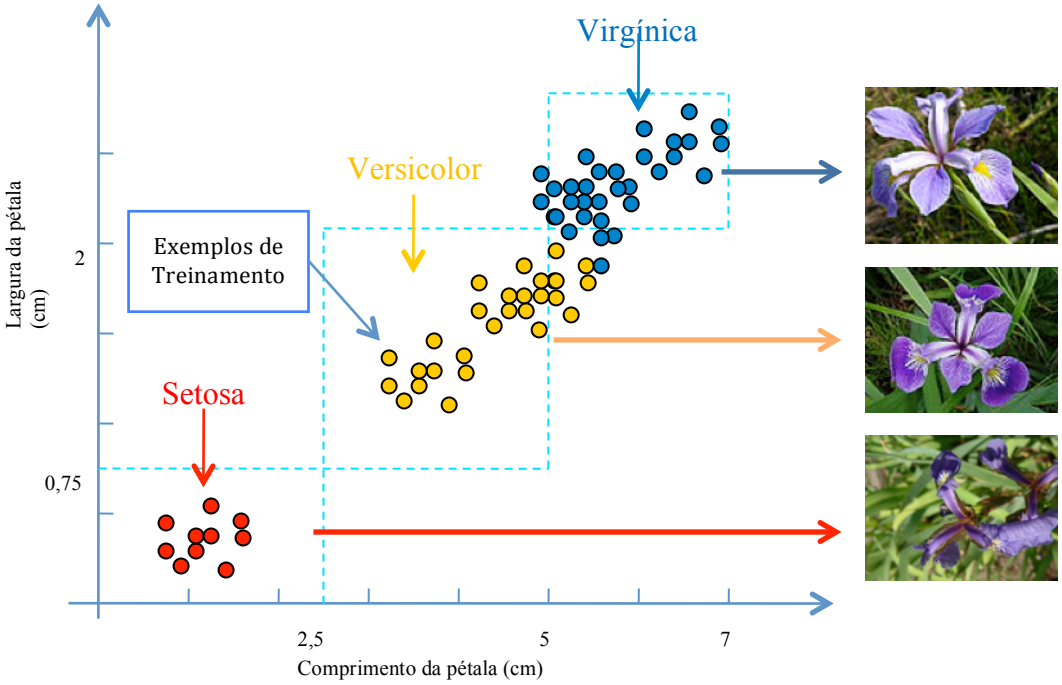


Figura 5.1 – Representação de Exemplos de Treinamento no Plano Cartesiano. [4]

Tanto as **Regras de Associação** e **Classificação** quanto as **Árvores de Decisão** podem ser vistas como representantes de um paradigma de aprendizado supervisionado denominado **Aprendizado Orientado a Conhecimento**, porque são de fácil compreensão e utilização pelos seres humanos. Outro nome que se

[4] Fonte: http://en.wikipedia.org/wiki/File:Iris_virginica.jpg (Acessado em 19.02.13).
Fonte: http://en.wikipedia.org/wiki/File:Kosaciec_szczecinkowaty_Iris_setosa.jpg (Acessado em 19.02.13).
Fonte: http://en.wikipedia.org/wiki/File:Iris_versicolor_3.jpg . (Acessado em 19.02.13).

dá a este tipo de aprendizado é **Paradigma de Aprendizado Simbólico**, porque ele cria **Representações Simbólicas do Modelo** gerado, tais como Árvores de Decisão, Regras etc.

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

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

Dois dos algoritmos de aprendizado supervisionado mais populares dos Modelos Funcionais, são as **Redes Neurais**[5] ou *Neural Networks*, e as **Máquinas de Vetores de Suporte (MVS)** ou *Support Vector Machine (SVM)*. A teoria matemática sobre as Máquinas de Vetores de Suporte foi apresentada à comunidade científica por (CORTES & VAPNIK, 1995), em meados da década de 1990, e desde então suas implementações algorítmicas têm produzido resultados animadores para o problema do **reconhecimento de padrões** em Bases de Dados.

Representação dos Exemplos como Vetores

Nesta introdução a **Máquinas de Vetores de Suporte** (**MVS**), vamos passar a representar um **Exemplo de Treinamento** por um vetor, e o **Modelo Aprendido**, por outro vetor, chamado de **Vetor Peso w**. Qual a vantagem dessa forma de representação? A vantagem é que para classificar um Exemplo, bastará fazer o produto de dois vetores, algo computacionalmente simples. No contexto de classificação com MVS, mostraremos que o resultado dessa multiplicação será sempre ou “+1” ou “-1”. Dessa forma, por meio da operação de **produto interno**

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

entre dois vetores, os **Exemplos de Teste** serão classificados como pertencentes ou à classe “y = +1” ou à “y = -1”.

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

Um classificador binário é equivalente a uma função que faz o mapeamento de um conjunto de **atributos de entrada**, agora representados por um vetor **X**, em uma das classes “y = +1” ou à “y = -1”. Em termos matemáticos,

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

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

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

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

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

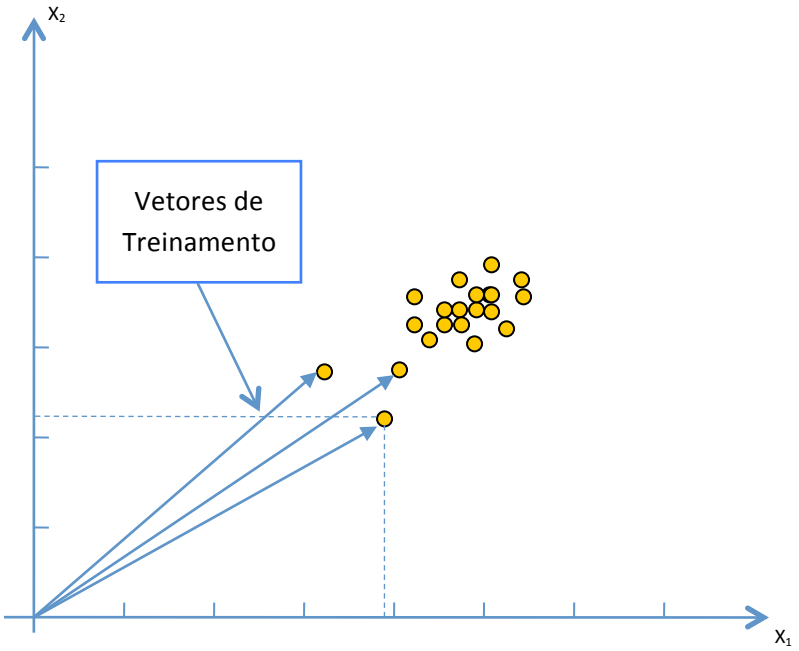


Figura 5.2 – Representação de Vetores de Treinamento no Plano Cartesiano.

Classificadores Lineares

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

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

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

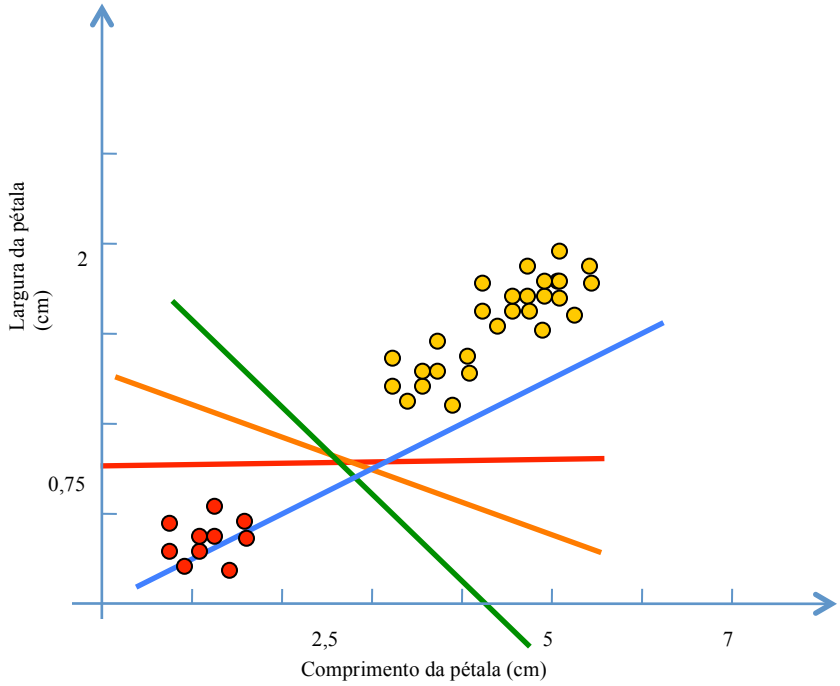


Figura 5.3 – Qual Classificador Escolher?

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

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

Em outros algoritmos de Aprendizado Supervisionado de Classificadores Lineares, como o Perceptron[6] (ROSENBLATT, 1957), os três classificadores da Figura 5.3 que não cometem erros de classificação durante o treinamento seriam considerados igualmente bons. Aliás, como na resposta do Perceptron, e das **Redes Neurais** em geral, há um **componente probabilístico**, em cada simulação com o mesmo conjunto de Vetores de Treinamento, qualquer uma das três retas poderia ser escolhida alternadamente como o classificador linear. Para o Perceptron, desde que um classificador linear faça a separação das classes sem cometer erros durante o treinamento, ele é tão bom quanto seus pares que mostrarem o mesmo resultado.

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

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

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

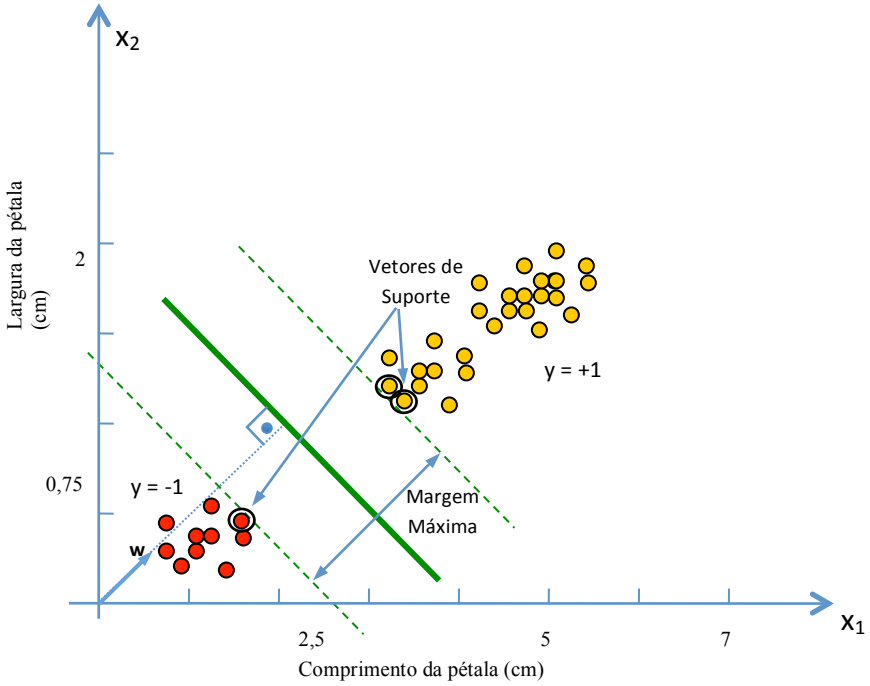


Figura 5.4 - Nas MVS, o Melhor Classificador Possui Margem Máxima.

Conjunto Convexo

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

A Figura 5.5 mostra como as duas classes de Vetores de Treinamento podem ser representadas por dois conjuntos convexos (com as conexões dos pontos internos omitidas para tornar mais clara a apresentação).

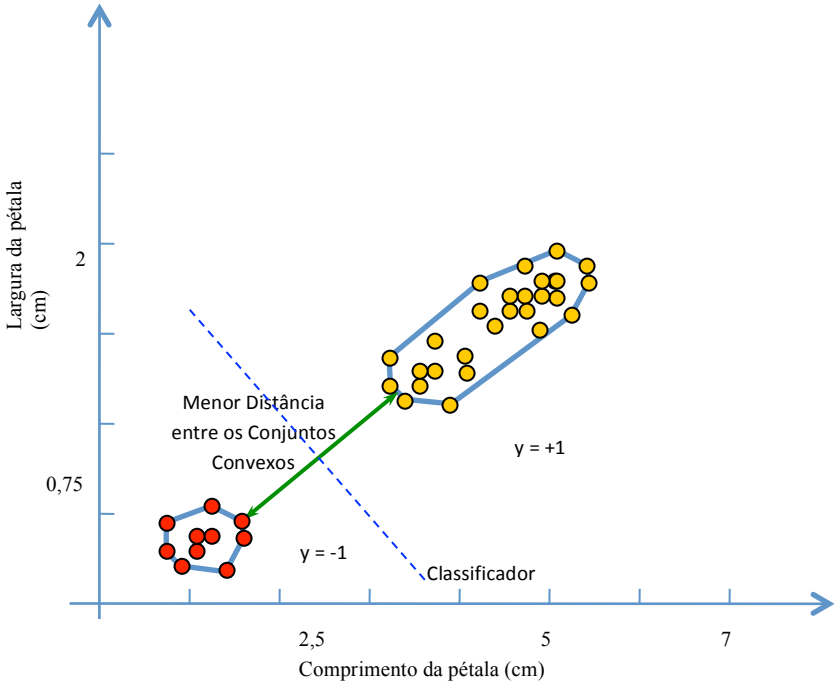


Figura 5.5 – Representação das Classes como Conjuntos Convexos.

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

A analogia para este procedimento é que o **Algoritmo de Aprendizado** pode ser visto como um método de otimização[7] de uma **Função Custo**. A cada iteração do processo de aprendizado a Função Custo é avaliada e, se seu valor decrescer, os pesos do vetor peso **w** são atualizados. O processo continua até encontrar o valor mínimo.

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

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

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

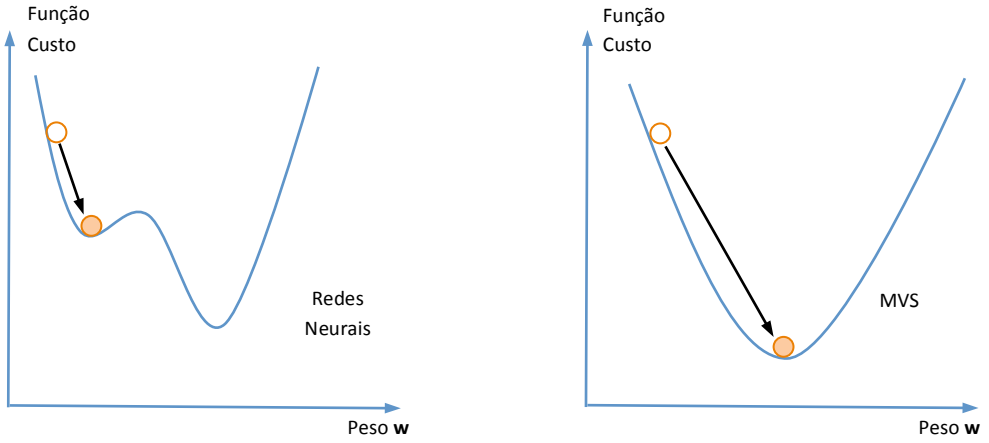


Figura 5.6 – Função Custo (a) Com e (b) Sem Mínimo Local.

Classificadores Não Lineares

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

O uso de funções polinomiais, além de apresentar risco de alto custo computacional, pode favorecer o ajuste excessivo da curva polinomial aos Vetores de Treinamento, podendo então ocorrer problemas de ***overfitting*** decorrentes da **instabilidade** causada pela enorme influência que um ou outro vetor pode ter sobre a curva polinomial. Já o princípio de **margem máxima** das MVS traz mais **estabilidade** na classificação, diminuindo as chances de *overfitting*.

Para entender melhor este comportamento, vamos primeiramente ver o que significa usar um *kernel*[8].

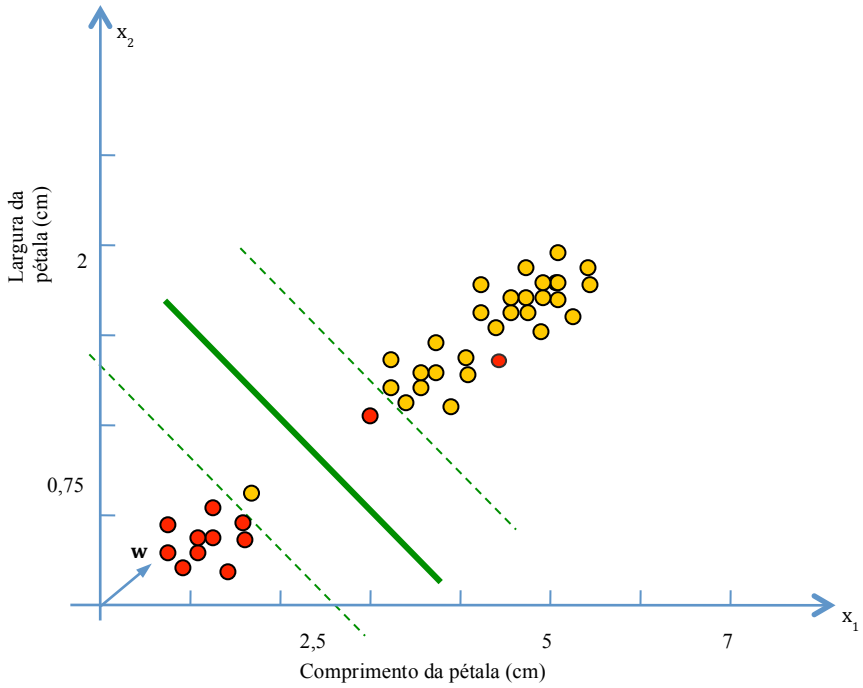


Figura 5.7 – Exemplo de Classes Não Separáveis Linearmente.

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

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

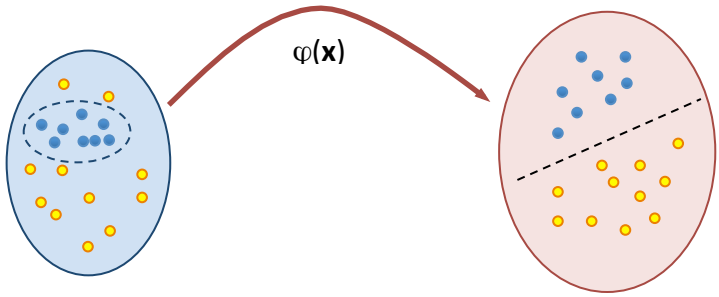


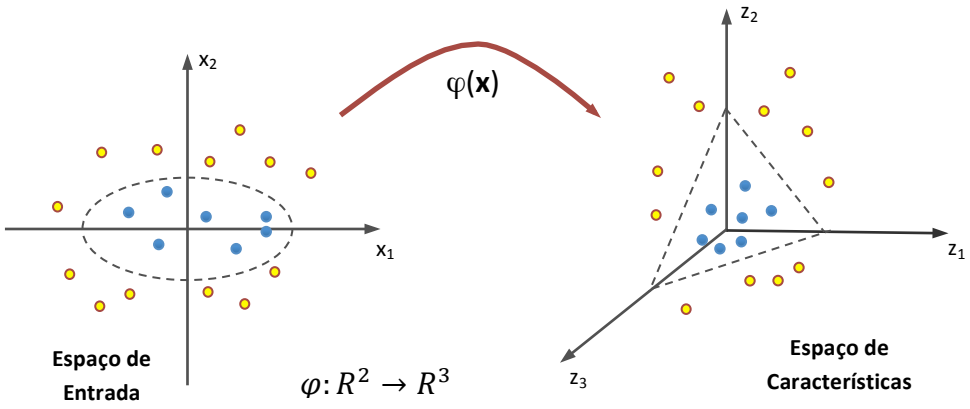
Figura 5.8 – Transformação Não Linear do Espaço de Entrada para o Espaço de Características.

Essa transformação em que os vetores são representados num espaço de dimensão mais elevada, geralmente é uma transformação não linear $\varphi(\mathbf{x})$. O espaço de partida dessa transformação não linear é conhecido como **Espaço de Entrada** e o espaço com dimensão mais alta é conhecido como **Espaço de Características**.

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

Vamos reproduzir um exemplo, apresentado por (SCHÖLKOPF & SMOLA, 2001), ilustrando como esta transformação não linear pode ser feita. Suponha um **Espaço de Entrada bidimensional**, composto por duas classes não separáveis linearmente. Aplicando-se o truque do *kernel*, vamos mapear esses dados num **Espaço de Características tridimensional**, como ilustrado na Figura 5.9.

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



$$(x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

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

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

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

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

Margem Suave Máxima

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

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

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

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

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

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

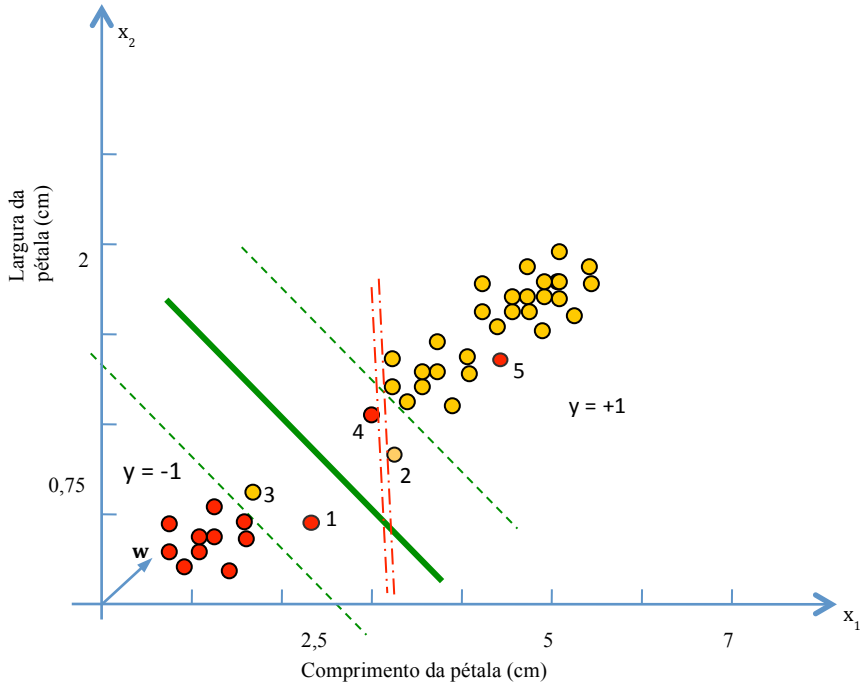
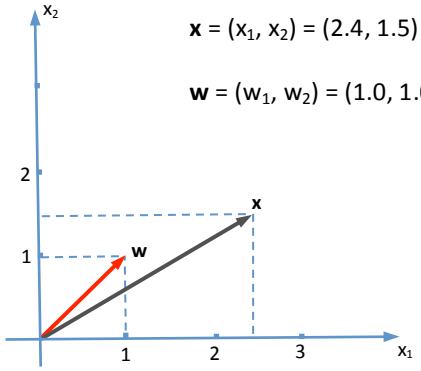


Figura 5.10 – Diferentes Margens com Diferentes Capacidades de Generalização.

Ferramental Matemático Básico

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



$\mathbf{x} = (x_1, x_2) = (2.4, 1.5)$

$\mathbf{w} = (w_1, w_2) = (1.0, 1.0)$

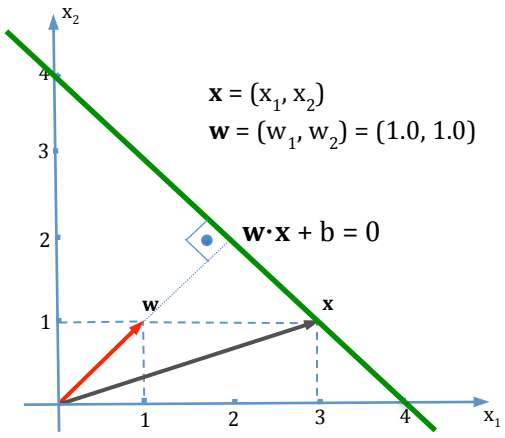
Produto Interno

$\mathbf{w}\cdot\mathbf{x} = (w_1 \cdot x_1 + w_2 \cdot x_2)$
$= (1.0 \cdot 2.4 + 1.0 \cdot 1.5)$
$= (2.4 + 1.5)$
$\mathbf{w}\cdot\mathbf{x} = 3.9$

Note que o resultado de um **Produto Interno** é um **valor escalar** e não um vetor!

Figura 5.11 – Produto Interno dos Vetores w e x.

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



Equação da Reta
w·x + b = 0

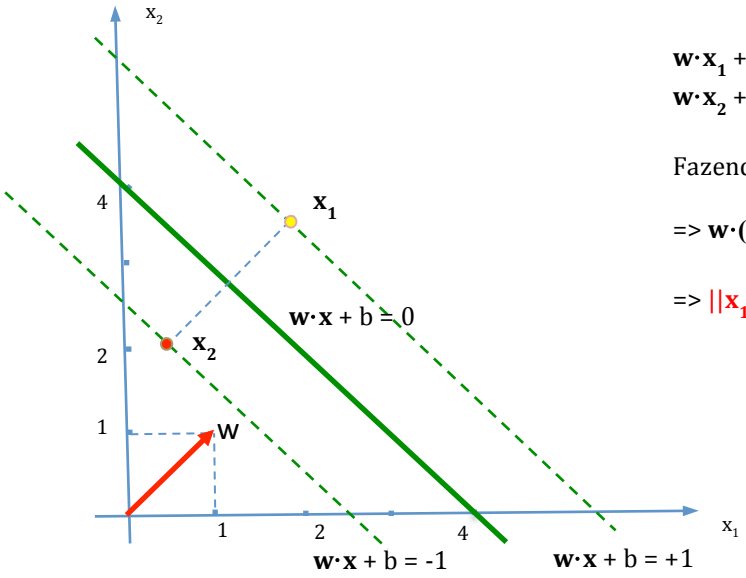
Suponha b = -4

Variando-se x_1 e x_2, obtém-se uma reta:

$x_1 = 0.0 \Rightarrow x_2 = 4.0$
$x_1 = 3.0 \Rightarrow x_2 = 1.0$

Figura 5.12 – Equação da Reta: w·x + b = 0.

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



$\mathbf{w}\cdot\mathbf{x}_1 + b = +1$ (1)

$\mathbf{w}\cdot\mathbf{x}_2 + b = -1$ (2)

Fazendo (1) - (2)

$\Rightarrow \mathbf{w}\cdot(\mathbf{x}_1 - \mathbf{x}_2) = +2$

$\Rightarrow ||\mathbf{x}_1 - \mathbf{x}_2|| = 2/||\mathbf{w}||$

Figura 5.13 - Equação do Classificador e Margens.

Como estamos interessados em maximizar as margens, ou seja, tornar o módulo da subtração de $\mathbf{x_1}$ por $\mathbf{x_2}$, $||\mathbf{x_1} - \mathbf{x_2}||$, o mais largo possível, isso equivale a minimizar o módulo do vetor peso **w**, já que

$$||\mathbf{x_1} - \mathbf{x_2}|| = 2/||\mathbf{w}|| \qquad (5.1)$$

A Equação mostra que quanto menor o módulo de **w**, mais largas serão as margens!

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

Existem técnicas matemáticas de otimização deste problema, por exemplo a **Programação Quadrática**, bastante conhecida entre os matemáticos, e que foram aproveitadas por (CORTES & VAPNIK, 1995) para desenvolver o algoritmo da MVS. Sucintamente o problema pode ser formalizado da forma exposta a seguir.

Dado um **Conjunto de Treinamento**: $(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2) \ldots (\mathbf{x_N}, y_N)$

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

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

Sob as restrições: $\forall i = 1 .. N, \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$

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

Há uma interessante implementação de MVS, proposta por (PLATT, 1998), conhecida como **SMO** *(Sequential Minimal Optimization)*, que em vez de considerar todos os Vetores de Treinamento conjuntamente, eles são divididos em pares e seus valores ótimos são deduzidos analiticamente. Embora o número de operações matemáticas aumente com relação à forma mais tradicional de resolver numericamente o problema com um sistema de equações, estas operações matemáticas da forma analítica são simples, operações aritméticas básicas, e portanto muito rápidas num computador. Dessa forma, o ganho no tempo total de computação pode ser significativo.

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

Como Visualizar as Bordas Decisão de uma SVM Usando o Weka

A ferramenta Weka (Weka, 2013) permite rodar várias implementações de **SVM**, ajustar o **Parâmetro de Complexidade C**, entre outros, e visualizar as **Bordas de Decisão** obtidas.

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

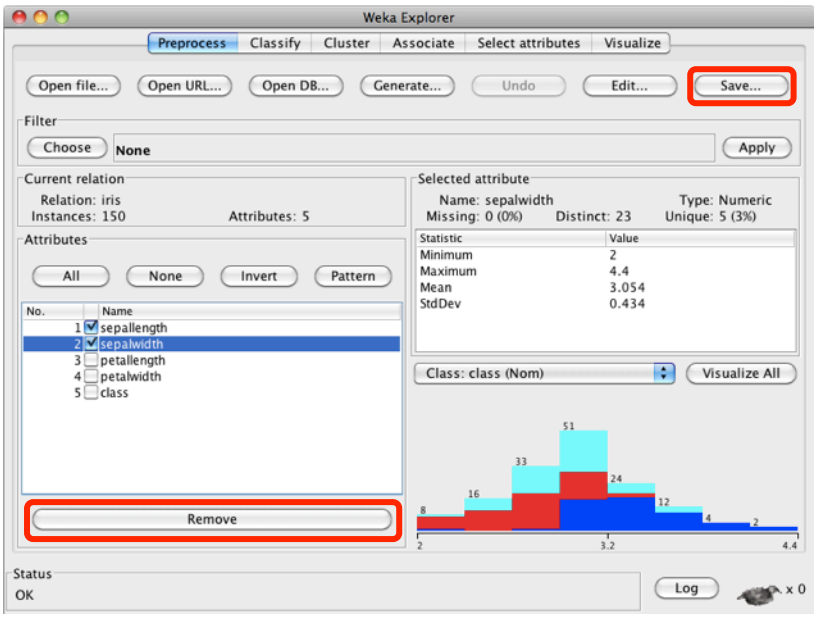


Figura 5.14 – Remoção de Atributos de um Arquivo “.arff”.

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

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

Figura 5.15 – Interface "Weka GUI Chooser".

Uma janela semelhante à mostrada na Figura 5.16 deve aparecer.

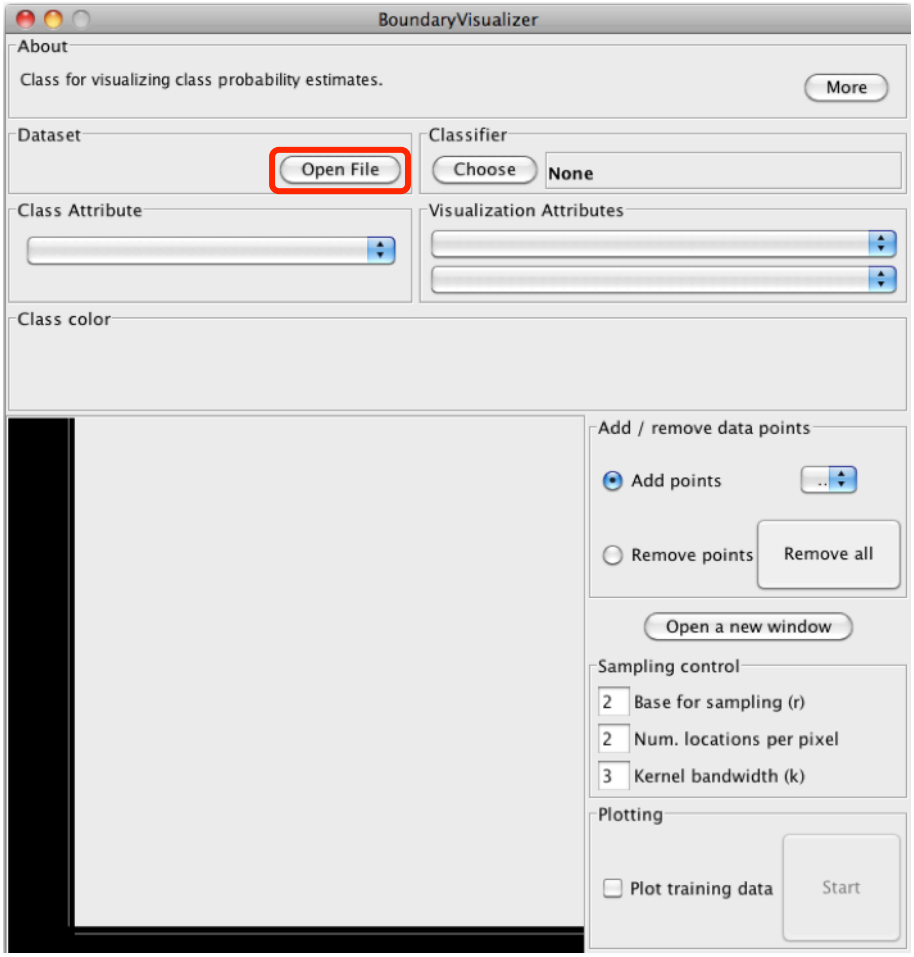


Figura 5.16 – Janela do "BoundaryVizualizer".

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

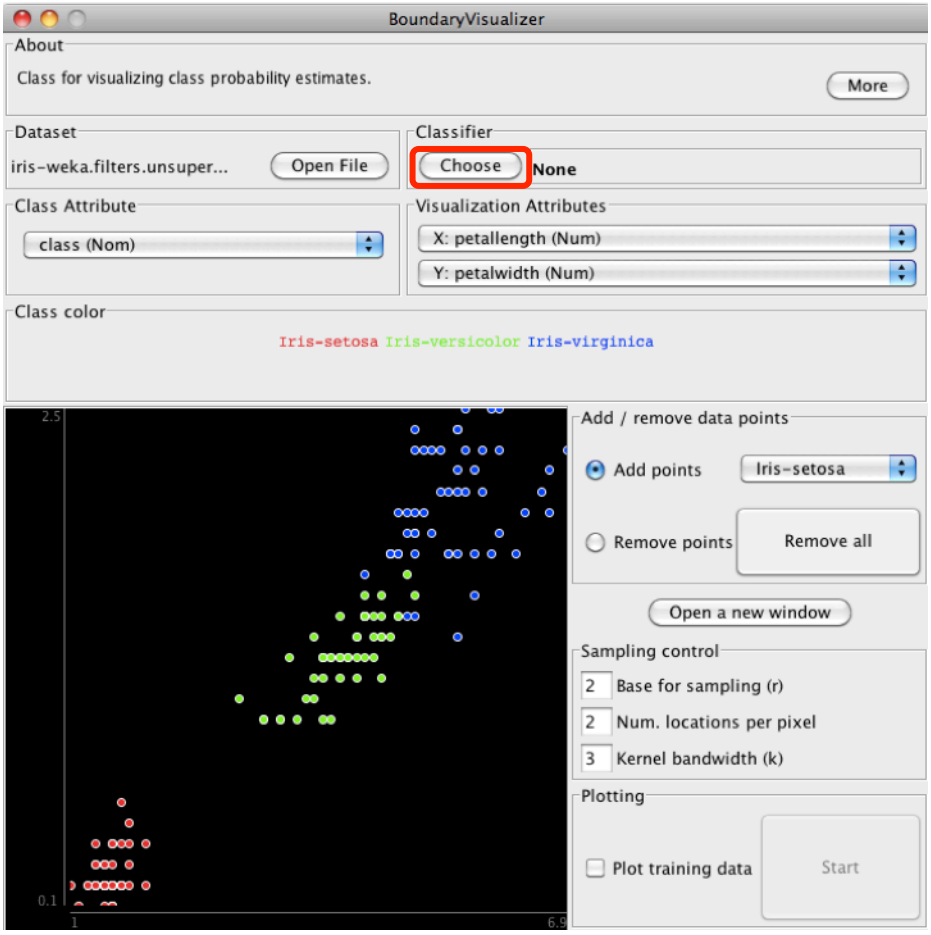


Figura 5.17 – Dados do Arquivo "íris_mod.arff" na Tela do "BoundaryVizualizer.

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

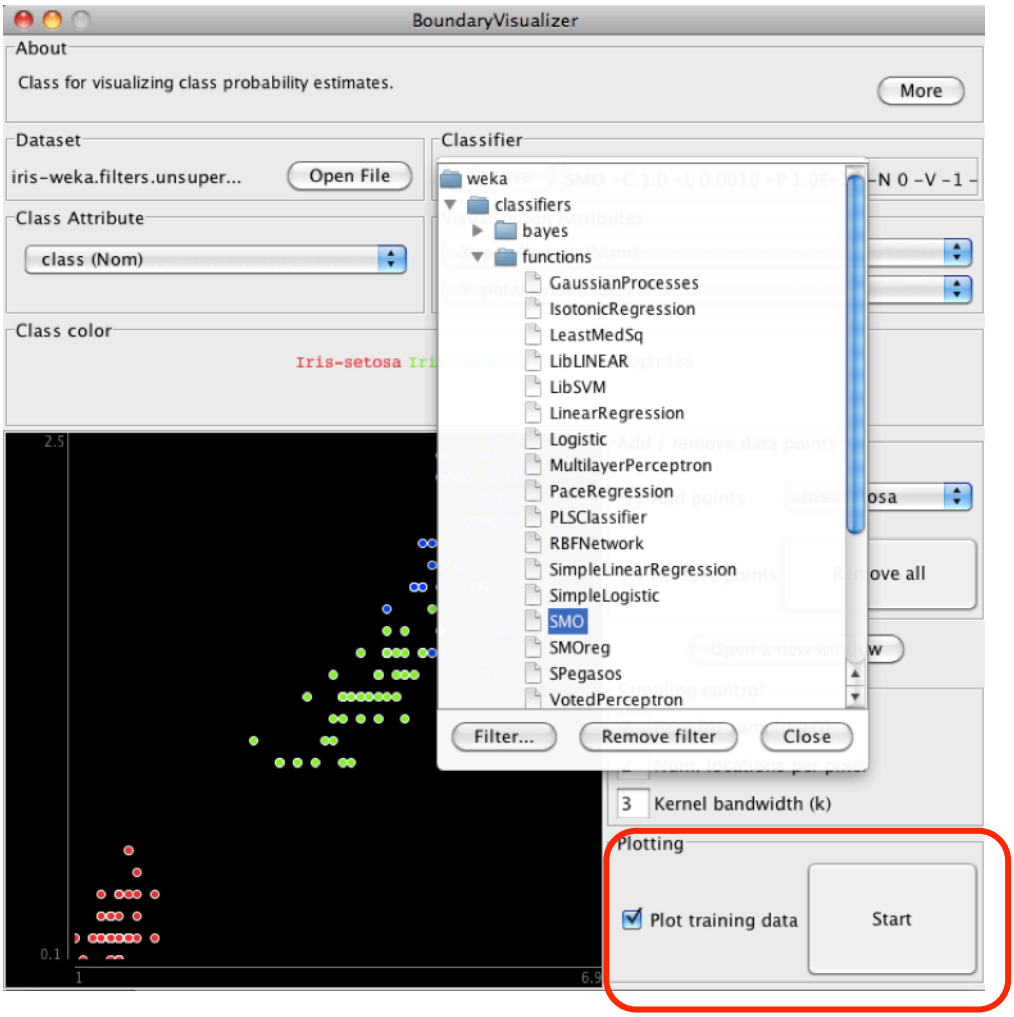


Figura 5.18 – Escolha do Algoritmo SMO.

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

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

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

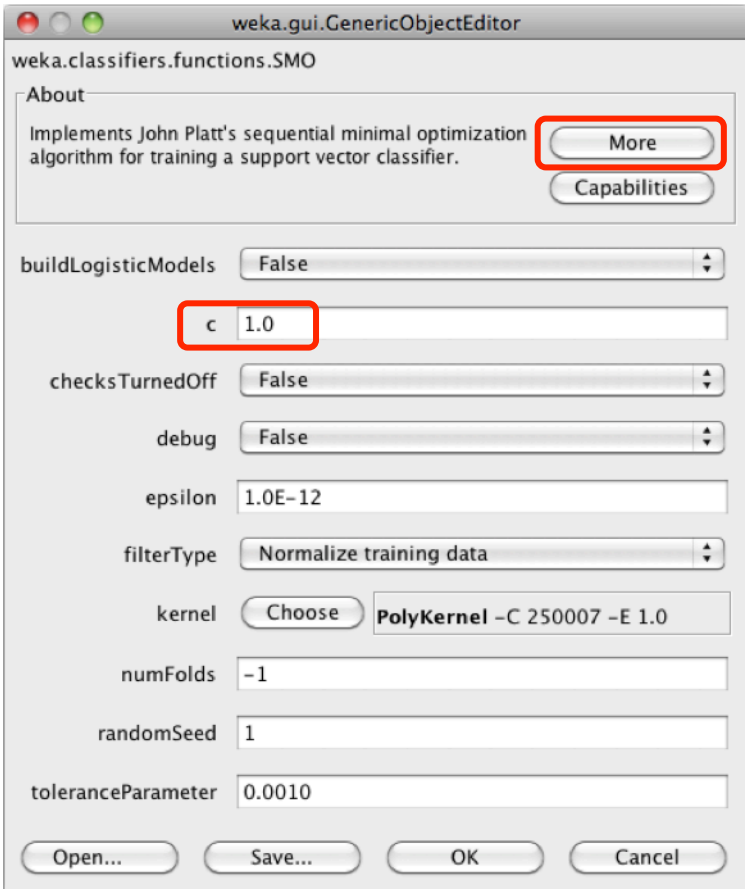


Figura 5.19 – Janela de Ajustes de Parâmetros do SMO.

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

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

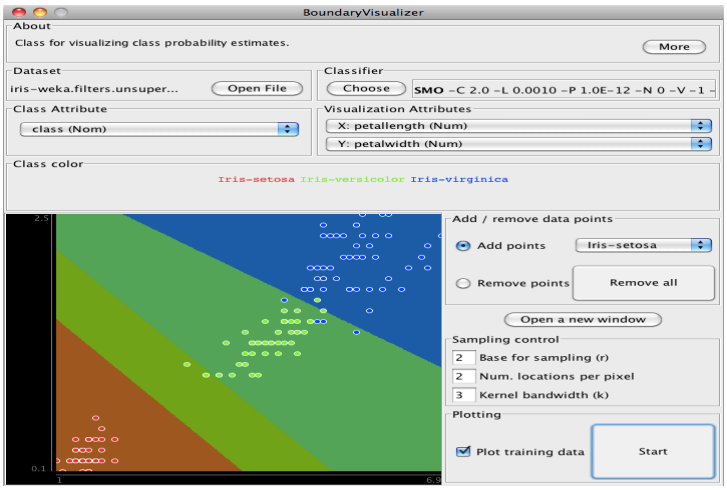


Figura 5.20 – Bordas de Decisão da Simulação para C = 2.0 e Kernel "PolyKernel".

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

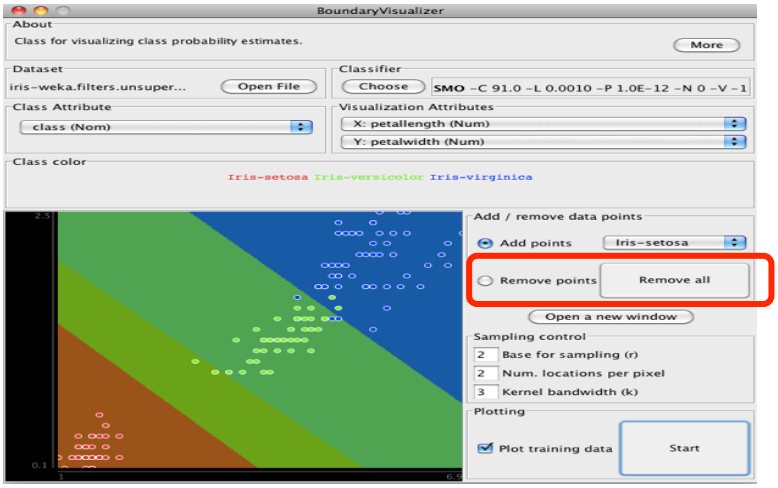


Figura 5.21 – Bordas de Decisão da Simulação para C = 91 e Kernel "PolyKernel".

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

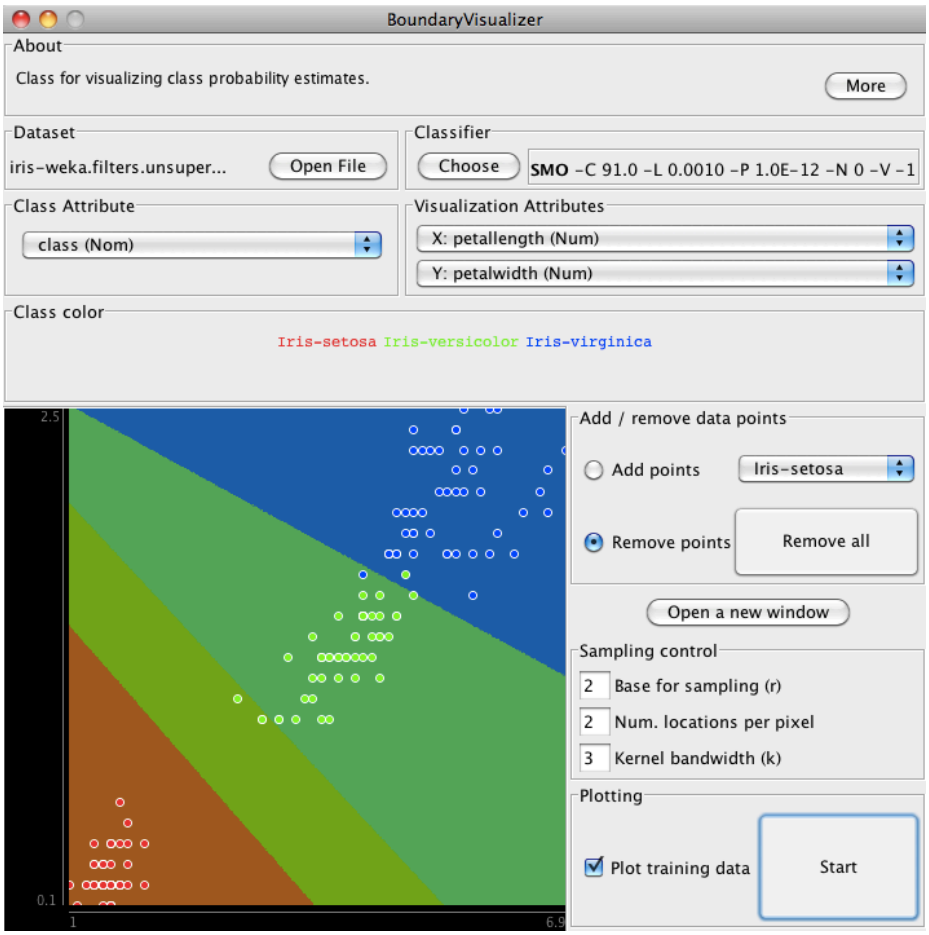


Figura 5.22 – Simulação com a Remoção de Alguns Pontos Perto das Bordas.

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

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

Considerações Finais

Nesta unidade, apresentamos um tipo de algoritmo de **Aprendizado Estatístico** conhecido como **Máquina de Vetores de Suporte** (**MVS** ou **SVM**).

Entre suas grandes vantagens, podemos citar:

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

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

Como desvantagem com relação aos algoritmos de Aprendizado Orientado a Conhecimento, podemos citar a dificuldade em interpretar os resultados do aprendizado, que na prática é um conjunto de pesos do vetor **w**, ou algo equivalente. Outro aspecto que pode ser visto como uma desvantagem das MVS em relação a outros modelos, é que não é possível incorporar Conhecimento do Domínio do Problema no Modelo gerado. Numa Rede Neural, por exemplo, a

topologia da rede reflete o Conhecimento do Domínio do Problema. Além disso, redes com multicamadas de neurônios podem aprender a ignorar Atributos irrelevantes (WITTEN & FRANK, 2005).

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

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

Lista de Exercícios

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

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

3. Carregue o arquivo **"iris.arff"** no Weka e faça a classificação com uma **Máquina de Vetores de Suporte** usando o algoritmo ***"SMO"***, (clique na aba "*Classify*", depois "*Choose*", escolha "*functions*" e clique em "SMO") com o método ***"Cross-validation"*** (10 *Folds*).

(a) Escolha valores para o **Fator de Complexidade *C*** entre 1 e 5 e analise os resultados da Matriz de Confusão. (Para ajustar o valor de ***C***, clique com o

botão esquerdo do mouse sobre a palavra "*SMO*", ao lado de "*Choose*", e mude o valor de **C** na janela que deve se abrir).

(b) Compare o desempenho dos *kernels* "***PolyKernel***" e "***RBFKernel***". (Para escolher o tipo de *kernel*, clique com o botão esquerdo do mouse sobre a palavra "*SMO*", ao lado de "*Choose*", e escolha o *kernel* na janela que deve se abrir).

Referência Bibliográfica

CORTES, C. & VAPNIK, V. **Support Vector Networks**. Netherlands: Machine Learning, 20, pp. 273-297, Springer Verlag, 1995.

PLATT, J. **Fast Training of Support Vector Machines Using Sequential Minimal Optimization**. http://research.microsoft.com/apps/pubs/?id=68391 . Acessado em 22.03.13.

REZENDE, S. O. (Organizadora). **Sistemas Inteligentes: Fundamentos e Aplicações.** Barueri: Editora Manole Ltda., 2005.

ROCHA, M.; CORTEZ, P. & NEVES, J. M. **Análise Inteligente de Dados: Algoritmos e Implementação em Java.** Lisboa: Editora de Informática, 2008.

ROSENBLATT, F. **The Perceptron: a Perceiving and Recognizing Automaton**. Report 85, pp. 460-1, Cornell Aeronautical Laboratory, 1957.

SCHÖLKOPF, B. & SMOLA, A. J. **Learning with Kernels**. Cambridge: Cambridge Press, 2001.

TAN, P.N.; STEINBACH, M. & KUMAR, V. **Introdução ao Data Mining Mineração de Dados.** Rio de Janeiro: Editora Ciência Moderna Ltda., 2009.

WITTEN, I. H. & FRANK, E. **Data Mining: Practical Machine Learning Tools and Techniques.** Second Edition. Amsterdam: Morgan Kaufmann Publishers, 2005.

Weka. The Waikato University. In http://www.cs.waikato.ac.nz/ml/weka. Acessado em 03.03.13.

WITTEN, I. H. & FRANK, E. **Data Mining: Practical Machine Learning Tools and Techniques.** Thirdy Edition. Amsterdam: Morgan Kaufmann Publishers, 2011.