Universidade Federal do Paraná – UFPR

Departamento de Engenharia Química

Introdução aos Algoritmos de Aprendizagem Supervisionada

Prof. Éliton Fontana

2020/1

Prefácio

Aprendizagem supervisionada e aprendizagem de máquina em geral se tornaram muito populares nos últimos anos devido ao seu grande potencial de aplicação nas mais diversas áreas. Uma das vantagens disto é que hoje dispõe-se de uma extensa literatura sobre o assunto, com vários livros com enfoque distinto, atendendo as necessidades de um público muito diversificado.

Dentre os materiais que serviram de base para a elaboração desta apostila, gostaria de destacar os ***excelentes*** livros de Deisenroth et al. (2020) e Bishop (2016). Ambos apresentam uma abordagem detalhada e intuitiva do funcionamento dos principais algoritmos, podendo ser facilmente indicados como *leitura obrigatória* para engenheiros que queiram se aprofundar nesta área. Para aqueles que por algum motivo não estejam interessados nos detalhes sobre os aspectos matemáticos de cada algoritmo (a.k.a. "*pessoas sem graça*"), uma recomendação é o livro de Géron (2019), que possui um bom balanço entre fundamentos teóricos e aplicação em Python usando o *sklearn*.

A existência desta extensa literatura relativamente recente faz com que, em muitos casos, não exista uma simboliga unificada. Ao longo deste material busquei manter a mesma simbologia em todos os tópicos, porém eventualmente podem haver alguns erros (associados à simbologia ou não) que tenham passado despercebidos[1]. Utilize este material com cautela e, na dúvida, consulte outra fonte (confiável) para comparar as informações.

Como material complementar, uma série de vídeos aplicando algoritmos de aprendizagem supervisionadas utilizando a extremamente útil biblioteca *Scikit-learn* está disponivel nesta playlist[2].

[1] Caso você tenha encontrado algum erro, por favor entre em contato (eliton.fontana@ufpr.br) para que eu possa corrigir em versões futuras.

[2] https://www.youtube.com/playlist?list=PL7C0pd9gDKkMVdnnxDTT4RQdupikYvSZ-

Contents

1 Introdução

Aprendizado de máquina (*machine learning*) é um termo geral utilizado para definir uma série de algoritmos que extraem informação a partir de um conjunto de dados, sem ser necessário definir um modelo matemático específico. A partir de um conjunto de dados de *treinamento*, estes algoritmos buscam um padrão relacionando entradas e saídas, permitindo utilizar este padrão para realizar predições. Dependendo da forma como estes dados são fornecidos, os algoritmos são classificados em diferentes categorias, sendo as principais apresentadas a seguir.

1.1 Aprendizagem Supervisionada

Os algoritmos de aprendizagem supervisionada relacionam uma saída com uma entrada com base em dados *rotulados*. Neste caso, o usuário alimenta ao algoritmo pares de entradas e saídas conhecidos, normalmente na forma de vetores. Para cada saída é atribuido um rótulo, que pode ser um valor numérico ou uma classe. O algoritmo determina uma forma de prever qual o rótulo de saída com base em uma entrada informada.

Por exemplo, uma mistura de ar e um combustível pode ou não entrar em combustão dependendo das condições do meio. Pode-se realizar uma série de experimentos variando parâmetros de interesse, como composição, pressão, velocidade, temperatura externa, etc., e para cada caso atribuir um rótulo "com combustão" ou "sem combustão". O algortimo pode então ser treinado com estes dados, sendo capaz de prever se haverá ou não combustão para uma dada condição de entrada. Este tipo de algoritmo, onde a saída pode assumir somente um conjunto de rótulos pré-definidos (e não um valor qualquer) são chamados de ***algoritmos de classificação***.

De maneira similar, o rótulo de saída pode ser um valor real, como por exemplo a temperatura do meio reacional. Neste caso, o algoritmo deve prever qual o valor desta temperatura

para uma dada condição de entrada, sendo que a saída pode assumir qualquer valor real. Estes algoritmos são chamados de ***algoritmos de regressão***.

1.2 Aprendizagem Não-Supervisionada

No caso dos algoritmos de aprendizagem não-supervisionada, não é atribuído um rótulo para os dados de saída. Com base em um número grande de dados, o algoritmo busca padrões e similaridades entre os dados, permitindo identificar grupos de itens similares ou similaridade de itens novos com grupos já definidos. Estes algoritmos podem ser divididos em ***algoritmos de transformação*** e ***algoritmos de agrupamento***.

Os algoritmos de transformação são utilizados para criar uma nova representação de um conjunto de dados que seja mais conveniente que a original, seja para facilitar a interpretação humana ou para melhorar o desempenho de outros algoritmos de aprendizagem.

Os algoritmos de agrupamento (*clustering*) particionam os dados em grupos com características similares com base em critérios pré-estabelecidos, permitindo encontrar padrões entre os dados fornecidos. Diversos métodos de agrupamento podem ser aplicados, podendo estes serem baseados na distância geométrica entre os pontos, em distribuições estatísticas específicas ou levar em conta a densidade de pontos em areas específicas do conjunto de dados.

Em alguns casos, pode-se aplicar um conjunto de dados onde somente parte destes dados é rotulada. Normalmente, somente uma pequena fração dos dados recebe um rótulo, porém isto tende a melhorar significativamente o desempenho dos algoritmos de aprendizagem não-supervisionada. Esta abordagem híbrida é normalmente chamada de ***aprendizagem semi-supervisionada***.

1.3 Aprendizagem por Reforço

Neste tipo de sistema, um agente realiza uma ação (dentre uma série de ações possíveis) em um ambiente e recebe uma recompensa de acordo com o resultado dessa ação, sendo o objetivo do algoritmo receber a maior recompensa possível. Estes sistemas possuem três componentes principais: o agente, o ambiente e a forma de interação entes estes dois.

O agente é o programa que está sendo treinado. De alguma forma, este agente precisa observar, interagir e modificar o ambiente ao longo do tempo. As etapas envolvidas costumam seguir a seguinte sequência:

1. O agente faz uma observação do ambiente;
2. O agente escolhe uma ação dentre diversas ações possíveis baseado na observação;
3. O agente entra em um estado de espera para que o ambiente envie novas observações;
4. O ambiente executa a ação recebida pelo agente e envia para o agente uma recompensa e a nova configuração do ambiente;
5. Repete-se as etapas 1-4.

Como a interação entre o agente e o ambiente ocorre de forma sequencial, é preciso operar de forma transiente, fazendo que a cada passagem pelo loop o tempo seja incrementado em um passo de tempo definido. Por definição, o agente inicia no tempo zero sem nenhuma forma de treinamento (sem saber como atingir o objetivo). Através da resposta do ambiente à ações do agente, um sinal de recompensa é gerado, sendo o objetivo do agente maximizar este sinal. A recompensa pode ser enviada no final de cada passo de tempo ou somente após uma determinda etapa ser atingida, dependendo das características do problema.

2 Algoritmos de Classificação

Algoritmos de classificação são algoritmos de aprendizagem supervisionada onde o objetivo é prever uma *classe* ou *rótulo* associado com uma variável de entrada contendo determinados *atributos*. Inicialmente, o algoritmo é treinado com um conjunto de dados com classes conhecidas, podendo estes dados estar divididos em somente duas (classificação binária) ou em várias classes (classificação multiclasse).

Para ilustrar o funcionamento básico de um algoritmo de classificação, a seguir será apresentado um exemplo considerando somente duas classes, pois isso permite a representação dos dados em um plano xy. Porém, é importante lembrar que estes algoritmos são utilizados principalmente quando se possuem múltiplas classes, sendo que neste caso a separação entre as classes não pode ser facilmente visualizada.

2.1 Aspectos Básicos dos Algoritmos de Classificação

Nesta seção será apresentado um exemplo simples de aplicação de algoritmos de classificação, com o objetivo de definir as etapas básicas envolvidas. Nas seções subsequentes serão abordados, com mais detalhes, os aspectos específicos do funcionamento dos algoritmos.

2.1.1 Importação dos Dados

Considere um processo onde é possível controlar a temperatura e a pressão, sendo que o resultado final pode ou não ser adequado, dependendo das condições impostas. Uma série de 200 ensaios foram realizados e os resultados foram rotulados como 0 para os casos não-adequados e como 1 para os adequados. Assim, neste exemplo os dados possuem dois atributos (temperatura e pressão) e estão divididos em duas classes (adequados e não-adequados, ou 1 e 0). Na Figura 2.1 são apresentados os dados de acordo com sua classificação.

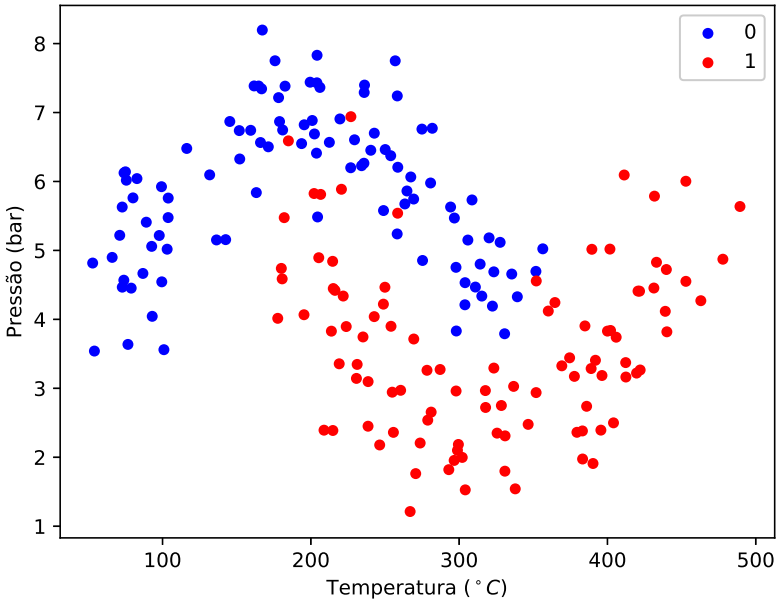


Figure 2.1: Conjunto de dados rotulados.

Antes de aplicar o algoritmo de classificação, é importante avaliar os dados em busca de possíveis erros. Além disso, é interessante avaliar a distribuição do número de pontos associados com cada variável. Por exemplo, pode-se ter muitos pontos associados com uma pequena faixa de temperatura e poucos pontos fora deste intervalo, o que pode prejudicar o desempenho do algoritmo. Isto pode ser analisado através de um histograma de distribuição de pontos, como mostrado na Figura 2.2.

Para a variável *classe*, existem exatamente 100 pontos para cada valor (0 ou 1), ou seja, 50% dos dados estão rotulados com adequados e 50% como não-adequados. Para a pressão e a temperatura, existe uma concentração maior de pontos para os valores médios, porém, a distribuição está relativamente homogênea.

2.1.2 Divisão em Grupos de Teste e Treinamento

Para garantir que o modelo ajustado aos dados possua uma precisão adequada, é necessário realizar algum teste de generalização para saber se o modelo é capaz de predizer corretamente o rótulo de novos dados. É importante que estes dados utilizados para testar o modelo não sejam os mesmos utilizados no ajuste (treinamento) do modelo, pois neste caso o algoritmo poderia simplesmente armazenar o rótulo destes valores e prever sempre corretamente.

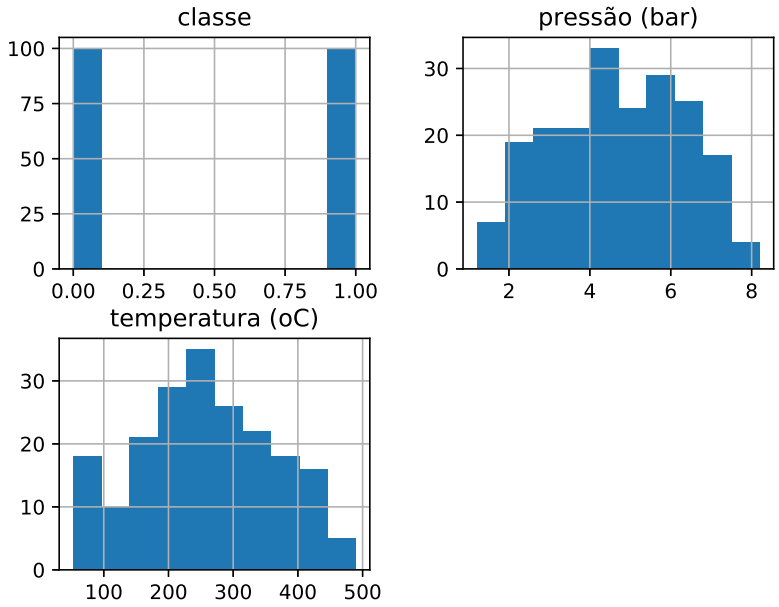


Figure 2.2: Histograma de distribuição dos pontos.

Para analisar o desempenho do modelo, é comum dividir o conjunto em dois grupos[1]: um grupo de *treinamento* e outro grupo de *teste*. De modo geral, pode-se escolher aleatoriamente em torno de 20-30% dos dados como parte do grupo de teste e o restante ser utilizado como grupo de treinamento. Como esta divisão é feita de forma aleatória, as medidas de desempenho do algoritmo podem variar dependendo de quais dados forem selecionados para cada grupo, especialmente se o conjunto de dados inicial for relativamente pequeno.

Neste exemplo, serão utilizados 20% dos dados para teste e os 80% restante para treinamento, ou seja, 40 pontos para teste e 160 para treinamento. Na Figura 2.3 é apresentado o grupo de teste obtido de forma aleatória. É importante que os dados não estejam concentrados em uma região específica do domínio, pois isso pode gerar uma falsa medida de desempenho do modelo em outras regiões. Comparando os dados das Figuras 2.1 e 2.3, pode-se observar que o grupo de teste está distribuído de forma aproximadamente homogênea ao longo de todo o domínio.

2.1.3 Aplicação do Algoritmo de Classificação e Análise dos Resultados

Após a separação dos dados, o grupo de treinamento pode ser aplicado para ajustar o algoritmo de classificação. Neste exemplo, será o utilizado o método dos k-vizinhos mais

[1]Pode-se também utilizar um terceiro grupo para validação de certos aspectos do modelo, mas inicialmente serão considerados somente 2 grupos.

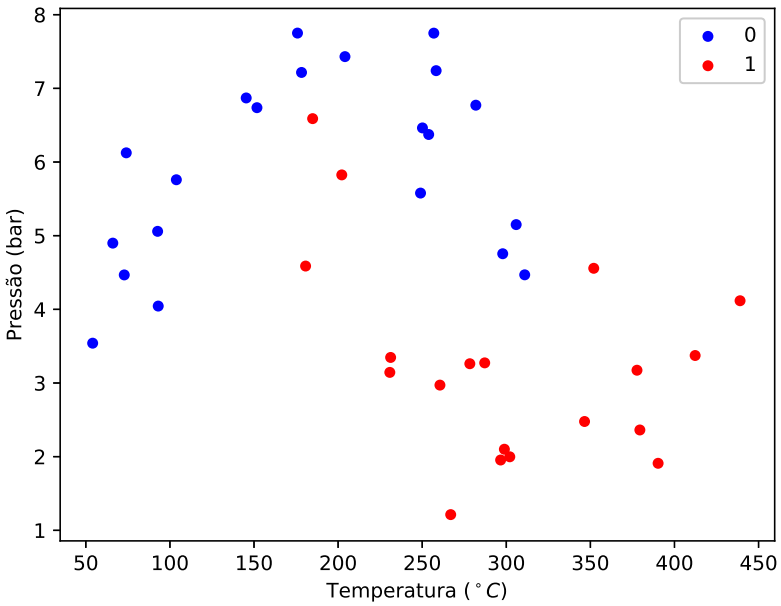


Figure 2.3: Grupo de teste considerando 20% dos dados.

próximos (k-NN - *K-Nearest Neighbors*), sendo este um dos métodos mais utilizados para classificação. Este método realiza a classificação com base na distância (geométrica) entre o ponto avaliado e os k vizinhos mais próximos que fazem parte do grupo de treinamento. O valor de k deve ser informado pelo usuário, sendo um importante parâmetro para o desempenho do método. Por exemplo, se $k = 1$ for utilizado, o algoritmo irá buscar, dentro do conjuto de treinamento, o ponto mais próximo àquele que está sendo avaliado e irá atribuir o mesmo rótulo a este ponto. Mais detalhes sobre este método serão apresentados na sequência. Neste exemplo, será considerado $k = 3$.

No caso do k-NN, o ajuste do modelo consiste basicamente em armazenar as posições e a classe de cada ponto do grupo de treinamento, para que na etapa de predição o rótulo de um dado ponto possa ser definido com base na sua posição e na classe dos vizinhos mais próximos. Após esta etapa, é realizada a predição dos dados do grupo de teste, ou seja, o algoritmo irá utilizar os valores de temperatura e pressão presentes neste grupo e tentará prever a qual classe estes pontos pertencem. Através da comparação com a classe real destes pontos (informada no conjundo de dados inicial), pode-se então estimar a precisão do algoritmo.

Existem diversas métricas que podem ser utilizadas para estimar o desempenho de um algoritmo de classificação, sendo elas baseadas nas quantidades de predições corretas e erradas dentro do grupo de teste. Para o exemplo deste algoritmo onde os dados são divididos em duas classes, pode-se definir os seguintes valores:

- Verdadeiros Positivos (TP): Valores da classe 1 preditos de forma correta;
- Verdadeiros Negativos (TN):Valores da classe 0 preditos de forma correta;
- Falsos Positivos (FP): Valores da classe 1 preditos de forma errada;
- Verdadeiros Negativos (FN):Valores da classe 0 preditos de forma errada;

A soma de todos estes conjuntos deve ser igual ao número total de pontos utilizados no grupo e teste. Uma maneira muito simples de representar estes valores é através da ***matriz de confusão*** (*confusion matrix*), que para este caso será definida como representado na Tabela 2.1[2].

Table 2.1: Definição da matriz de confusão

		Valores Previstos	
		Negativo	Positivo
Valores Reais	Negativo	TN	FP
	Positivo	FN	TP

Os valores encontrados para a matriz de confusão para este exemplo são apresentados na Tabela 2.2. Neste caso, 17 valores da classe 0 foram previstos corretamente (verdadeiros negativos), assim como 12 valores da classe 1 (verdadeiros positivos). Porém, 4 valores da classe 0 foram previstos como pertencentes a classe 1 (falsos positivos) e 7 valores da classe 1 foram previstos como 0 (falsos negativos). De forma geral, quanto maiores os elementos da diagonal principal da matriz de confusão, melhor será o desempenho do algoritmo.

Table 2.2: Matriz de confusão obtida no exemplo.

		Valore Previstos	
		Negativo	Positivo
Valores Reais	Negativo	17	4
	Positivo	7	12

Para quantificar o desempenho do algoritmo, uma das métricas mais utilizadas é a *acurácia*, que relaciona o total de acertos com o total de dados presentes no grupo de teste,

[2]Pode-se encontrar definições diferentes desta matriz dependendo da fonte utilizada.

pondendo ser definida como:

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Com base nos dados obtidos na matriz de confusão, a acurácia para este exemplo é de $(17 + 12)/40 = 0.725$, ou seja, somente 72.5% dos dados foram previstos corretamente. De maneira geral, este valor é relativamente baixo, porém isto está muito associado com a natureza dos dados e com os parâmetros utilizados no algoritmo. Nas próximas seções serão discutidas formas de melhorar este resultado.

Por último, pode-se utilizar o algoritmo para fazer previsões de dados que não tenham sido utilizados em nenhum dos grupos. Como neste caso é possível visualizar aproximadamente a região de separação entre as classes, é interessante avaliar alguns valores que estejam claramente em uma região com classe conhecida. Por exemplo, para $300°C$ e $2\,bar$, o algoritmo prevê a classe 1, o que está de acordo com o esperado com base na Figura 2.1.

É importante destacar que o algoritmo pode fazer previsões fora da região definida pelos dados de treinamento (neste caso, entre $20 - 500°C$ e $1 - 8\,bar$), porém, estes resultados tendem a ser muito menos confiáveis, então devem ser utilizados com cautela.

A implementação deste exemplo utilizando Python pode ser visualizada neste vídeo[3].

2.1.4 Extensão para Classificação Multi-Classe

Em problemas reais, é esperado que existam mais de dois parâmetros de entrada, bem como é possível que os dados sejam divididos em mais de duas classes. As etapas envolvidas na aplicação do algoritmo são as mesmas descritas anteriormente para a classificação binária, com algumas pequenas mudanças na forma como o resultado é avaliado.

Para ilustrar a aplicação em um problema multi-classe, será utilizado um conjunto de dados contendo 1600 elementos apresentado por Cortez et al[4], onde os autores relacionam 11 parâmetros físico-químicos de vinhos tintos com uma classificação de 3 a 8 obtida através de análise sensorial. Assim, os dados foram divididos em 6 classes.

Neste exemplo, não é possível visualizar o gráfico de dispersão dos pontos, pois os dados estão dispostos em um espaço vetorial com dimensão 11. Através da aplicação dos algoritmos

[3] https://www.youtube.com/watch?v=wMFqyOn2khA

[4] Cortez, et al. Modeling wine preferences by data mining from physicochemical properties, Decision Support Systems 47 (2009) 547-533

de classificação, pode-se obter hiperplanos de separação entre as classes, de forma semelhante a uma curva 1D de separação na Figura 2.1.

Da mesma forma que para a classificação binária, os dados devem ser divididos em um grupo de treinamento e outro de teste. Neste exemplo, serão utilizados 33% dos dados para teste, pois este valor apresentou uma acurácia maior. Para ilustrar, o algoritmo KNN também será utilizado, porém a aplicação considerando somente 3 vizinhos leva a uma acurácia de 0.48, enquanto que com 7 vizinhos este valor aumente para 0.525. Neste caso, a acurácia também é calculada como a razão entre o total de acertos pelo total de elementos no grupo de teste.

Esta acurácia pode parecer muito baixa, mas é interessante avaliar a matriz de confusão neste caso para verificar onde o classificação está errando. Como neste caso existem 6 classes, a matriz de confusão será uma matriz 6×6, relacionando os dados reais com as previsões, como mostrando na Tabela 2.3.

Table 2.3: Matriz de confusão para o exemplo de classificação da qualidade de vinhos.

		Valores Previstos					
		3	**4**	**5**	**6**	**7**	**8**
Valores Reais	**3**	0	0	1	2	1	0
	4	0	8	8	3	3	0
	5	0	1	165	70	7	0
	6	0	1	94	106	9	0
	7	0	3	22	22	9	0
	8	0	0	2	3	1	0

Neste caso, os elementos da matriz de confusão relacionam a quantidade real de elementos pertencentes a uma classe com a quantidade prevista de elementos para esta classe. A diagonal principal corresponde às predições corretas, enquanto elementos fora da diagonal principal são previsões erradas. Por exemplo, a linha 1 coluna 3 indica que houve 1 elemento pertencente à classe 3 que foi previsto como pertencente à classe 5. A maior parte das previsões erradas são adjacentes à diagonal principal, ou seja, o algoritmo erra em uma classe para mais ou para menos. Como estes valores representam uma quantificação da qualidade do produto, este erro em uma classe pode não ser muito significativo.

A implementação deste exemplo utilizando Python pode ser visualizada neste vídeo[5].

A seguir serão discutidos uma série de conceitos fundamentais relacionados a obtenção e aplicação de modelos capazes de fazer predições de classes, como o apresentado nesta seção.

2.2 Conceitos Fundamentais

Nesta seção serão apresentados alguns conceitos fundamentais aplicados aos algoritmos de classificação e de aprendizagem supervisionada em geral, com o objetivo de definir formalmente diversos conceitos que serão, na sequência, aplicados no estudo de algoritmos específicos. Uma análise mais detalhada destes conceitos pode ser encontrada em Deisenroth et al. (2020).

2.2.1 Representação dos Dados

Para a aplicação dos algoritmos de aprendizagem supervisionada em geral, é importante que os dados estejam em um formato adequado que possa ser interpretado pelo algoritmo. Em muitos casos, é preciso converter informações qualitativas em quantitativas, para que estas possam ser operadas numericamente. No exemplo anterior, isto foi utilizado para reprensetar duas classes, rotuladas inicialmente como "processo adequado" e "processo não-adequado", em um formato binário 0 ou 1[6].

Os atributos "temperatura" e "pressão" já estavam em um formato numérico adequado, por isso não foi necessário manipular os dados. Porém, quando os atributos possuem *ordens de grandeza* muito diferentes, uma grande melhoria no desempenho do algoritmo pode ser conseguida ***normalizando*** os dados, ou seja, deixando todos eles na mesma escala (ou pelo menos em uma escala próxima). Por exemplo, considere que um atributo esteja no intervalo $(0, 0.1)$, enquanto outro esteja no intervalo $(0, 10^5)$. Em operações que envolvam a manipulação destes atributos, é provável que a influência do segundo seja superestimada por conta da sua magnitude. Para eliminar este problema, pode-se definir os dois em uma escala similar, por exemplo, indo de -1 (valor mínimo) até $+1$ (valor máximo). Isto é semelhante ao processo de adimensionalização comumente utilizado em fenômenos de transporte, com a diferença que neste caso pode-se atribuir uma escala arbitrária.

[5] https://www.youtube.com/watch?v=YFQQAee-FI0

[6] De forma equivalente, poderiam ser atribuídos outros valores, como -1 e +1, por exemplo.

Em muitos casos, também pode ser necessário *extrair* os atributos a partir de um conjunto de dados, por exemplo, em sistemas onde os dados são informados como imagens. De alguma maneira, deve-se converter a informação contida nestas imagens em um atributo mensurável, o que requer um conhecimento do sistema avaliado.

Neste material, será considerado que os atributos são apresentado na forma de um vetor $\mathbf{x}$[7], enquanto que a classe associada a este vetor será identificada como y. Para algoritmos de classificação, os dados alimentados consistem em pares relacionando atributos e classes. Considerando um conjunto com m elementos, os dados devem ser informados como um conjunto da forma:

$$T = (\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots (\mathbf{x_m}, y_m) \tag{2.2}$$

Observe que enquanto a classe é um escalar (cada elemento pode pertencer a somente uma classe), os atributos são vetores com dimensão finita, por exemplo, considerando que existam n atributos, $\mathbf{x_i} = (x_{i,1}, x_{i,2}, \ldots, x_{i,n})$.

2.2.2 Obtenção de Preditores

Partindo dos dados expressos em um formato adequado, pode-se treinar o algoritmo para prever a qual classe irá pertencer um novo elemento $\mathbf{x}'$, sendo neste caso o modelo é chamado de *preditor*. Esta mesma lógica pode ser aplicada para algoritmos de regressão, com a única diferença que neste caso a saída não será uma classe e sim um valor real qualquer.

Os preditores podem ser obtidos em duas diferentes formas, dependendo do algoritmo utilizado. O primeiro caso é a obtenção de uma *função preditora*, que é uma função f que recebe como argumento de entrada um vetor com dimensão n e gera como saída um escalar (classe para classificação ou valor real para regressão), ou seja:

$$f : \mathbb{R}^n \to \mathbb{R} \tag{2.3}$$

Na maioria dos casos esta função é uma reta do tipo:

$$f(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \boldsymbol{\theta_0} \tag{2.4}$$

sendo que os coeficientes angular $\boldsymbol{\theta}$ e linear $\boldsymbol{\theta_0}$ devem ser ajustados. Exemplos de algoritmos que operam desta forma são o $k - NN$ utilizado anteriormente e as *máquinas de vetores de suporte* que serão apresentadas na sequência.

[7] Por padrão, variáveis em negrito serão utilizadas para representar vetores.

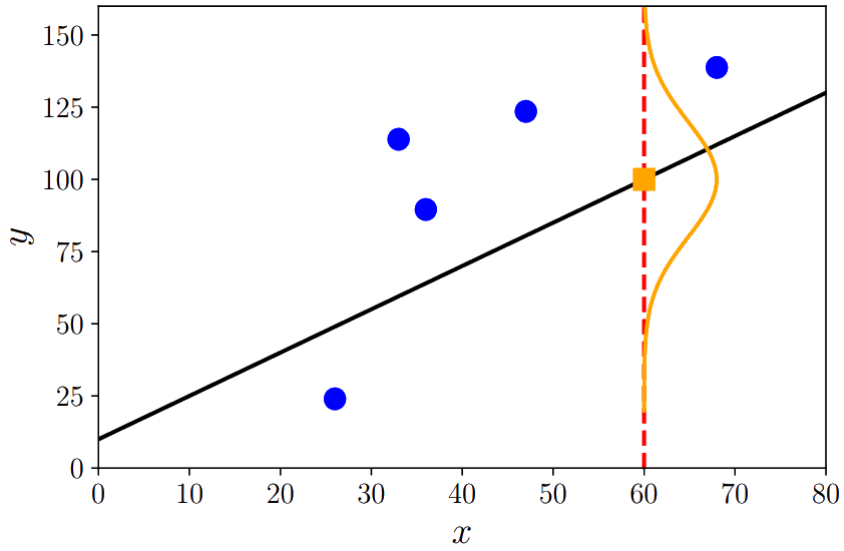


Figure 2.4: Comparação entre uma função preditiva (linha preta) e um modelo probabilístico (linha amarela). Fonte: Deisenroth et al. (2020)

.

A segunda possibilidade é que os modelos preditivos sejam obtidos como *modelos probabilísticos*, ou sejam, expressam a probabilidade de um dado elemento pertencer a uma certa classe ou estar associado com um certo valor. Estes modelos são especialmente úteis quando os dados de entrada possuem muito ruído, pois permitem ponderar o efeito destes pontos. Como exemplo de modelos probabilísticos, pode-se destacar os baseados em Naive Bayes, que serão discutidos posteriormente.

Na Figura 2.4 é apresentada uma comparação entre as duas abordagens para um problema de regressão. No caso de uma função preditiva, obtém-se a reta em preto, correspondendo a uma função que associa cada ponto do domínio x com uma única imagem y. No caso do modelo probabilístico (linha amarela), obtém-se uma *função densidade de probabilidade* que relaciona a probabilidade de a saída y assumir um certo valor para uma entrada $x = 60$.

Para obter os parâmetros destes modelos, deve-se resolver um problema de otimização onde o erro associado aos dados deve ser de alguma forma estimado e minimizado. A seguir serão discutidos dois procedimentos utilizados para a obtenção de parâmetros de funções preditoras, o princípio de minimização do risco empírico e o princípio da máxima verossimilhança. A abordagem utilizada para modelos probabilísticos será discutida posteriormente, juntamente com os algoritmos baseados em Naive Bayes.

2.2.3 Minimização do Risco Empírico

O princípio de minimização do risco empírico é utilizado para a obtenção dos parâmetros relacionados com funções preditoras. Considere novamente um conjunto de dados como apresentado na Eq. 2.2, contendo m elementos, onde cada elemento é composto por um par vetor de dimensão n ($\mathbf{x}$) e classe (y). O objetivo é encontrar uma função preditora que, com base em um conjunto de parâmetros $\boldsymbol{\theta}$ e no valor de um vetor $\mathbf{x}$, seja capaz de prever a classe y associada com $\mathbf{x}$, ou seja:

$$f(\mathbf{x_i}, \boldsymbol{\theta}) \approx y_i \qquad\qquad i = 1, 2, 3, \ldots m \tag{2.5}$$

Considere que o valor previsto por esta função seja representado como $\hat{y}_i$, ou seja, $\hat{y}_i = f(\mathbf{x_i}, \boldsymbol{\theta})$. Para avaliar a diferença entre estes valores previstos $\hat{y}_i$ e os valores reais y_i, utiliza-se uma função para mensurar o erro associado com cada predição. Esta função é chamada de *função de perda*, sendo representada como $\ell(\hat{y}_i, y_i)$. Como saída desta função, obtém-se um escalar positivo, chamado de *perda*, que representa uma estimativa do erro[8]. Os parâmetros $\boldsymbol{\theta}$ utilizados na função preditora devem ser valores que minimizem esta função de perda para o conjunto de valores.

Estimativa do Erro Empírico

Uma hipótese normalmente utilizada em algoritmos de aprendizagem é que os elementos do conjunto de dados são *independentes* entre si, o que implica que a média amostral (também conhecida como *média empírica* é uma boa estimativa da média real da população. Para um vetor $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ qualquer, esta média é definida como:

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.6}$$

Considerando a mesma hipótese para a função de perda, pode-se assumir que a média empírica das perdas é uma boa aproximação da perda associada com todo o conjunto de dados:

$$\bar{\ell} = \frac{1}{m} \sum_{i=1}^{m} \ell(\hat{y}_i, y_i) \tag{2.7}$$

Como a função $\ell(\hat{y}_i, y_i)$ depende tanto da função preditora f quanto dos dados utilizados, a média $\hat{\ell}$, esta relação pode ser expressa em termos de uma *função de risco empírico* $\mathbf{R}_{emp}$,

[8] Veja que está função tem relação com a definição de acurácia discutida anteriormente.

da seguinte forma:

$$\mathbf{R}_{emp}(f, \mathbf{X}, \mathbf{y}) = \frac{1}{m} \sum_{i=1}^{m} \ell(\hat{y}_i, y_i) \tag{2.8}$$

onde $\mathbf{X}$ representa uma matriz com dimensão $n \times m$ contendo todos os vetores $\mathbf{x}$ da forma $\mathbf{X} = (\mathbf{x_1}, \mathbf{x_1}, \ldots, \mathbf{x_m})^T$ e $\mathbf{y} = (y_1, y_2, \ldots, y_m)^T$ é um vetor contendo todas as classes. A estratégia geral de minimizar esta função é chamada de *minimização do risco empírico.*

O problema em utilizar o risco empírico nesta forma para obter os parâmetros $\boldsymbol{\theta}$ que minimizem $\mathbf{R}_{emp}(f, \mathbf{X}, \mathbf{y})$ é que esta função considera somente dados já rotulados, não traz nenhuma informação sobre como o algoritmo irá desempenhar para classificar novos dados. Na prática, busca-se um classificador com boa capacidade de prever a classe de elementos novos, que não tenham sido usados no treinamento. Neste caso, diz-se que o classificador tem boa capacidade de ***generalização***. Por isso, deve-se buscar alguma forma de estimar um erro que leve em conta somente a função preditora f e não o conjunto de dados utilizados.

Estimativa do Erro Esperado

Uma maneira de avaliar o erro real esperado seria supor que temos acesso a um número infinito de amostras rotuladas. Com isso, seria possível eliminar a influência das amostras individuais e ter um valor real para a média. De forma semelhante ao erro empírico, pode-se definir uma função *erro esperado* $\mathbf{R}_{exp}$ da seguinte forma:

$$\mathbf{R}_{exp}(f) = \lim_{k \to \infty} \left(\frac{1}{k} \sum_{i=1}^{k} \ell(y_i, f(\mathbf{x_i})) \right) \tag{2.9}$$

onde y_i representa a classe real das amostras e $f(\mathbf{x_i})$ o valor previsto. Veja que neste caso é considerado que os parâmetros $\boldsymbol{\theta}$ da função f estejam fixados, ou seja, o treinamento do algoritmo já foi realizado. Claramente esta definição não possui aplicação prática, visto que a quantidade de dados disponíveis é finita.

Considerando que temos a disposição um conjunto finito de dados, como comentado no exemplo resolvido anteriormente, uma maneira de avaliar o desempenho do algoritmo em dados "não-vistos" durante o treinamento é separar os dados em um conjunto de treinamento e um conjunto de teste. Neste caso, os dados $(\mathbf{X}, \mathbf{y})$ são divididos em $(\mathbf{X}_{treino}, \mathbf{y}_{treino})$ e $(\mathbf{X}_{teste}, \mathbf{y}_{teste})$.

O treinamento do algoritmo é feito de modo a encontrar os parâmetros $\boldsymbol{\theta} \in \mathbb{R}^n$ que minimizem o risco empírico associado com o conjunto de treinamento, o que pode ser expresso

como:

$$\min_{\boldsymbol{\theta}} \mathbf{R}_{emp}(f, \mathbf{X}_{treino}, \mathbf{y}_{treino}) \tag{2.10}$$

Mesmo que este valor seja minimizado, isto não garante que o algoritmo será capaz de classificar bem novos dados, ou seja, tem capacidade de generalização. Para avaliar isto, pode-se estimar o erro esperado utilizando o erro empírico associado ao conjunto de teste:

$$\mathbf{R}_{exp} \approx \mathbf{R}_{emp}(f, \mathbf{X}_{teste}, \mathbf{y}_{teste}) \tag{2.11}$$

A princípio, se o erro empírico associado aos dois conjuntos for baixo, isto indica que o algoritmo foi bem ajustado e possui boa capacidade de generalização. Porém, neste ponto é importante levantar duas questões:

1. Como proceder caso o risco empírico associado ao conjunto de treinamento for baixo (ou seja, o algoritmo está bem ajustado), porém, o risco associado ao conjunto de teste for alto?

2. Como garantir que o conjunto de teste escolhido gera uma boa representação do erro esperado?

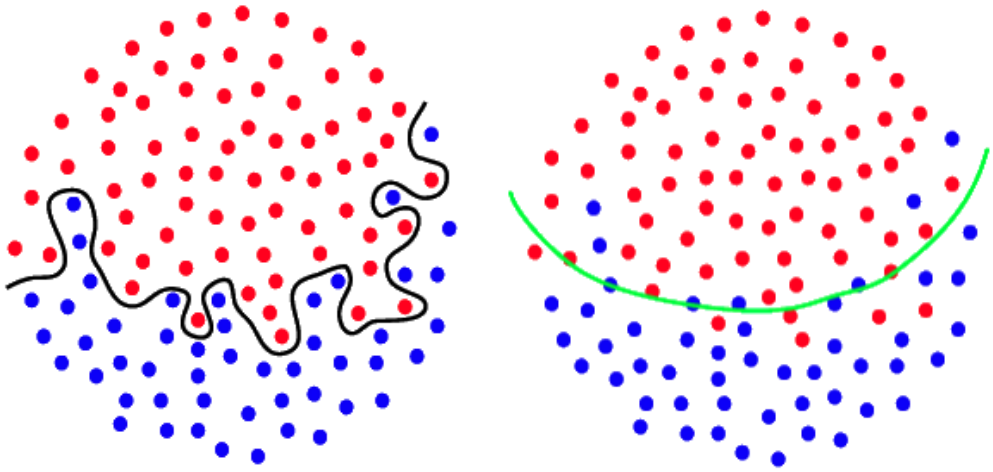
A primeira pergunta está associada com um conceito muito importante na área de aprendizagem que é a noção de sobreajuste (*overfitting*), que pode ser contornada com uma estratégia de *regularização*. A segunda pergunta pode ser respondida utilizado uma estratégia de *validação cruzada*. Estas duas abordagens serão discutidas a seguir.

Sobreajuste e Regularização

Modelos de ajuste complexos são capazes de detectar pequenas variações no conjunto de dados. Porém, se estas variações forem causadas por ruídos, o modelo acaba se ajustando a um padrão definido pelos ruídos e perde a capacidade de generalização. Este problema ocorre principalmente se o conjunto de treinamento possuir muito ruído ou for muito pequeno. O sobreajuste também pode ocorrer em problemas de regressão.

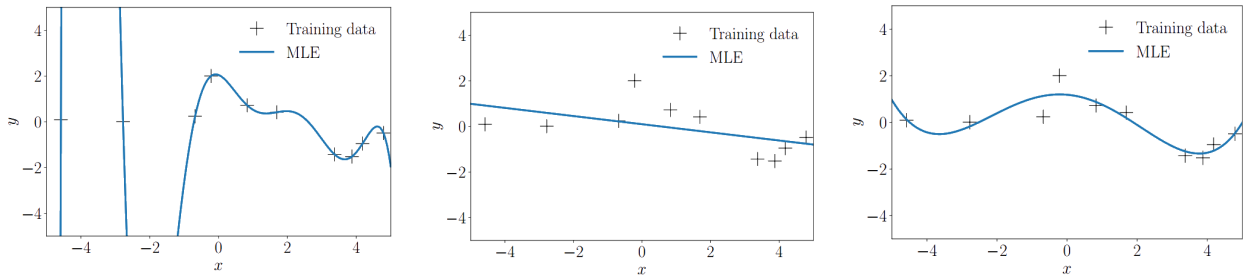
Por exemplo, considere os exemplos de sobreajuste ilustrados na Figura 2.5(a) para classificação e na Figura 2.6(a) pra regressão. No caso da classificação, o algoritmo é capaz de separar os dados perfeitamente nas duas classes, porém é muito provável que em um processo físico real esta fronteira irregular seja causada por ruídos na obtenção dos dados. De forma semelhante, para o exemplo da regressão, a curva consegue cruzar todos os pontos do

conjunto de treinamento, porém, em alguns intervalos entre os pontos, assume valores muito distantes dos pontos, o que dificilmente irá ocorrer em um fenômeno físico.



(a) Sobreajuste (b) Regularizado

Figure 2.5: Exemplos de sobreajuste e regularização em um algoritmo de classificação. Fonte: Hulsen et al. (2019)



(a) Sobreajuste (b) Subajuste (c) Regularizado

Figure 2.6: Exemplos de sobreajuste e regularização em um algoritmo de regressão. Fonte: Deisenroth et al. (2020)

O fenômeno de sobreajuste leva a uma baixa capacidade de generalização, já que existe uma grande chance de que dados não utilizados no treinamento não sejam avaliados corretamente. Por isso, o erro empírico relacionado com o conjunto de teste tende a ser alto.

Uma estratégia muito utilizada para reduzir o problema de sobreajuste é a introdução de um termo de *penalização* na função objetivo, de modo que o otimizador perca parte da flexibilidade de encontrar valores muito grande para os parâmetros. Este procedimento é chamado de ***regularização***. O problema de otimização (Eq. 2.10) regularizado por ser

expresso como:

$$\min_{\boldsymbol{\theta}} \left(\mathbf{R}_{emp}(f, \mathbf{X}_{treino}, \mathbf{y}_{treino}) + \lambda ||\boldsymbol{\theta}||^2 \right) \tag{2.12}$$

onde λ é chamado de *parâmetro de regularização* e o termo $||\boldsymbol{\theta}||^2$ de *regularizador*. Veja que neste caso o algoritmo é penalizado caso os valores do vetor $bm\theta$ sejam, em magnitude, muito grandes. Esta estratégia costuma funcionar pois, de modo geral, estes valores tendem a ser elevados no caso de sobreajuste.

A regularização é uma maneira de ponderar entre a complexidade do modelo e a capacidade de generalização. Considerando que o parâmetro de regularização seja ajustado corretamente, obtém-se uma resposta menos sensível a ruídos, como ilustrado na Figura 2.5(b) para um exemplo de classificação na na Figura 2.6(c) para regressão.

Em alguns casos, também pode ocorrer o problema oposto, ou seja, o modelo pode ser simplificado demais, ao ponto de não conseguir capturar o real comportamento dos sistema. Por exemplo, considere um caso onde o conjunto de dados foi gerado por um problema periódico, descrito aproximadamente por uma função seno. Se o modelo tentar ajustar estes pontos a uma reta, a informação referente à oscilação será perdida, o que costuma ser chamado de subajuste (*underfitting*). Um exemplo de subajuste é ilustrado na Fig. 2.6(b).

Validação Cruzada

Considerando que o problema de sobreajuste pode ser contornado regularizando o otimizador, pode-se partir agora para o segundo problema: como escolher um conjunto de teste significativo? A abordagem de utilizar parte dos dados para verificar a capacidade de generalização do algoritmo é chamada de *validação cruzada*[9].

Infelizmente, não existe como garantir que os dados reservados para teste irão de fato gerar uma representação satisfatória do desempenho do algoritmo, indicando, por exemplo, se existe sobreajuste ou não. Ao invés disso, o que pode ser utilizado é uma abordagem onde os dados são separados de uma forma um pouco diferente, em dados para treinamento e dados para *validação*. O conjunto de validação é um subconjunto dos dados de treinamento utilizado para avaliar o desempenho do algoritmo *durante* a etapa de obtenção dos parâmetros, ao contrário do conjunto de testes que só é aplicado após os parâmetros serem definidos.

[9] O método onde os dados são simplesmente divididos em conjunto de treinamento e de teste costuma ser chamado de *Método Holdout*.

A forma mais utilizada de validação cruzada utilizado conjuntos de validação é o *método K-fold*, onde o conjunto de dados de treinamento é dividido em K subconjuntos, onde $K-1$ subconjuntos irão formar o conjunto de treinamento e o subconjunto restante irá formar o conjunto de validação. A ideia geral, neste caso, é repetir o processo de obtenção dos parâmetros estimativa do erro K vezes, sempre mudando qual subconjunto será utilizado para validação. Por exemplo, na Figura 2.7 é ilustrada uma divisão em 5 subconjuntos (método 5-fold).

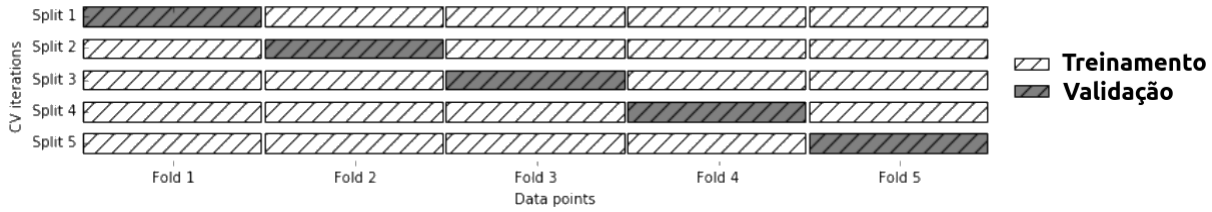


Figure 2.7: Divisão dos dados em conjuntos de treinamento (branco) e de validação (cinza) considerando uma abordagem 5-fold. Fonte: Muller and Guido (2017)

Neste exemplo, utilizado um método 5-fold, para cada uma das 5 separações, o conjunto de treinamento é utilizado para obter a função preditora f, que por sua vez é aplicada no conjunto de validação para computar o risco empírico. Após fazer este procedimento em todas as 5 combinações, o erro estimado do predito é avaliado como a média dos erros empíricos de cada combinação. De forma semelhante, os parâmetros $\boldsymbol{\theta}$ serão aproximados como a média dos parâmetros obtidos para cada combinação.

A validação cruzada utilizando um método k-fold é uma maneira de aplicar o conceito de minimização do risco empírico e evitar problemas de sobreajuste, sendo uma abordagem muito aplicada para a obtenção de funções preditoras. Outra abordagem que pode ser utilizada para estimar os parâmetros de um modelo é a partir do princípio de máxima verossimilhança, como ser apresentado a seguir.

2.2.4 Princípio da Máxima Verossimilhança

A obtenção dos parâmetros utilizando máxima verossimilhança baseia-se em definir uma função dos parâmetros que permita estimar quão bem os dados estão ajustados. Por exemplo, considere que o modelo possua um conjunto de parâmetros definidos pelo vetor $\boldsymbol{\theta}$. O objetivo é que, dado um vetor de atributos $\boldsymbol{x}$, o modelo seja capaz de prever a classe y.

A *probabilidade* de que o modelo preveja uma classe y_i, para um dado vetor $\boldsymbol{x}_i$ e um conjunto de parâmetros $\boldsymbol{\theta}$ é representada como $p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})$. Esta função (também chamada de função de verossimilhança) pode ser avaliada assumindo alguma forma de distribuição de densidade de probabilidade específica, como por exemplo uma distribuição normal:

$$p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - f(\boldsymbol{x}_i, \boldsymbol{\theta}))^2}{2\sigma^2}\right) \tag{2.13}$$

Utilizando uma distribuição neste formato, pode-se buscar valores de $\boldsymbol{\theta}$ que maximizem a chance da classe y_i ser prevista corretamente.

A expressão anterior é válida para um dos elementos $\mathbf{x}_i$ do conjunto de dados. Para obter os parâmetros $\boldsymbol{\theta}$ que maximizem a chance de acerto para todo o conjunto de dados $\mathbf{X} = (\mathbf{x_1}, \mathbf{x_1}, \ldots, \mathbf{x_m})^T$ com classes $\mathbf{y} = (y_1, y_2, \ldots, y_m)^T$, pode-se assumir que os dados são independentes e identicamente distribuídos, o que implica que a probabilidade da classes $\mathbf{y}$ serem previstas para os elementos $\mathbf{X}$, considerando os parâmetros $\boldsymbol{\theta}$ conhecidos, pode ser calculada como o produto das probabilidades de cada elemento:

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = \prod_{i=1}^{m} p(y_i|\boldsymbol{x}_i, \boldsymbol{\theta})) \tag{2.14}$$

Esta abordagem é muito semelhante ao uso de uma função de perda, como visto anteriormente. Por exemplo, considerando que a variância é um parâmetro constante, maximizar a expressão 2.13 implica em minimizar o termo $(y_i - f(\boldsymbol{x}_i, \boldsymbol{\theta}))^2$ (lembrando do sinal negativo na exponencial). Isto é equivalente a utilizar uma função de perda do tipo mínimos quadrados, $\ell(\hat{y}_i, y_i) = (y_i - \hat{y}_i)^2$. Por isso, assim como no caso da minimização do risco empírico, esta abordagem pode levar a sobreajuste.

Este problema pode ser reduzido adicionando um termo na função objetivo, multiplicando a verossimilhança, que leva em consideração informações adicionais sobre a distribuição dos parâmetros $\boldsymbol{\theta}$ (informação *a priori*). Este termo irá ter um papel semelhante à regularização aplicada para minimização do risco empírico. Utilizando este termo com conjunto com o Teorema de Bayes, obtém-se uma abordagem chamada de *probabilidade máxima a posteriori* (MAP).

Neste material não serão apresentados detalhes sobre a aplicação deste procedimento para a estimação de parâmetros, porém, na Seção 2.5 aplicações do Teorema de Bayes serão avaliada para a construção de algoritmos baseados na abordagem Naive Bayes. Por enquanto, é importante saber que a obtenção dos parâmetros de um modelo pode ser feita através da minimização do risco empírico ou através da maximização da verossimilhança, sendo que ambas abordagens podem levar a sobreajuste (ou subajuste).

Nas próximas seções serão apresentados detalhes sobre a elaboração e aplicação de diferentes algoritmos de classificação. Estes algoritmos também podem ser aplicados para problemas de classificação, como será discutido no próximo capítulo, porém, no momento a discussão será limitada a problemas de classificação. Em particular, serão abordados três tipos de algoritmos (k-NN, máquinas de vetores de suporte e Naive Bayes), sendo estes escolhidos pois cada um possui uma abordagem distinta. Diversos outros algoritmos podem ser aplicados para este fim (por exemplo, árvores de decisão, regressão logística, random forest, classificadores baseados em redes neurais, etc.), porém, tendo um entendimento destas três categorias básicas é possível resolver grande parte dos problemas práticos de classificação.

2.3 k-Vizinhos mais Próximos (k-NN)

Os algoritmos baseados em Vizinhos Próximos são uma das classes de algoritmos mais simples utilizados para classificação. O princípio básico de funcionamento destes algoritmos é memorizar um conjunto de dados de treinamento e prever a classe de um novo ponto com base na classe dos vizinhos mais próximos deste ponto, assumindo que pontos próximos tem uma maior chance de pertencer à mesma categoria. Com isso, durante o treinamento não é necessário ajustar parâmetros a um modelo.

A distância utilizada na maioria dos algoritmos baseados em k-NN é a própria distância euclidiana entre os elementos. Por exemplo, considere que seja utilizada uma função $f(\mathbf{x}, \mathbf{x}')$ para determinar a distância entre dois elementos $\mathbf{x}$ e $\mathbf{x}'$, de modo que[10]:

$$f(\mathbf{x}, \mathbf{x}') = ||\mathbf{x}' - \mathbf{x}|| \tag{2.15}$$

Assumindo que $\mathbf{x}$ seja um vetor com n pontos, ou seja, $\mathbf{x} = (x_1, x_2, x_3, \ldots, x_n)$, a norma é dada por:

$$||\mathbf{x}' - \mathbf{x}|| = \sqrt{\sum_{i=1}^{n} (x_i - x_i')^2} \tag{2.16}$$

Como o objetivo é fazer a classificação com base na classe dos vizinhos mais próximos, esta distância pode ser interpretada como uma função de perda que deve ser minimizada.

Considere que seja utilizado um conjuto de treinamento T contendo m elementos, ou seja, m vetores $\mathbf{x}$ e suas respectivas classes y:

$$T = (\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots (\mathbf{x_m}, y_m) \tag{2.17}$$

[10]Neste material, será considerado a norma ℓ_2 como padrão.

Para determinar a classe de um novo elemento $\mathbf{x}'$, a primeira etapa é determinar a sua distância com relação aos elementos do grupo de treinamento. Após, os elementos devem ser reordenados de acordo com a distância em um novo grupo, por exemplo T':

$$T' = (\mathbf{w_1}, y_{w1}), (\mathbf{w_2}, y_{w2}), \ldots (\mathbf{w_m}, y_{wm}) \tag{2.18}$$

onde $\mathbf{w_i}$ corresponde a um dos elementos $\mathbf{x_i}$ e y_{wi} é a classe associada a este elemento. Os elementos do grupo T' são ordenados de forma que:

$$f(\mathbf{w_i}, \mathbf{x}') \leq f(\mathbf{w_{i+1}}, \mathbf{x}') \tag{2.19}$$

ou seja, o elemento $\mathbf{w_i}$ está mais próximo de $\mathbf{x}'$ do que o elemento $\mathbf{w_{i+1}}$. Com isso, pode-se interpretar que o conjunto T' possui os elementos do conjunto T ordenados em uma ordem crescente de acordo com o valor da função de perda associada com cada elemento.

O exemplo mais simples é quando somente um vizinho é utilizado para a classificação ($k = 1$). Neste caso, a classe prevista para o elemento $\mathbf{x}'$, representada por $y_{x'}$, será a classe do vizinho mais próximo:

$$y_{x'} = y_{w1} \tag{2.20}$$

Na Figura 2.8 é apresentado uma ilustração das barreiras de divisão binária utilizando o algoritmo 1-NN.

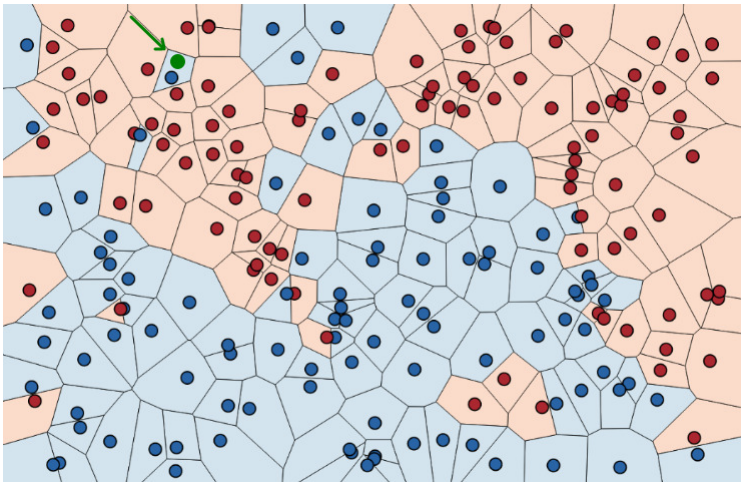


Figure 2.8: Exemplo de classificação usando 1-NN.

O principal problema associado a utilização de um valor de k pequeno é a presença de ruído nos dados de treinamento, ou seja, pontos que apresentam um desvio significativo. Por exemplo, considere que se busque determinar a classe associada ao ponto verde na Figura 2.8 (indicado com um seta para facilitar sua localização). Caso seja utilizado somente um

vizinho, este será rotulado com a classe azul. Porém, este ponto azul parece corresponder a algum erro de medição, já que está cercado por pontos vermelhos. Uma maneira de resolver este problema é utilizar mais vizinhos, por exemplo, se $k = 3$ a maioria dos pontos vizinhos será vermelho e a classificação será provavelmente mais adequada.

O peso dado para cada vizinho pode ser o mesmo ou uma função específica pode ser utilizada para, por exemplo, diminuir a influência de um ponto quanto mais distante estiver do ponto avaliado. O problema em utilizado muitos pontos ocorre quando existe uma classe dominante no grupo de treinamento. Por exemplo, considere que 90% dos dados correspondam a uma classe 1 e 10% a outra classe 2. Se um valor de k alto for utilizado, existe uma grande chance de qualquer ponto ser classificado com pertencente à classe 1, já que ela é dominante. Neste caso, a utilização de uma função peso é importante.

Uma variação do k-NN são os algoritmos baseados em um *raio de vizinhança*. A diferença para o k-NN é que neste caso, ao invés de fixar um número k de vizinhos, é fixado um raio em torno do ponto e são analisados todos os vizinhos dentro deste raio. Este tipo de algoritmo costuma ser mais adequado quando os espaçamento entre os dados é muito não-uniforme.

2.4 Máquinas de Vetores de Suporte (SVM)

O objetivo dos algoritmos de classificação é definir *fronteiras de decisão*, que permitam separar as diferentes classes. No caso do k-NN, esta fronteira é definida supondo similaridade entre elementos próximos. Outra maneira de definir esta separação é criar diretamente um *plano de separação* entre as classes, ou seja, uma região que limita a extensão de cada classe. Os algoritmos de separação podem ser aplicados em espaços com uma dimensão N qualquer, sendo o plano de separação um *hiperplano* neste espaço (plano com dimensão $N-1$), onde N será equivalente ao número de atributos de cada elemento. Por simplificação, neste material o termo "plano de separação" será utilizado para se referir a este hiperplano em todos os casos, incluindo o caso 2D onde o hiperplano será simplesmente uma reta.

Considere, por exemplo, os dados apresentados na Figura 2.9. Estes dados estão relativamente bem separados, de forma que é fácil imaginar retas que separem as duas classes, como ilustrado na figura. Porém, fica claro que as três fronteiras propostas, apesar de todas separarem os dados com 100% de precisão, trazem informações distintas. Por exemplo, a linha 1 (vermelha) está muito mais próxima dos pontos roxos, tendendo a subestimar a influência desta classe. De forma semelhante, a linha 2 (verde) está muito mais próxima dos pontos

amarelos. A linha 3 (azul) também não é uma separação ótima, pois está mais próxima a uma das classes em regiões distintas.

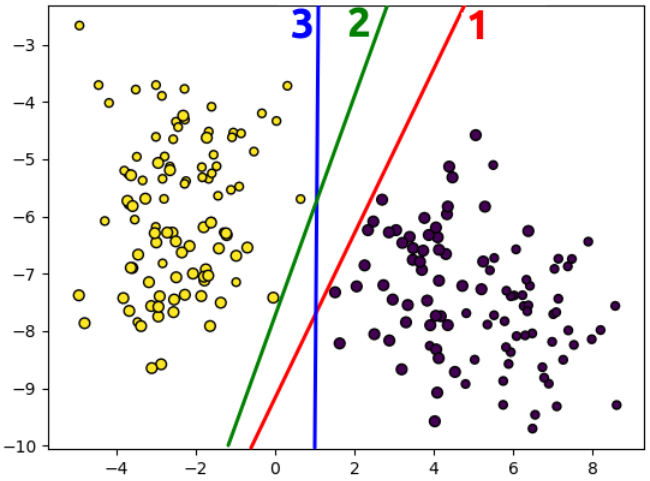


Figure 2.9: Possíveis planos de separação para o conjunto de dados.

Através da utilização dos algoritmos baseados em máquinas de vetores de suporte (*Support Vector Machines - SVM*), busca-se qual é o plano de separação entre os grupos que esteja o mais distante possível dos elementos de cada grupo, de forma a se obter uma separação que está "no meio do caminho" entre as duas classes.

A princípio, estes algoritmos são classificadores binários lineares, ou seja, separam dados de duas classes utilizado reta, ou em um espaço vetorial com dimensão N, hiperplanos com dimensão $N-1$, como discutido anteriormente. Porém, pode-se também aplicar estes algoritmos para classificação multiclasse, utilizando intersecções entre classificadores binários, e em problemas com fronteiras de decisão não-lineares, através da utilização de funções núcleo (*kernel*) específicas, como será comentado posteriormente.

Por enquanto, considere um caso de classificação binária com fronteira de decisão linear, como ilustrado na figura anterior. O objetivo é encontrar um plano de separação (neste exemplo uma reta, já que os dados formam um espaço 2D), como ilustrado na linha b representada na Figura 2.10. Esta reta possui a propriedade de estar equidistante dos elementos mais próximos de cada uma das classes. As linhas tracejadas a e c, ambas paralelas a b, representam os planos onde o elemento mais próximo de uma das classes é encontrado. Observe que neste caso existe uma *margem de separação* que representa a distância entre a e b ou entre b e c, ou ainda, de forma equivalente, a distância entre os pontos mais próximos em cada uma das classes e o plano de separação.

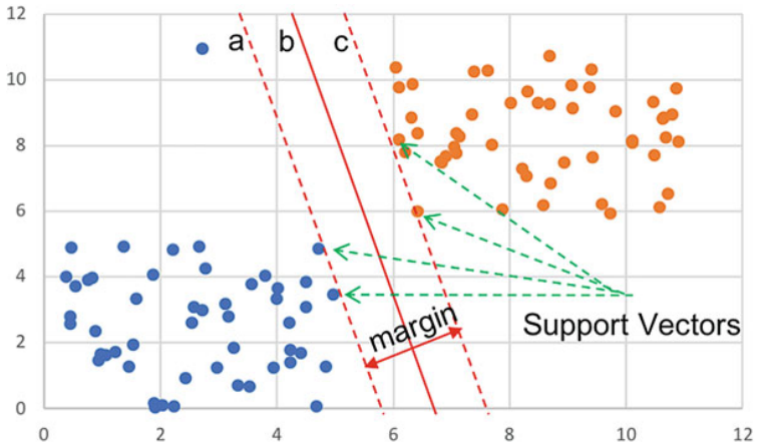


Figure 2.10: Representação do plano de separação usando SVM. Fonte:

De forma ideal, o plano de separação divide o espaço em regiões ocupadas somente por elementos de uma mesma classe. No caso da classificação binária, é comum chamar estas duas regiões de *positiva* e *negativa*. Para definir se um ponto está na parte positiva ou na parte negativa, será utilizado um vetor $\mathbf{w}$ normal ao plano de separação e com comprimento arbitrário, como ilustrado na Figura 2.11.

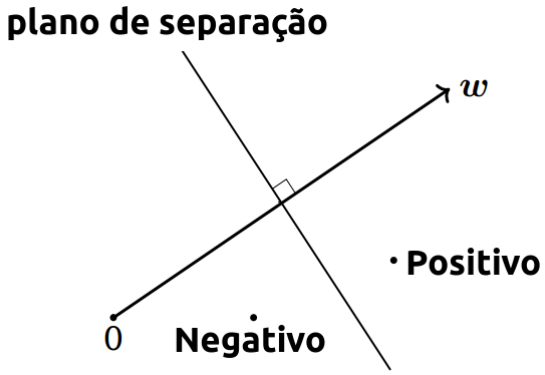


Figure 2.11: Projeção do vetor normal ao plano de separação.

Considere um grupo de dados de treinamento da forma

$$T = (\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots (\mathbf{x_m}, y_m) \tag{2.21}$$

onde, neste caso, as classe y_i são divididas em dois grupo, +1 e -1 (positivas e negativas) e $\mathbf{x_i}$ são vetores contendo 2 elementos. Uma maneira de determinar se um ponto $\mathbf{x_i}$ está na parte positiva ou na parte negativa dividida pelo plano de separação é avaliar o produto escalar deste vetor com o vetor $\mathbf{w}$. Lembrando que, do conceito de álgebra linear, o produto escalar

entre os vetores é definido como:

$$\mathbf{w} \cdot \mathbf{x} = ||\mathbf{w}|| \, ||\mathbf{x}|| \cos(\theta)$$

onde θ é o ângulo formado entre os vetores. Este operador retorna um *escalar* que representa o produto da norma do vetor $\mathbf{w}$ com a projeção do vetor $\mathbf{x}$ sobre $\mathbf{w}$, ou seja, representa uma *distância* na direção do vetor $\mathbf{w}$. Se esta distância for suficientemente grande, o ponto estará na parte positiva, senão estará na parte positiva. Por exemplo, pode-se definir que para que um ponto esteja na parte positiva, temos que:

$$\mathbf{w} \cdot \mathbf{x} \geq C \tag{2.22}$$

onde C é uma constante (desconhecida já que $\mathbf{w}$ ainda não é conhecido). De forma equivalente, para um ponto na região negativa, o produto escalar deve ser menor que C. Estas relações também pode ser escritas da seguinte forma, usando uma constante arbitrária no lado esquerdo:

- Se $\mathbf{w} \cdot \mathbf{x_i} + b \geq 0$, então $y_i = 1$;
- Se $\mathbf{w} \cdot \mathbf{x_i} + b < 0$, então $y_i = -1$;

Estas relações servem como um critério de decisão para saber se um dado ponto está na parte positiva ou negativa. Para aplicar este critério de decisão, é preciso definir valores para b e $\mathbf{w}$, que até este ponto não são conhecidos. Uma maneira mais compacta de representar as duas condições acima é multiplicar o termo $\mathbf{w} \cdot \mathbf{x_i} + b$ por y_i. Como no primeiro caso este termo é positivo e no segundo negativo, as duas relações podem ser escritas de maneira equivalente como:

$$y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 0 \tag{2.23}$$

sendo que esta relação é válida para os dois casos.

2.4.1 Obtenção da Margem Rígida

Considere um ponto qualquer $\mathbf{x_a}$ posicionado, por exemplo, no lado positivo do plano de separação, como ilustrado na Figura 2.12. A distância deste ponto até o plano de separação é indicada como r. Para determinar a margem (distância entre o plano de separação e o ponto mais próximo), é necessário determinar o valor de r para todos os pontos.

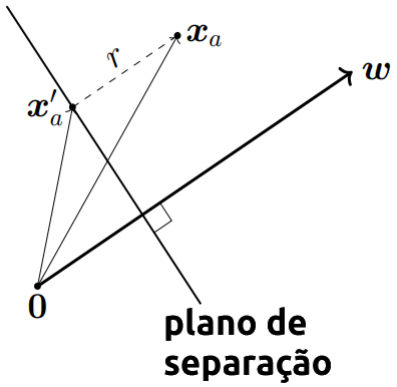


Figure 2.12: Projeção de um ponto $\mathbf{x_a}$ sobre o plano de separação.

Para isso, podemos utilizar a projeção ortogonal de $\mathbf{x_a}$ sobre o plano de separação, indicado na figura como $\mathbf{x'_a}$. Como o vetor $\mathbf{w}$ também é ortogonal ao plano de separação, sabemos que o vetor formado por $\mathbf{x_a} - \mathbf{x'_a}$ será paralelo ao vetor $\mathbf{w}$. Isto implica que pode-se somar o vetor $\mathbf{x'_a}$ com o vetor $\mathbf{w}$ multiplicado por alguma constante para se obter $\mathbf{x_a}$.

Porém, como a magnitude de $\mathbf{w}$ não é conhecida, esta constante não pode ser determinada. Por conveniência, pode-se transformar o vetor $\mathbf{w}$ em um vetor unitário (normalizado) dividindo os elementos pela norma $(\mathbf{w}/||\mathbf{w}||)$ [11]. Com isso, quando somado ao vetor $\mathbf{x'_a}$, a constante que deve multiplicar $\mathbf{w}$ será a própria distância r, ou seja:

$$x_a = x'_a + r\frac{\mathbf{w}}{||\mathbf{w}||} \tag{2.24}$$

De forma geral, pode-se entender que $\mathbf{w}/||\mathbf{w}||$ define a direção em que se deve seguir e r o quanto seguir para sair do ponto $\mathbf{x'_a}$ e chegar no ponto $\mathbf{x_a}$.

Para obter a margem associada com os pontos, deve-se encontrar o maior valor de r possível tal que todos os pontos, tanto no lado positivo quanto no negativo, estejam a uma distância de pelo menos r do plano de separação. Lembrando ainda a regra de decisão estabelecida na Eq. 2.23, que definia a distância até o plano de separação, temos que para todos os pontos a condição

$$y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq r \tag{2.25}$$

deve ser satisfeita. Esta relação impõe que qualquer elemento, tanto na região positiva quanto na negativa, deve estar a uma distância de pelo menos r do plano de separação.

O objetivo do algoritmo é encontrar o maior valor de r possível tal que estas condições sejam satisfeitas. Isto gera um problema de otimização, onde deve-se encontrar $\mathbf{w}$ e b que

[11] Vetores unitários são vetores com comprimento igual a 1

maximizem r. Para obter este valor, é conveniente definir uma *escala* para os dados, de forma que $\mathbf{w} \cdot \mathbf{x} + b = 1$ para o ponto mais próximo ao plano de separação. Assumindo que este ponto mais próximo seja denotado $\mathbf{x_a}$, esta representação em escala pode ser vista na Figura 2.13.

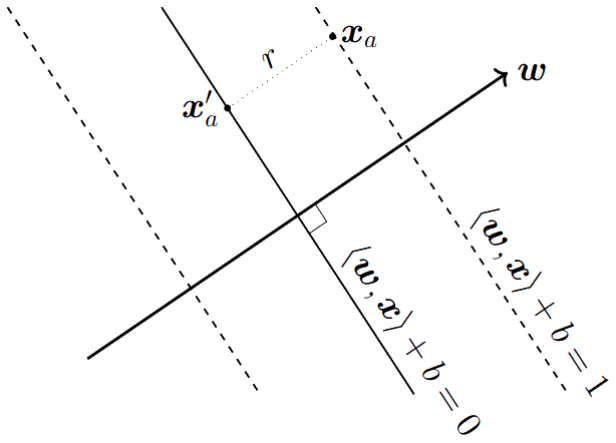


Figure 2.13: Representação em escala de $\mathbf{x_a}$ como o ponto mais próximo ao plano de separação.

Como a projeção ortogonal de $\mathbf{x_a}$, denotada por $\mathbf{x'_a}$ está no plano de separação, como ilustrado na figura temos que:

$$\mathbf{w} \cdot \mathbf{x'_a} + b = 0 \tag{2.26}$$

Isolando $\mathbf{x'_a}$ na equação 2.24 e substituindo na expressão anterior:

$$\mathbf{w} \cdot \left(\mathbf{x_a} - r\frac{\mathbf{w}}{||\mathbf{w}||} \right) + b = 0 \tag{2.27}$$

Como o produto escalar é um operador bilinear[12], a expressão pode ainda ser escrita como:

$$(\mathbf{w} \cdot \mathbf{x_a} + b) - r\frac{\mathbf{w} \cdot \mathbf{w}}{||\mathbf{w}||} = 0 \tag{2.28}$$

O termo entre parênteses será, por definição, igual a 1, como indicado anteriormente. Como o ângulo formado pelo vetor $\mathbf{w}$ com ele mesmo é zero, temos que $\mathbf{w} \cdot \mathbf{w} = ||\mathbf{w}||^2$. Substituindo estas expressões na relação anterior e simplificando, chega-se a conclusão que:

$$r = \frac{1}{||\mathbf{w}||} \tag{2.29}$$

Considerando que o ponto $\mathbf{x_a}$ seja o mais próximo ao plano de separação, o critério de decisão apresentado anteriormente (Eq. 2.23) pode agora ser reescrito como:

$$y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1 \qquad\qquad i = 1, 2, 3, \ldots m. \tag{2.30}$$

[12]Isto implica que $\mathbf{a} \cdot (r\mathbf{b} + \mathbf{c}) = r(\mathbf{a} \cdot \mathbf{b}) + \mathbf{a} \cdot \mathbf{c}$

Juntando estas informações, o problema de otimização pode ser expresso como:

Obtenção da margem: Dado um conjunto $T = (\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots (\mathbf{x_m}, y_m)$, deve-se encontrar $\mathbf{w}$ e b tal que a variável $r = 1/||\mathbf{w}||$ seja maximizada considerando as restrições $y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1$ para $i = 1, 2, 3, \ldots, m$.

Este problema pode ser escrito de forma mais compacta como:

$$\max_{\mathbf{w},b} \frac{1}{||\mathbf{w}||}, \qquad \text{sujeito à } y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1, \ i = 1, 2, 3, \ldots, m. \tag{2.31}$$

Maximizar a variável $r = 1/||\mathbf{w}||$ é equivalente a minimizar $||\mathbf{w}||$, lembrando que este valor será sempre positivo. Na maioria dos casos, a condição de otimização é apresentada como sendo minimizar $||\mathbf{w}||^2/2$, pois isto apresenta vantagens do ponto de vista da aplicação dos algoritmos de otimização (a função irá gerar um mínimo global), então a relação anterior pode também ser expressa como:

Forma equivalente para obtenção da margem: Dado um conjunto $T = (\mathbf{x_1}, y_1)$, $(\mathbf{x_2}, y_2), \ldots (\mathbf{x_m}, y_m)$, deve-se encontrar $\mathbf{w}$ e b tal que $||\mathbf{w}||^2/2$ seja minimizada considerando as restrições $y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1$ para $i = 1, 2, 3, \ldots, m$.

Ou ainda:

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2, \qquad \text{sujeito à } y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1, \ i = 1, 2, 3, \ldots, m. \tag{2.32}$$

Neste material não será abordada a resolução deste problema de otimização. Isto normalmente é feito através da introdução de uma função de Lagrange que engloba as restrições (veja Deisenroth et al. (2020)) para mais detalhes).

Neste caso, os valores de $\mathbf{w}$ e b são obtidos como uma função do conjunto de pontos de treinamento $\mathbf{x}$, por isso, estes vetores são chamados de *vetores de suporte*. Nesta formulação, a margem é totalmente definida pelos elementos mais próximos ao plano de separação, não sendo permitido que nenhum elemento esteja no lado errado do plano (um elemento positivo no lado negativo ou vice-versa). Devido à esta restrição, esta abordagem é chamada de ***SVM com margem rígida***.

Apesar de ser um algoritmo muito eficiente quando os pontos estão linearmente separados, a utilização de uma margem rígida gera problemas quando existem *outliers* nos dados. Considere, por exemplo, os cenários apresentados na Figura 2.14. No primeiro caso, como o outlier está cercado por pontos de outra classe, é impossível encontrar um plano de separação

que satisfaça o critério definido pelo problema de otimização apresentado anteriormente. No segundo caso, este plano até pode ser encontrado, porém, o resultado claramente apresenta uma forte tendência de favorecer uma das classes devido à presença de um outlier. Para evitar estes problemas, deve-se permitir algum grau de flexibilização no algoritmo, de modo que seja possível que alguns pontos *violem* o critério de separação. Com o objetivo de encontrar uma margem o maior possível limitando a quantidade de pontos que violam a separação, utiliza-se a abordagem de margem suave, como será discutido a seguir.

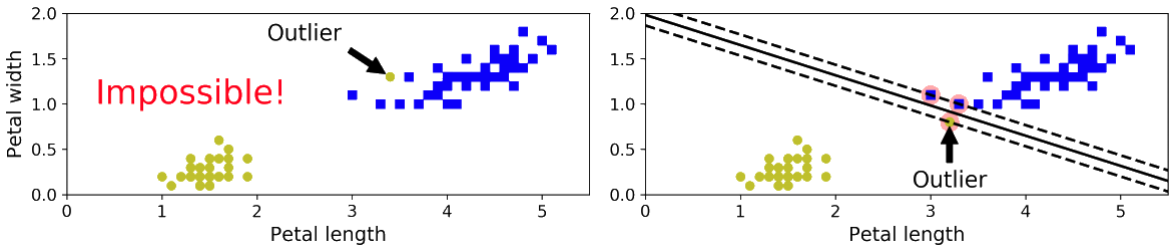


Figure 2.14: Representação de casos onde a utilização de margem rígida não é adequada. Fonte: Géron (2019)

2.4.2 Obtenção da Margem Suave

A abordagem com margem rígida funciona muito bem quando os dados estão linearmente separados, porém, caso o plano de separação seja não-linear ou os dados de treinamento possuam ruídos, é conveniente permitir que alguns elementos estejam dentro da região definida pela margem ou mesmo no lado errado do plano de separação. Para este procedimento, pode-se utilizar uma abordagem de ***SVM com margem suave***. Esta formulação pode ser obtida por duas formas diferentes, como será discutido a seguir.

Utilização de uma Variável de Folga

Umas das formas de se obter uma SVM com margem suave é introduzir uma *variável de folga* ξ_i (*slack variables*, muitas vezes também traduzidas como variáveis de relaxamento), para cada par $(\mathbf{x_i}, y_i)$. Quando o elemento estiver no lado correto da região definida pelo plano de separação e fora da margem, é atribuído o valor zero para ξ_i.

Porém, caso a variável esteja no lado errado (ou seja, classificada de forma errada) ou dentro da margem, a variável ξ_i irá receber o valor da *distância* do ponto até a marge do

lado correto. Assim, para pontos positivos ($y_i = 1$) ξ_i será a distância até a margem do lado positivo e para pontos negativos ($y_i = -1$) será a distância até a margem do lado negativo. Esta definição é representada na Figura 2.15 [13].

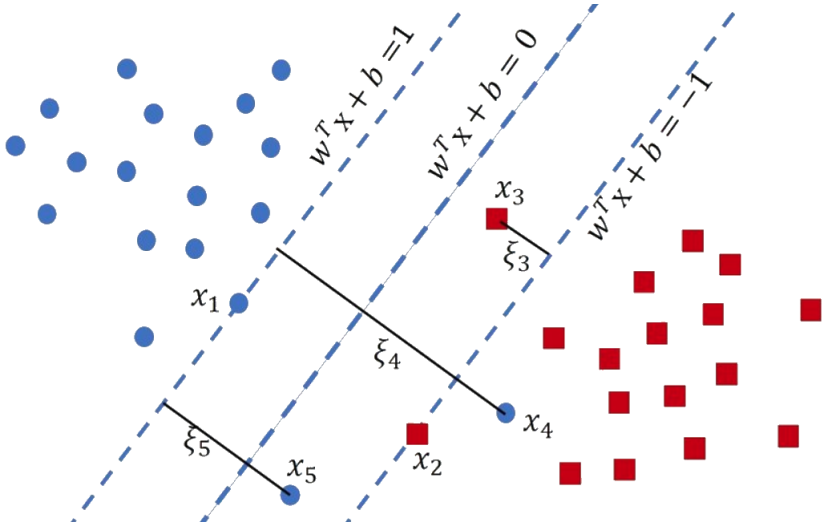


Figure 2.15: Representação da determinação das variáveis de folga. Fonte: Le et al. (2018)

Considerando a presença da variáveis de folga, o problema de otimização apresentado na Eq. 2.32 é reformulado. Agora, ao invés de simplesmente minimizar $||\mathbf{w}||$ (o que equivale a maximizar r), será buscado minimizar a seguinte relação:

$$\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i$$

onde C é uma constante que deve ser informada pelo usuário, sendo chamada de *parâmetro de regularização*, de forma análoga ao processo de regularização discutido na Seção 2.2.3. Se um valor pequeno de C for utilizado, as variáveis de folga terão pouca influência no resultado da otimização, e como resultado é comum obter uma separação com maior margem (e consequentemente mais elementos dentro da marge ou classificados de forma errada). Se um valor de C alto for utilizado, obtém-se uma menor margem, como ilustrado na Figura 2.16.

A princípio, pode parecer que uma margem mais estreita é vantajosa, porém nem sempre este é o caso. Quanto menor a margem, menor será a capacidade de generalização do algoritmo, o que pode levar a sobreajuste. De fato, comparando os resultados apresentados na Figura 2.16, pode-se ver que o desempenho dos dois casos é muito similar.

[13] Nesta figura, o termo $\mathbf{w} \cdot \mathbf{x}$ é apresentado como $\mathbf{w}^T\mathbf{x}$, o que é equivalente considerando que os dois termos são vetores com mesma dimensão.

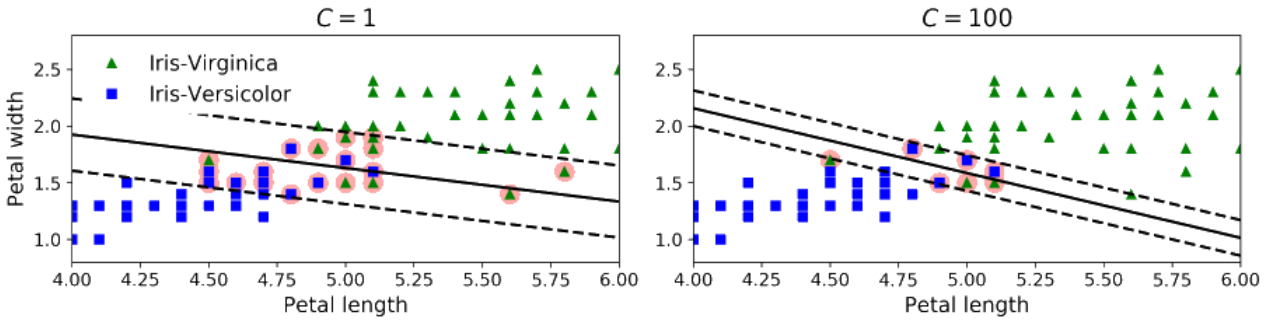


Figure 2.16: Margens obtidas com $C = 1$ e $C = 100$. Fonte: Géron (2019)

Além de alterar o parâmetro que deve ser minimizado, é preciso rever a restrição $y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1$, pois se ela for mantida, nenhum elemento poderá estar dentro da margem. Para permitir que isto ocorra, pode-se subtrair o termo ξ_i do lado direito, assim, os casos que estão dentro da margem ou no lado errado da classificação não impedem a otimização. Com isso, o problema de otimização é retomado como:

Obtenção da margem suave usado variáveis de folga: Dado um conjunto $T = (\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots (\mathbf{x_m}, y_m)$, deve-se encontrar $\mathbf{w}$, b e ξ_i tal que $||\mathbf{w}||^2/2 + C\sum_{i=1}^{m} \xi_i$ seja minimizado considerando as restrições $y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1 - \xi_i$ para $i = 1, 2, 3, \ldots, m$ e $\xi_i \geq 0$.

Ou ainda:

$$\min_{\mathbf{w},b,\xi_i} \left(\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m} \xi_i \right), \quad \text{sujeito à } y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1 - \xi_i, \; i = 1, 2, \ldots, m., \; \xi_i \geq 0 \quad (2.33)$$

Este tipo de formulação, utilizando um parâmetro de regularização C, é muitas vezes chamada de *C-SVM*. Observe que, caso $\xi_i = 0$ para todos os elementos, o problema retorna a sua formulação com margem rígida (Eq. 2.32). Outra maneira de se obter uma formulação com marge suave é utilizar uma *função de perda* (*loss function*) no lugar de uma variável de folga, como será visto a seguir.

Utilização de uma Função de Perda

Em otimização, uma função de perda (ou função custo) é uma função utilizada para mensurar algum "custo" associado com uma série de eventos, como discutido na seção 2.2. No contexto de algoritmos de classificação, esta é uma função utilizada para penalizar o algoritmo por alguma classificação incorreta. Como visto anteriormente, o critério ideal

para todos os pontos é que (Eq. 2.30):

$$y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1 \qquad\qquad i = 1, 2, 3, \ldots m$$

Isto implica que todos os pontos estão classificados corretamente e fora da margem. Esta relação pode ser utilizada para a obtenção de uma função de perda, já que quanto menor estiver este valor, pior será a classificação. A função de perda mais utilizada neste caso é a função *perda de articulação* (*hinge loss*), definida como:

$$\ell(t) = \max(0, 1 - t) = \begin{cases} 0 & \text{se } \; t \geq 1 \\ 1 - t & \text{se } \; t < 1 \end{cases} \tag{2.34}$$

onde $t = y_i(\mathbf{w} \cdot \mathbf{x_i} + b)$. Assim, se o critério anterior for satisfeito (Eq. 2.30 ou de forma equivalente $t \geq 1$), a função objetivo não será penalizada. Se o critério não for satisfeito, a função é penalizada de maneira linearmente proporcional ao desvio. Utilizando esta função de perda, o problema para otimizar a margem pode ser escrito como:

Obtenção da margem suave usado função perda de articulação: Dado um conjunto $T = (\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots (\mathbf{x_m}, y_m)$, deve-se encontrar $\mathbf{w}$ e b tal que $||\mathbf{w}||^2/2$ $+C\sum_{i=1}^{m} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x_i} + b))$ seja minimizado.

Ou ainda:

$$\min_{\mathbf{w},b} \left(\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x_i} + b)) \right) \tag{2.35}$$

Veja que neste caso o problema de otimização não exige uma restrição quanto ao valor $y_i(\mathbf{w} \cdot \mathbf{x_i} + b))$, já que a própria função objetivo irá buscar o melhor valor para este termo.

A classificação utilizando SVM com marge suave funciona muito bem quando a separação dos dados é aproximadamente linear, porém para fronteiras não-lineares o algoritmo neste formato não irá desempenhar um bom resultado. Para estes casos, pode-se realizar algumas mudanças na forma como os dados são representados, como será discutido a seguir.

Um exemplo de aplicação de máquinas de suporte lineares para classificação binária é apresentado neste vídeo[14].

[14] https://www.youtube.com/watch?v=pF2fLlUnqE4

2.4.3 SVM Não-Lineares

Quando não existe uma separação linear entre as classes, pode-se utilizar uma estratégia de adicionar um novo atributo ao conjunto de dados de forma a torná-los linearmente separáveis em um espaço com maior dimensão. Por exemplo, considere o conjunto de dados com dois atributos apresentados na Figura 2.17.

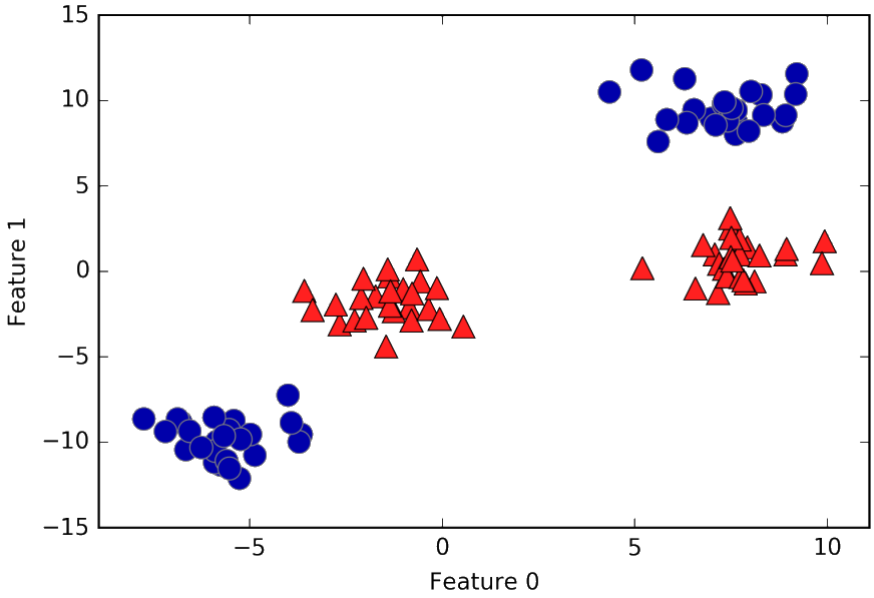


Figure 2.17: Conjunto de dados não-linearmente separáveis. Fonte: Muller and Guido (2017)

Analisando os dados, fica claro que não é possível utilizar uma fronteira de decisão linear para separar os dados. Uma maneira de buscar uma separação entre os dados é atribuir um terceiro valor para cada ponto, passando o conjunto para um espaço 3D. Por exemplo, considere que este novo valor seja o valor do parâmetro indicado como *Feature 0* (eixo x do gráfico) elevado ao quadrado. Isto fará com que os pontos onde o parâmetro *Feature 0* tem um valor diferente de zero se afastem da origem, neste novo eixo z criado, proporcionalmente ao valor da *Feature 0* ^{2}. O resultado pode ser visto na Figura 2.18, juntamente com o plano de separação obtido. Com isso, é possível separar as classes neste novo espaço criado.

Para entender como a obtenção de parâmetros em uma dimensão maior pode ser feita, é interessante analisar o problema de otimização definido na Equação 2.33 após a aplicação do Lagrangiano. A obtenção detalhada desta equação pode ser encontrada em Deisenroth et al. (2020):

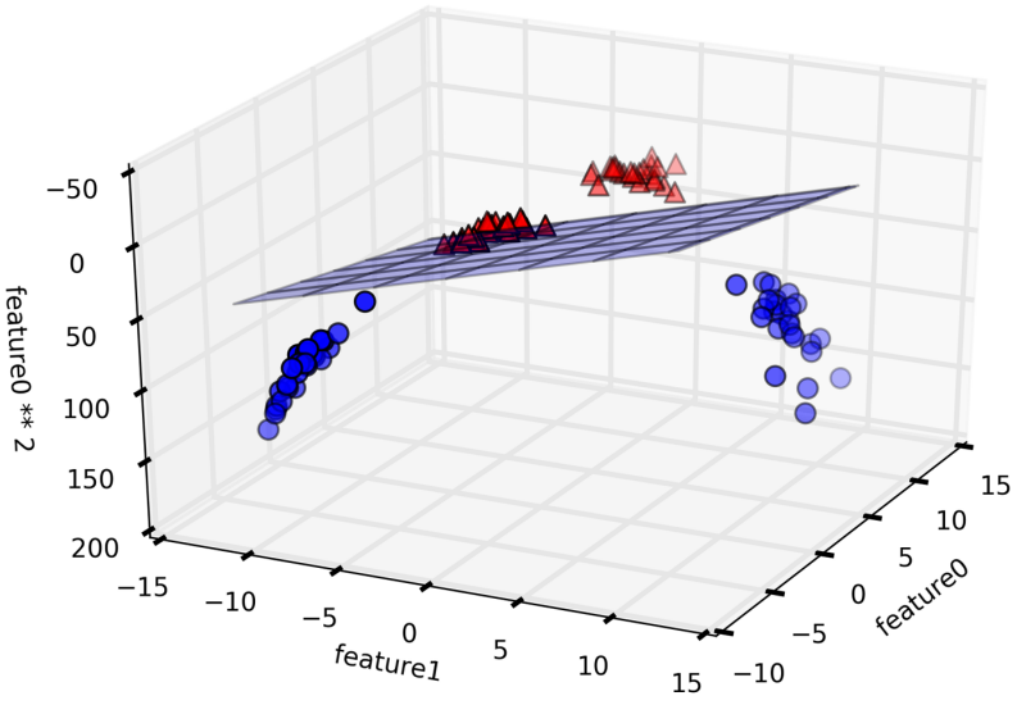


Figure 2.18: Plano de separação considerando um terceiro parâmetro no conjunto de dados. Fonte: Muller and Guido (2017)

$$\min_{\alpha} \left(\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j (\mathbf{x_i} \cdot \mathbf{x_j}) - \sum_{i=1}^{m} \alpha_i \right) \tag{2.36}$$

$$\text{sujeito à} \quad \sum_{i=1}^{m} y_i \alpha_i = 0 \qquad \text{onde} \qquad 0 \leq \alpha_i \leq C \quad i = 1, 2, 3, \ldots m$$

Nesta expressão, os termos α_i representam os multiplicadores de Lagrange utilizados para incluir as restrições na função objetivo (por enquanto, é suficiente saber que estes valores são escalares). O interessante desta função objetivo é que os pontos $\mathbf{x_i}$ não aparecem de forma explícita na função objetivo, sendo que esta depende somente do produto escalar entre os pontos.

Como discutido anteriormente, uma maneira de tornar o problema linearmente separável é adicionar mais atributos para cada vetor $\mathbf{x}$, aumentando o número de elementos deste vetor. De forma geral, isto pode ser representado como um operador $\phi(\mathbf{x})$ que irá *mapear* o vetor $\mathbf{x}$ em um espaço de maior dimensão. No exemplo anterior, onde foi adicionado um terceiro atributo equivalente a outro elevado ao quadrado, está função $\phi(\mathbf{x})$ mapeia um espaço 2D

em um espaço 3D ($\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$) da forma:

$$\phi(\mathbf{x}) = \phi(x_1, x_2) = (x_1, x_2, x_1^2) \tag{2.37}$$

Com isso, o problema de otimização anterior pode ser re-escrito considerando este mapa $\phi(\mathbf{x})$ da seguinte forma:

$$\min_{\alpha} \left(\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j (\phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_j})) - \sum_{i=1}^{m} \alpha_i \right) \tag{2.38}$$

$$\text{sujeito à} \qquad \sum_{i=1}^{m} y_i \alpha_i = 0 \qquad \text{onde} \qquad 0 \leq \alpha_i \leq C \quad i = 1, 2, 3, \ldots m$$

Isto equivale a aplicar o algoritmo de SVM com margem suave em um espaço com dimensão maior que o original, definido pelo operador $\phi(\mathbf{x})$.

Para problemas simples pode-se definir esta função $\phi(\mathbf{x})$, mapear os elementos neste novo espaço e calcular os produtos escalares resultantes. Porém, além de ser computacionalmente custoso, na prática é muito difícil determinar qual função irá tornar o problema linearmente separável. Para auxiliar nesta tarefa, é utilizado uma estratégia chamada de *truque do kernel*, que consiste em utilizar uma função *kernel* para mapear os pontos em um espaço de maior dimensão. De forma geral, uma função kernel $\mathbf{K}$ é uma função aplicada em um par de vetores $\mathbf{x}$ pertencentes a um conjunto $\mathcal{X}$ e que retorna um valor real, ou seja, $\mathbf{K} : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.

Considerando que esta função seja contínua, simétrica e não-negativa (*teorema de Mercer*), pode-se definir uma função kernel de forma que

$$k(\mathbf{x_i}, \mathbf{x_j}) = \phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_j}) \tag{2.39}$$

Ou seja, ao invés de definir explicitamente um mapa não-linear $\phi(\mathbf{x})$, calcular os novos elementos e seus produtos escalares, é definida uma função kernel que define de forma implícita o produto escalar entre dois elementos. Com isso, não é necessário determinar a função $\phi(\mathbf{x})$ ou mesmo o novo espaço onde os dados serão definidos. De fato, pode-se até mesmo projetar os dados em um *espaço com dimensão infinita* para obter uma separação linear.

Com base na definição da função kernel, o problema de otimização para SVM não-linear pode finalmente se escrito como:

$$\min_{\alpha} \left(\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j k(\mathbf{x_i}, \mathbf{x_j}) - \sum_{i=1}^{m} \alpha_i \right) \tag{2.40}$$

$$\text{sujeito à} \qquad \sum_{i=1}^{m} y_i \alpha_i = 0 \qquad \text{onde} \qquad 0 \leq \alpha_i \leq C \quad i = 1, 2, 3, \ldots m$$

A grande vantagem da utilização de kernels é que a estrutura do algoritmo praticamente não é alterada, sendo somente necessário calcular esta função para cada conjunto de pontos, o que normalmente é muito mais rápido do que definir a função $\phi(\mathbf{x})$ e calcular os produtos escalares.

O kernel mais simples que pode ser utilizado é o próprio *kernel linear*:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i} \cdot \mathbf{x_j} \tag{2.41}$$

que irá retornar a própria formulação das SVM lineares.

Existem diversos kernels propostos na literatura, entre os mais utilizados pode-se citar os seguintes:

- Kernel polinomial: definido de forma geral como:

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\gamma(\mathbf{x_i} \cdot \mathbf{x_j}) + r)^d \tag{2.42}$$

  onde γ, r e d são parâmetros definidos pelo usuário.

- Kernel sigmoidal: este kernel utiliza uma separação da seguinte forma:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \tanh(\gamma(\mathbf{x_i} \cdot \mathbf{x_j}) + r) \tag{2.43}$$

  novamente, os parâmetros γ e r são definidos pelo usuário.

- Kernel RBF (*Radial Basis Function*) : esta função é uma das mais utilizadas, sendo definida de forma semelhante a uma distribuição Gaussiana, como:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma||\mathbf{x_i} - \mathbf{x_j}||^2) \tag{2.44}$$

  Como este kernel considera diretamente a distância entre os pontos, ele possui uma abordagem com estratégia similar ao k-NN, onde pontos mais próximos possuem uma influência maior (lembrando que o expoente e negativo e $\gamma > 0$).

  No caso do RBF, somente dois parâmetros devem ser ajustados: γ e o parâmetro de regularização C. Como discutido anteriormente, um valor baixo de C diminui o peso dos dados classificados de forma errada ou dentro da margem, gerando uma classificação mais suave, enquanto que um valor alto tende a classificar mais pontos corretamente com uma menor generalização. Como este parâmetro multiplica todo o somatório, ele pode ser visto como um parâmetro *global*.

O parâmetro γ, por sua vez, pondera o peso de cada par do conjunto de treinamento. Quanto maior o valor de γ, mais próximo os pontos devem estar para serem afetados (novamente, lembrando que o expoente é negativo). Com isso, altos valores de γ tendem a gerar uma fronteira de decisão mais irregular, com curvas em volta de elementos individuais (semelhante ao k-NN). Baixos valores de γ, por sua vez, geram uma fronteira mais suave. A escolha de valores adequados para C e γ é muito importante, pois estes tem uma influência muito grande no resultado.

Um exemplo de aplicação de máquinas de suporte com kernel não-linear para classificação binária é apresentado neste vídeo[15].

2.5 Naive Bayes

Os métodos baseados na abordagem *Naive Bayes* são uma classe de algoritmos probabilísticos, onde é determinada a probabilidade de um dado elemento, possuindo um conjunto de atributos, pertencer a uma classe específica. Esta classificação é feita com base no *Teorema de Bayes*[16], que será apresentado a seguir. Mais detalhes sobre a aplicação do teorema de Bayes em algoritmos de classificação podem ser encontrados em Kubat (2017).

2.5.1 Introdução ao Teorema de Bayes

Para entender como a classificação Bayesiana é realizada, é interessante antes definir os conceitos de conhecimentos *a priori* e *a posteriori*. Estes termos foram definitos por Kant na introdução à sua *Crítica da Razão Pura* (Kant, 2012). O conhecimento *a priori* é "*todo aquele que seja adquirido independentemente de qualquer experiência*", ou seja, o conhecimento que não depende de observações, aquilo que pode ser assumido como verdade. Nas palavras do autor, "*(...) se alguém escava os alicerces de uma casa, a priori poderá esperar que ela desabe, sem precisar observar a experiência da sua queda, pois, praticamente, já sabe que todo corpo abandonado no ar sem sustentação cai ao impulso da gravidade*".

O conhecimento *a posteriori*, em oposição, é todo aquele que depende da experiência ou observação, aquilo que não pode ser assumido sem que uma análise do sistema seja feita.

[15] https://www.youtube.com/watch?v=VCXngi7-U4s

[16] Para uma excelente explicação deste teorema, veja este vídeo.

Por exemplo, pode-se considerar *a priori* que todo fluido escoa, pois esta informação está contida na definição de fluido. Porém, para afirmar, por exemplo, que todo fluido em um dado processo é líquido, devem ser feitas observações que comprovem ou refutem esta ideia, sendo este um conhecimento *a posteriori*.

No contexto de algoritmos de classificação, os conhecimentos *a priori* são as definições que não dependem da relação entre os atributos de um elemento e sua classe. Para ilustrar, considere uma versão simplificada do exemplo avaliado no início deste capítulo, onde um processo foi classificado como *adequado* ou *não-adequado* em função da temperatura e pressão. Para simplificar, estes dois casos serão tratados como *positivo* e *negativo*. Além disso, considere também que o único atribuito analisado foi a temperatura e esta podia assumir somente 2 valores, alta ou baixa. Um possível cenário de análise nesta condição é representado na Tabela 2.4, com um conjunto de 100 amostras.

Table 2.4: Classificação para ilustração do Teorema de Bayes.

Condição	Amostras	Condição	Amostras
Processo positivo	50	Temperatura alta e processo positivo	45
Processo negativo	50	Temperatura alta e processo negativo	15
Temperatura alta	60	Temperatura baixa e processo positivo	5
Temperatura baixa	40	Temperatura baixa e processo negativo	35

Considerando os valores apresentados, como existem 50 processos positivos em 100 amostras, a probabilidade de um processo qualquer ser positivo é de 50%, o que pode ser expresso como $P(pos) = 0.5$. De forma semelhante, a probabilidade de ser negativo também é 50% ($P(neg) = 0.5$). Com relação à temperatura, existe 60% de chance de uma amostra possuir temperatura alta e 40% de possuir temperatura baixa, assim, $P(alta) = 0.6$ e $P(baixa) = 0.4$. Veja que estes valores não dependem da relação entre a temperatura e o tipo de processo, por isso, são informações conhecidas *a priori*.

Avaliando a segunda coluna da tabela, obtém-se uma relação entre as variáveis temperatura e tipo de processo. Esta é uma informação *a posteriori*, pois depende de uma observação que relacione as duas variáveis. Considerando as 60 amostras com temperatura alta, 45 delas equivalem a um processo positivo, o que quer dizer que *dado um processo com temperatura alta, existe 45/60 = 75% de chance de ele ser positivo*. Isto pode ser representado como $P(pos|alta) = 0.75$. O símbolo '|' pode ser lido como '*dado que*'.

Agora, pode-se fazer a pergunta contrária: Qual a probabilidade de uma amostra ter temperatura alta dado que o processo é positivo? Para isto, utiliza-se o teorema de Bayes, que neste caso pode ser apresentado como:

$$P(alta|pos) = \frac{P(pos|alta)P(alta)}{P(pos)} \tag{2.45}$$

Substituindo os valores, obtém-se que $P(alta|pos) = 0.9$. Isto está de acordo com os dados da tabela, onde de 50 amostras positivas, 45 possuem temperatura alta ($45/50 = 0.9$). De forma genérica, o teorema de Bayes pode ser expresso como:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.46}$$

ou seja, a probabilidade de A ocorrer desde que B seja verdadeiro é igual à probabilidade de B ocorrer desde que A seja verdadeiro multiplicado pela razão entre as probabilidades de A e B ocorrerem individualmente. Apesar de simples, o teorema de Bayes é capaz de relacionar conhecimentos *a priori* e *a posteriori*, sendo um dos mais importantes teoremas na teoria da probabilidade.

Exemplo: Uma aplicação prática do teorema de Bayes é na determinação da eficácia de testes de doenças. Por exemplo, considere que uma pessoa aleatória faça um teste rápido do tipo IgG para Covid-19 e o resultado seja positivo. Como este resultado pode ser transformado em uma *probabilidade* de que a pessoa de fato tenha a doença? Ou seja, qual $P(covid|pos)$?

Para responder esta pergunta, a primeira coisa que precisamos saber é $P(pos|covid)$, ou seja, qual a chance de o resultado ser positivo dado que a pessoa possua a doença. Este valor é chamado de *sensibilidade* do teste[17]. Considerando, por exemplo, o teste COVID-19 IgG ECO, sua sensibilidade é de 95%, ou seja, a cada 100 pessoas infectadas analisadas, 5 resultados serão *falsos negativos*. Porém, isto não quer dizer que uma pessoa analisada e com resultado positivo tem 95% de chance de estar com a doença, para responder isto é preciso também conhecer $P(covid)$ e $P(pos)$, ou seja, as probabilidades individuais de alguém ter a doença e a de ser testado positivo.

Considerando o número de casos it **confirmados** em 16/06/2020, 0.432% da população brasileira possui a doença, ou seja, para esta amostra $P(covid) = 0.00432$. Por último, é preciso saber a probabilidade de um indivíduo ser testado como positivo ($P(pos)$).

[17] A sensibilidade e a especificidade de diferentes testes podem ser encontradas neste relatório. Obs.: Sensibilidade = (positivos verdadeiros)/(positivos verdadeiros + falsos positivos), especificidade = (negativos verdadeiros)/(negativos verdadeiros + falsos positivos)

Para determinar este valor, é conveniente pensar em duas diferentes situações: existe uma chance de o indivíduo testar positivo e ter a doença (positivo verdadeiro) e uma chance de testar positivo e não ter a doença (falso positivo). A chance de ter um positivo verdadeiro será igual ao produto da sensibilidade do teste pela probabilidade de o indivíduo ter a doença ($P(covid|pos)P(covid)$). De forma semelhante, a chance de um falso positivo será igual a probabilidade de o resultado ser positivo para uma pessoal saudável ($P(pos|saudavel)$) multiplicado pela probabilidade de o indivíduo ser saudável ($1-P(covid)$). O valor $P(pos|saudavel)$ é relacionado com a *especifidade* do teste (1 - especificidade), e para o COVID-19 IgG ECO é de 0.01. Com isso, o teorema de Bayes pode ser expresso como:

$$P(pos|covid) = \frac{P(covid|pos)P(pos)}{P(covid|pos)P(covid) + P(pos|saudavel)(1 - P(covid))} \tag{2.47}$$

Substituindo os valores, obtém-se que $P(pos|covid) = 29.18\%$, ou seja, a probabilidade de uma pessoa aleatória, testada como positivo, ter a doença é de menos de 30%. Esta é uma das grandes dificuldades em se analisar um grande número de indivíduos e obter dados precisos utilizando testes rápidos.

2.5.2 Classificação Bayesiana

Considere agora um caso onde existe um conjunto de amostras contendo vetores $\mathbf{x}$ com diversos atributos e classes y associadas, da forma:

$$T = (\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), (\mathbf{x_3}, y_3), \dots (\mathbf{x_m}, y_m) \tag{2.48}$$

Neste caso, o vetor com os atribuitos pode possuir um número qualquer de valores, bem como é permitido que y possua mais de duas classes (classificação multiclasse). O teorema de Bayes ainda pode ser aplicado nesse caso. Por exemplo, a probabilidade de um elemento $\mathbf{x}$ pertencer a umda da classe y_i pode ser representado como $P(y_i|\mathbf{x})$ (o que pode ser lido como a probabilidade da classe ser y_i dado que o vetor dos atributos seja $\mathbf{x}$). Usando o teorema de Bayes, isto pode ser calculado como:

$$P(y_i|\mathbf{x}) = \frac{P(\mathbf{x}|y_i)P(y_i)}{P(\mathbf{x})} \tag{2.49}$$

Para saber a qual classe o elemento $\mathbf{x}$ possui maior chance de pertencer, deve-se avaliar sua probabilidade com relação a todas as possíveis classes y_i e determinar o valor máximo. Como o objetivo é maximizar $P(y_i|\mathbf{x})$, esta estratégia é muitas vezes chamada de *máximo a*

posteriori. Veja que na expressão anterior o denominador não depende de y_i, de modo que pode-se pensar que o objetivo é encontrar a classe que maximize o termo $P(\mathbf{x}|y_i)P(y_i)$.

O termo $P(y_i)$ é fácil de ser determinado, basta avaliar a frequência com que a classe y_i aparece no conjunto de dados (ou seja, somar todos os elementos pertencentes ao grupo y_i e dividir pelo número total de elementos).

O termo $P(\mathbf{x}|y_i)$ não é tão simples de ser calculado. Este termo pode ser entendido como a probabilidade de um elemento pertencente à classe y_i ser descrito exatamente pelo vetor $\mathbf{x}$. Em sistemas simples, com poucos atribuitos e onde estes podem assumir poucos valores, eventualmente este valor pode ser diretamente calculado, porém na maioria dos casos isso não é possível. Considere, por exemplo, um caso onde cada elemento possui 12 atributos e cada atributo pode assumir 8 valores. A chance de um vetor $\mathbf{x}$ aleatório possuir os 12 elementos que correspondam exatamente à valores presentes no conjunto de testes é muito baixa [18].

A ideia dos algoritmos baseados em Naive Bayes é considerar que pode-se relacionar a probabilidade $P(\mathbf{x}|y_i)$ com a probabilidade associada com cada elemento do vetor $\mathbf{x}$ individualmente. Considere que $\mathbf{x} = (x_1, x_2, x_3, \ldots x_n)$, a probabilidade de que a amostra seja da classe y_i dado que o i-ésimo elemento do vetor $\mathbf{x}$ seja x_i será $P(x_i|y_i)$. A hipótese de Naive Bayes é assumir que $P(\mathbf{x}|y_i)$ pode ser calculado como o produtório das probabilidades associadas a cada elemento do vetor $\mathbf{x}$, ou seja:

$$P(\mathbf{x}|y_i) = \prod_{i=1}^{n} P(x_i|y_i) \tag{2.50}$$

Esta abordagem desconsidera qualquer efeito que um atributo possa ter sobre o outro, assumindo que todos são independentes. Por isso, é chamada de *naive* (ingênua). Juntando as expressões obtidas anteriormente, pode-se definir que o classificador baseado em Naive Bayes irá buscar a classe y_i tal que o termo $P(y_i)\prod_{i=1}^{n} P(x_i|y_i)$ seja maximizado, sendo que este problema de otimização pode ser expresso como:

$$\max_{y} P(y_i) \prod_{i=1}^{n} P(x_i|y_i) \tag{2.51}$$

Para casos onde os atributos podem assumir apenas um conjunto de valores, as probabilidades individuais $P(x_i|y_i)$ podem ser calculadas como a frequência relativa com que os elementos pertencentes à classe y_i possuem o atributo x_i (número de elementos da classe y_i que possuem o atributo x_i dividido pelo total de elementos da classe y_i). Com isso, o

[18] 1 chance em 68719476736.

problema pode ser resolvido e a probabilidade de uma dada amostra $\mathbf{x}$ pertencer a cada uma das classes y_i pode ser calculada.

Porém, quando os atribuitos x_i são variáveis contínuas (como por a temperatura ou pressão no exemplo inicial deste capítulo), é necessário transformar esta probabilidade discreta em contínua, o que pode ser facilmente realizado utilizando uma *função de densidade de probabilidade*, como será discutido a seguir.

2.5.3 Funções de Densidade de Probabilidade (PDF)

No caso de atributos que podem assumir valores contínuos, deve-se aplicar alguma estratégia para determinar como esta variável afeta a probabilidade da classificação. Uma maneira simples é dividir o intervalo onde esta variável está definido em subdomínios, ou seja, *discretizar* esta variável. Por exemplo, no início desta seção a variável *temperatura* foi dividida em duas classes (alta e baixa). Para ilustrar, considere que uma *temperatura alta* corresponde a mais de 300°C e uma *temperatura baixa* a menos de 300°C. No exemplo avaliado, haviam 60 amostras na classe *alta* e 40 na classe *baixa*.

É claro que neste processo perde-se muita informação, já que existe uma grande faixa de valores definidos, por exemplo, como *temperatura alta*. Para contornar este problema, pode-se definir um número maior de subclasses, cada uma englobando uma faixa menor de temperatura. Com isso, diminui-se também o número de elementos em cada classe. Suponha, no entanto, que o conjunto de dados seja muito grande (tendendo a infinito). Pode-se continuar dividindo os dados em subdomínios cada vez menores, até se obter divisões infinitesimais. Neste limite, a função que relaciona o número de amostras presentes em cada um dos subdomínios se torna uma *função contínua* $p(x)$. Esta função é chamada de ***função de densidade de probabilidade*** (pdf). Um valor alto de $p(x)$ indica que existem muitos elementos no conjunto de dados com o atribuito (temperatura, por exemplo) próximo a x, enquanto que um valor baixo indica que poucos elementos possuem esse valor.

Seguindo a nomenclatura normalmente utilizada, as pdf's serão identificadas por $p(x)$, porém, deve-se ter cuidado para não confundir estas funções com as probabilidades $P(x)$ usadas até então. De forma geral, $p(x)$ representa a pdf associada com todos os elementos, independente da classe a qual eles pertencem. Caso a função seja obtida considerando somente os elementos de uma dada classe y_i, ela será identificada como $p_{y_i}(x)$.

A função $p_{y_i}(x)$ representa a probabilidade de que os elementos da classe y_i possuam um

dado atributo com valor de x. Esta interpretação é muito similar ao conceito de $P(x_i|y_i)$, ou seja, a probabilidade de o atributo ser igual ao valor discreto x_i dado que a classe seja y_i. Por isso, na análise de atributos contínuos, o teorema de Bayes é reescrito da seguinte forma:

$$P(y_i|x) = \frac{p_{y_i}(x)P(y_i)}{p(x)} \tag{2.52}$$

Nesta equação, $P(y_i)$ continua sendo a frequência com que os elementos da classe y_i estão presentes (não depende do atribuito x). Assim como discutido anteriormente para atributos discretos, deve-se encontrar a classe com maior probabilidade de conter o elemento com o atributo x, ou seja, deve-se encontrar o maior valor de $P(y_i|x)$ dentre todas as classes y_i. Como $p(x)$ não depende da classe, novamente o objetivo é então maximizar o numerador da expressão acima.

Na prática, é comum que existam diversos atributos associados com cada elemento de modo que não existe uma única $p_{y_i}(x)$. Seguindo a mesma ideia do Naive Bayes, será considerado que os atribuitos são mutuamente independentes. Para um caso onde os atributos sejam dados por um vetor $\mathbf{x} = (x_1, x_2, \ldots x_n)$, a função de densidade de probabilidade com relação à este vetor será calculada como o produtório das pdf's com relação a cada um dos termos:

$$p_{y_i}(\mathbf{x}) = \prod_{i=1}^{n} p_{y_i}(x_i) \tag{2.53}$$

Com esta expressão, o teorema de Bayes, para estimar a probabilidade de um elemento com todos os atributos $(x_1, x_2, x_3, \ldots, x_n$ pertencer à classe y_i, pode ser expresso como:

$$P(y_i|\mathbf{x}) = \frac{P(y_i)}{p(\mathbf{x})} \prod_{i=1}^{n} p_{y_i}(x_i) \tag{2.54}$$

Como o objetivo é encontrar a classe y_i que maximize a expressão anterior e $p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i)$ não depende de y_i, o problema de otimização apresentado na Eq. 2.51 pode ser avaliado como:

$$\max_{y_i} P(y_i) \prod_{i=1}^{n} p_{y_i}(x_i) \tag{2.55}$$

Para finalizar o problema, resta a questão de como obter cada pdf $p_{y_i}(x_i)$. Normalmente, é nesta etapa que os diversos algoritmos baseados em Naive Bayes se diferem. Detalhes sobre este procedimento serão apresentados a seguir.

2.5.4 Obtenção das PDF's individuais

Para obter as funções de densidade de probabilidade, é comum a utilização de curvas padrão que se adequam a grande parte dos casos. A mais utilizada é a função Gaussiana, onde a pdf é calculada como:

$$p_{y_i}(x_i) = \frac{1}{\sqrt{2\pi\sigma_{y_i}^2}} \exp\left(-\frac{(x_i - \mu_{y_i})^2}{2\sigma_{y_i}^2}\right) \tag{2.56}$$

O resultado desta equação são as clássicas curvas mostradas na Figura 2.19. Para ajustar os parâmetros σ_{y_i} e μ_{y_i} (variância e média), pode-se utilizar os próprios valores conhecidos de x_i. Por exemplo, considere que existam m elementos no conjunto de dados que pertencem à classe y_i. Os parâmetros podem ser avaliados como:

$$\mu_{y_i} = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad\qquad \sigma_{y_i}^2 = \frac{1}{m-1}\sum_{i=1}^{m}(x_i - \mu)^2 \tag{2.57}$$

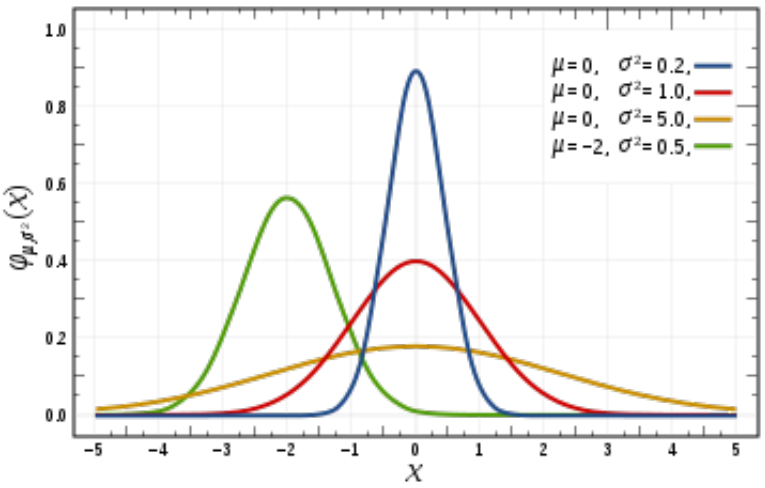


Figure 2.19: Exemplos de funções Gaussianas.

Para casos onde a distribuição Gaussiana não se encaixa bem na representação dos dados, pode-se utilizar outras formas de distribuição, como por exemplo uma *distribuição multinomial*[19]. Neste caso, as funções de densidade de probabilidade são calculadas como:

$$p_{y_i}(x_i) = \frac{k + \alpha}{n + \alpha m} \tag{2.58}$$

onde k é o número de vezes que o atributo x_i aparece em um elemento da classe y_i, n é o número total de atributos presntes nos elementos da classe y_i, m é o número total de

[19]Esta é uma das abordagens mais utilizadas para classificação de textos, por exemplo, para classificação de spams, linguagem em que foi escrito, sentimentos associados ao texto, etc.

atributos de cada elemento e α é um parâmetro de suavização definido pelo usuário, com valores entre 0 e 1. Por exemplo, se $\alpha = 0$, a função irá retornar k/n, que é a frequência com que o atribuito x_i aparece nos elementos da classe y_i. Uma vantagem de utilizar $\alpha \geq 0$ é que, caso algum atributo não apareça no conjunto de treinamento, sua probabilidade de aparecer no futuro não será automaticamente zerada.

Considerando as características gerais dos modelos baseados em Naive Bayes, estes algoritmos costumam apresentar um bom desempenho na classificação de conjuntos de dados grandes e onde a hipótese *ingênua* de que os um atributo não influencia no outro é válida.

3 Algoritmos de Regressão

Os algoritmos de aprendizagem que relacionam um conjunto de atributos de entrada com uma ou mais saídas contínuas, que podem assumir qualquer valor dentro de um intervalo, são chamados de algoritmos de regressão. O exemplo mais simples deste tipo de modelo é o simples ajuste de pontos a uma curva $y = f(x)$, onde pode-se associar um único atributo x com uma saída y. Porém, assim como no caso da classificação, pode ser necessário associar a saída com uma série de atributos em um vetor $\boldsymbol{x}$.

A maioria dos algoritmos de classificação podem ser aplicados para regressão tratando as classes como valores contínuos e não discretos. Por exemplo, considere o caso do k-NN. Uma forma de aplicação deste algoritmo para regressão consiste em obter os *valores* (e não as classes) associados com os k vizinhos mais próximos do ponto avaliado e atribuir a média destes valores para o ponto.

Neste capítulo serão apresentados os conceitos básicos relacionados com a obtenção e aplicação de algoritmo de regressão linear e utilizando redes neurais. A regressão linear consiste basicamente em ajustar os pontos a um modelo linearmente dependente de um conjunto de parâmetros, portanto, aqueles já familiarizados com a área de estimação de parâmetros verão muitas similaridades. É importante definir alguns conceitos básicos de forma clara, pois estes serão posteriormente generalizados para a formulação de algoritmos baseados em redes neurais.

3.1 Regressão Linear

De forma geral, o objetivo dos algoritmos de regressão é prever o valor de uma ou mais variáveis contínuas y, normalmente chamadas de *variáveis objetivo*, a partir de um vetor $\boldsymbol{x}$ contendo n atributos conhecidos. Assim, dado um conjunto de treinamento contendo m elementos $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)^T$ juntamente com os *valores* associados das variáveis objetivo

para cada elemento, $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)^T$, o objetivo é prever o valor y_i correspondente a um novo vetor $\boldsymbol{x}_i$.

Assim como no caso dos problemas de classificação, existe uma incerteza associada com a regressão. Do ponto de vista probabilístico, um modelo de regressão é utilizado para obter a probabilidade condicional $p(y|\boldsymbol{x})$, ou seja, uma função que descreve a distribuição de probabilidade da variável objetivo assumir o valor y dada uma entrada $\boldsymbol{x}$.

A maneira mais simples de relacionar o vetor de atributos com a variável objetivo é utilizando uma função $f(\boldsymbol{x})$. Inicialmente, considere que exista somente uma variável objetivo, de modo que $f(\boldsymbol{x})$ mapeia uma entrada $\boldsymbol{x} \in \mathbb{R}^n$ com uma saída $f(\boldsymbol{x}) \in \mathbb{R}$. Utilizando esta formulação, a variável objetivo prevista para um elemento $\boldsymbol{x}_i$ pode ser expressa como $y_i = f(\boldsymbol{x}_i) + \varepsilon$, onde ε é uma medida do erro associado com a representação, podendo ser causado por ruído no dados de treinamento ou simplesmente porque o modelo proposto para $f(\boldsymbol{x})$ não representa bem os dados. É comum considerar que ε possua uma distribuição Gaussiana com média igual a zero e variância σ. Na Figura 3.1 é ilustrado um exemplo de regressão (linha azul) considerando pontos associados com uma entrada contendo somente um atributo x. São apresentadas também curvas (linhas amarelas) ilustrando a distribuição de probabilidade associada com alguns pontos, seguindo uma distribuição Gaussiana para ε.

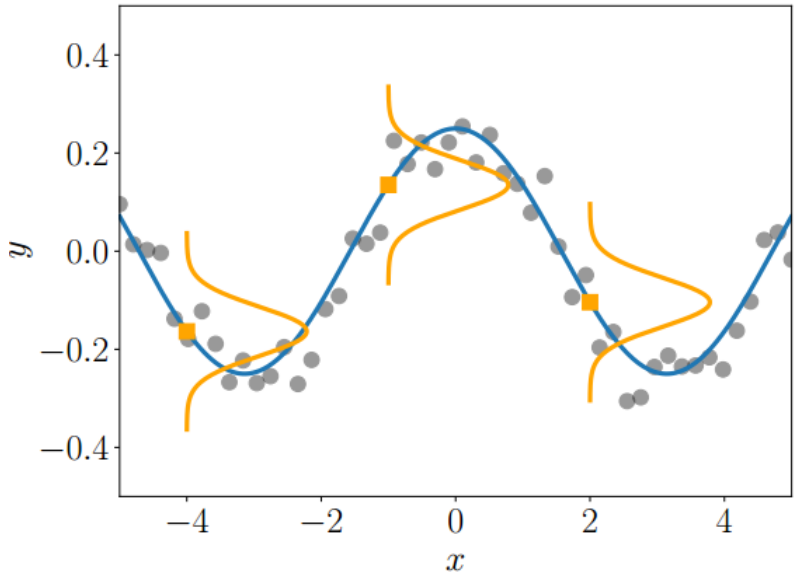


Figure 3.1: Exemplo de regressão. Pontos representam dados de treinamento com ruído. A linha azul representa o modelo $f(x)$, enquanto que as linhas amarelas representam $y = f(x)+\varepsilon$ considerando uma distribuição Gaussiana para ε. Fonte: (Deisenroth et al., 2020)

A escolha da função $f(\boldsymbol{x})$ é obviamente muito importante para o problema. Assim como para os problemas de classificação, deve-se buscar uma representação capaz de *generalizar* os dados e com isso prever de maneira correta valores associados com novos pontos. Por isso, uma estratégia de simplesmente minimizar o desvio ε pode levar a sobreajuste e deve ser utilizada com cautela. Conforme apresentado por (Deisenroth et al., 2020), a escolha de um modelo de regressão adequado esta associada com a resolução de uma série de problemas:

- Escolha de um formato para função $f(\boldsymbol{x})$ juntamente com uma parametrização (número de parâmetros $\boldsymbol{\theta}$ da função) adequados: Por exemplo, se for adotada uma representação polinomial, qual a ordem deste polinômio?

- Determinação dos parâmetros de $f(\boldsymbol{x})$: Nesta etapa, deve-se utilizar uma estratégia de otimização que permita obter os parâmetros $\boldsymbol{\theta}$ que minimizem o erro, como por exemplo a escolha de funções de perda adequadas e qual algoritmo de otimização será utilizado para minimizar a perda;

- Relação do modelo escolhido com sobreajuste: Modelos muito complexos (com muitos parâmetros) tendem a se ajustar melhor aos dados de treinamento, porém podem levar à sobreajuste e perda da capacidade de generalização;

- Modelagem da incerteza dos dados: A partir de um conjunto finito de dados de treinamento, como é possível obter uma medida da confiança relacionada com o modelo obtido?

Uma análise detalhada de todos estes objetivos está além do objetivo deste material, porém, será apresentado a seguir uma estratégia geral associada com a obtenção dos parâmetros $\boldsymbol{\theta}$ para o caso da regressão linear. A função $f(\boldsymbol{x})$ utilizada é chamada de *função base*, podendo ser uma função linear dos atributos de entrada ou uma função não-linear. A seguir será considerado o caso de bases lineares, sendo na sequência este conceito estendido para bases não-lineares.

3.1.1 Utilização de Bases Lineares

O modelo mais simples para regressão é considerar que a variável objetivo possa ser expressa como uma combinação linear atributos de entrada, ponderados por um conjunto de

parâmetros $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$. Neste caso, a função que mapeia o vetor dos atributos com a variável objetivo irá depender de $\boldsymbol{x}$ e $\boldsymbol{\theta}$, podendo ser expressa como:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \theta_0 + \sum_{i=1}^{n} \theta_i x_i \tag{3.1}$$

Assim, é considerado que a variável objetivo tem uma dependência linear tanto em relação aos parâmetros θ_i quanto em relação aos atributos x_i.

O parâmetro θ_0, que não multiplica nenhum atributo, é muitas vezes chamado de parâmetro de viés (bias), sendo utilizado para deslocar todos os dados em uma quantidade fixa. Por conveniência, será inserido um elemento $x_0 = 1$ no vetor dos atributos $\boldsymbol{x} = (x_0, x_1, x_2, \ldots, x_n)$, permitindo que a expressão anterior possa ser escrita como:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{i=0}^{n} \theta_i x_i = \boldsymbol{\theta}^T \boldsymbol{x} \tag{3.2}$$

Obtenção de θ

Para estimar os parâmetros $\boldsymbol{\theta}$, o método mais simples e um dos mais utilizados é o dos *mínimos quadrados*, que consiste em minimizar a distância entre os pontos e o hiperplano formado pela combinação linear acima. Isto pode ser representado definindo uma *função erro* que estima esta diferença para um dado conjunto de parâmetro $\boldsymbol{\theta}$, sendo definida para um conjunto de treinamento com m elementos como:

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{m} (y_i - f(\boldsymbol{x}_i, \boldsymbol{\theta}))^2 = \sum_{i=1}^{m} (y_i - \boldsymbol{\theta}^T \boldsymbol{x}_i))^2 \tag{3.3}$$

Este procedimento é ilustrado na Figura 3.2 para um exemplo onde os atributos possuem dimensão 2 (neste caso, os pontos são representados em um espaço $\mathbb{R}^3$ para incluir a variável objetivo). As linhas representam a distância entre os pontos e o plano obtido como resultado da minimização do erro.

Considerando novamente um conjunto de treinamento dado por $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)^T$ e as respectivas variáveis objetivo dadas por $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)^T$, o somatório anterior pode ser escrito como:

$$E(\boldsymbol{\theta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) = ||(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})||^2 \tag{3.4}$$

Assim, obtém-se uma expressão quadrática em função de $\boldsymbol{\theta}$. Isto permite obter o ponto de mínimo associado com a função erro, avaliando onde o gradiente da função erro emr relação aos parâmetros $\boldsymbol{\theta}$ se iguala a zero. Para este caso, temos que:

$$\frac{dE}{d\boldsymbol{\theta}} = \frac{d}{d\boldsymbol{\theta}} \left(\boldsymbol{y}\boldsymbol{y}^T - 2\boldsymbol{y}^T \boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{\theta}^T \boldsymbol{X}^T \boldsymbol{\theta}\boldsymbol{X} \right) = -2\boldsymbol{y}^T \boldsymbol{X} + 2\boldsymbol{\theta}^T \boldsymbol{X}^T \boldsymbol{X} = -2\boldsymbol{X}(\boldsymbol{y}^T - \boldsymbol{X}^T \boldsymbol{\theta}^T) \tag{3.5}$$

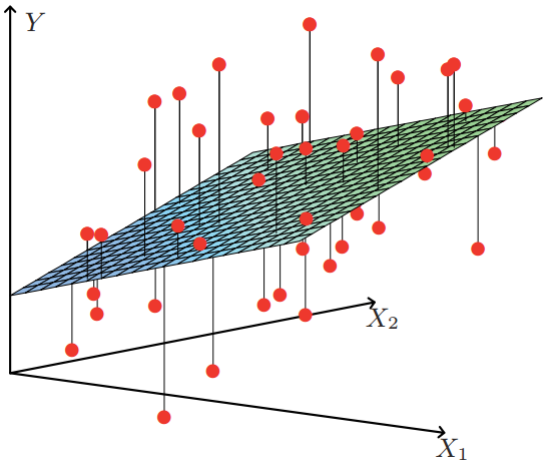


Figure 3.2: Representação de um ajuste por mínimos quadrados com $\boldsymbol{x} \in \mathbb{R}^2$. Fonte: (Hastie et al., 2009)

Avaliando a derivada segunda:

$$\frac{d^2E}{d\boldsymbol{\theta}^2} = 2\boldsymbol{X}\boldsymbol{X}^T = 2||\boldsymbol{X}||^2 \tag{3.6}$$

esta expressão será maior que zero desde que o posto (*rank*) de $\boldsymbol{X}$ seja igual a n, ou seja, tem-se uma equação linearmente independente para cada elemento de $\boldsymbol{\theta}$. Com isso, o ponto de inflexão corresponde a um ponto de mínimo (derivada segunda positiva).

Igualando a zero para obter o ponto de mínimo, obtém-se o conjunto de parâmetros $\boldsymbol{\theta} = \boldsymbol{\theta}_{min}$ que minimizam o erro:

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\theta}_{min}^T = \boldsymbol{X}\boldsymbol{y}^T \tag{3.7}$$

Multiplicando ambos os lados por $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ e transpondo a expressão obtida, temos:

$$\boldsymbol{\theta}_{min} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{3.8}$$

A obtenção desta expressão analítica para os parâmetros é uma das principais vantagens da utilização de mínimos quadrados. De maneira equivalente, considerando que os dados possuem um ruído dado por uma distribuição Gaussiana, pode-se chegar em uma expressão equivalente utilizando o princípio da máxima verossimilhança, conforme mostrado, por exemplo, em (Deisenroth et al., 2020).

É importante ressaltar que essa formulação só pode ser utilizada se as colunas da matriz $\boldsymbol{X}$ forem linearmente independentes. Por exemplo, se um dos elementos do conjunto de treinamento for dado como uma combinação linear dos demais, o termo $\boldsymbol{X}\boldsymbol{X}^T$ será singular

e portanto a sua inversa não pode ser calculada. Este problema pode ser resolvido removendo os elementos linearmente dependentes, quando possível.

Regularização

Problemas onde a matriz estiver muito próxima de ser singular ou quando o número de atributos for maior que o número de elementos podem ser minimizados utilizando um *parâmetro de regularização*, semelhante ao aplicado para as máquinas de vetores de suporte e também discutido na Seção 2.2.3. Com isso, a função erro passa a ser avaliada como:

$$E(\boldsymbol{\theta}) = ||(\boldsymbol{y} - \boldsymbol{X\theta})||^2 + \lambda||\boldsymbol{\theta}||^2 \tag{3.9}$$

Fazendo o mesmo procedimento para esta função, os parâmetros que minimizam o erro podem ser avaliados como:

$$\boldsymbol{\theta}_{min} = (\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{3.10}$$

Partindo do valor obtido para o conjunto ótimo dos parâmetros $\boldsymbol{\theta}$, pode-se prever o valor da variável objetivo associada com uma entrada $\boldsymbol{x}$:

$$\hat{y} = \boldsymbol{\theta}_{min}^T\boldsymbol{x} = \theta_{min,0} + \sum_{i=1}^{n}\theta_{min,i}x_i \tag{3.11}$$

onde $\hat{y}$ é utilizado para representar o valor previsto pelo método.

Obtenção da Variância

Conforme comentado no início desta seção, é comum adotar uma distribuição Gaussiana para estimar o ruído presente nos dados, permitindo a obtenção de uma distribuição de probabilidade de que a variável objetivo seja exatamente $\hat{y}$. O valor "real" da variável objetivo pode ser expresso como $y = \hat{y} + \varepsilon$, onde:

$$\varepsilon = \frac{1}{\sqrt{2\pi\sigma_{min}^2}}\exp\left(-\frac{1}{2\sigma_{min}^2}(y - \hat{y}_i)^2\right) \tag{3.12}$$

Conforme demostrando em (Deisenroth et al., 2020), a variância associada com $\boldsymbol{\theta}_{min}$ pode ser calculada como:

$$\sigma_{min}^2 = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{3.13}$$

Utilizando esta expressão, é possível obter as distribuições (curvas amarelas) ilustradas na Figura 3.1.

O procedimento apresentado nesta seção considera que existe uma relação linear entre os atributos e a variável objetivo. Esta é uma hipótese que claramente pode não ser válida, porém, pode-se estender a regressão linear para incluir uma dependência não-linear entre as variáveis através da utilização de *funções base não-lineares*, como será discutido a seguir.

3.1.2 Utilização de Bases Não-Lineares

Para considerar uma dependência não-linear entre os atributos e a variável objetivo, uma abordagem muito utilizada é realizar uma transformação não-linear arbitrária nos elemenos $\boldsymbol{x}$, denotada por $\boldsymbol{\phi}(\boldsymbol{x})$, e na sequência realizar uma combinação linear dos elementos desta transformação.

Por exemplo, considere uma transformação $\boldsymbol{\phi}(\boldsymbol{x}) = (\phi_0(\boldsymbol{x}), \phi_1(\boldsymbol{x}), \ldots, \phi_{K-1}(\boldsymbol{x}))^T$ que mapeie os elementos de um espaço original com dimensão n para um espaço com dimensão K ($\boldsymbol{\phi} : \mathbb{R}^n \rightarrow \mathbb{R}^K$), através de uma transformação dos valores dos atributos $\boldsymbol{x}$ em escalares ($\phi_i : \mathbb{R}^n \rightarrow \mathbb{R}$). Fazendo uma combinação linear destes valores, obtém-se:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{i=0}^{K-1} \theta_i \phi_i(\boldsymbol{x}) = \boldsymbol{\theta}\boldsymbol{\phi}^T(\boldsymbol{x}) \tag{3.14}$$

O valor $K-1$ é utilizado no somatório para que o problema tenha K parâmetros θ. Observe que esta expressão continua sendo uma função *linear* em relação aos parâmetros θ, por isso, ainda se trata de uma regressão linear, já que o objetivo é encontrar estes parâmetros.

Diversas bases pode ser utilizadas, dependendo das características dos dados, como por exemplo funções Gaussianas, sigmoidais e até mesmo bases de Fourier (veja Bishop (2016) para mais detalhes). A abordagem para obtenção dos parâmetros não depende muito da escolha destas bases, por isso, o procedimento visto anteriormente continua válido, visto que a Equação 3.2 pode ser obtida fazendo $\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{x}$. Para ilustrar isto, a seguir será apresentado um exemplo considerando bases polinomiais, sendo este um dos casos mais utilizados.

Exemplo de Bases Polinomiais

Para ilustrar a utilização de bases polinomiais, considere um caso contendo somente um atributo, de modo que $\boldsymbol{x} = x$. Dessa forma, o modelo de regressão irá buscar o valor $y = \boldsymbol{\phi}(x)\boldsymbol{\theta} + \varepsilon$ associado a x. A utilização de bases polinomiais consiste em definir uma

transformação da foma:

$$\boldsymbol{\phi}(x) = \begin{bmatrix} \phi_0(x) \\ \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_{K-1}(x) \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^{K-1} \end{bmatrix} \tag{3.15}$$

Observe que a partir de um atributo original $x \in \mathbb{R}$ obtém-se uma representação $\boldsymbol{\phi}(x) \in \mathbb{R}^K$. Esta estratégia é muito semelhante à utilização de *kernels* para as máquinas de vetores de suporte. A partir da definição desta base polinomial, é possível aplicar uma regressão linear para obter os parâmetros $\boldsymbol{\theta} \in \mathbb{R}^K$. A transformação imposta por esta base é equivalente a expressar a função $f(\boldsymbol{\theta}, x)$ como:

$$f(\boldsymbol{\theta}, x) = \sum_{i=0}^{K-1} \theta_i x^i \tag{3.16}$$

ou seja, a função está sendo ajustada a um polinômio de ordem $K-1$.

Obtenção dos Parâmetros θ

Considere agora novamente um caso geral onde o conjunto de treinamento é composto por m elementos associado vetores de entrada contendo n atributos ($\boldsymbol{x} \in \mathbb{R}^n$) com uma única variável objetivo ($y \in \mathbb{R}$). A aplicação da transformação $\boldsymbol{\phi}(\boldsymbol{x})$ irá gerar uma matriz $m \times K$ $\boldsymbol{\Phi}$ dada por:

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}^T(\boldsymbol{x}_1) \\ \boldsymbol{\phi}^T(\boldsymbol{x}_2) \\ \boldsymbol{\phi}^T(\boldsymbol{x}_3) \\ \vdots \\ \boldsymbol{\phi}^T(\boldsymbol{x}_m) \end{bmatrix} = \begin{bmatrix} \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \dots & \phi_{K-1}(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \phi_1(\boldsymbol{x}_2) & \dots & \phi_{K-1}(\boldsymbol{x}_2) \\ \phi_0(\boldsymbol{x}_3) & \phi_1(\boldsymbol{x}_3) & \dots & \phi_{K-1}(\boldsymbol{x}_3) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_m) & \phi_1(\boldsymbol{x}_m) & \dots & \phi_{K-1}(\boldsymbol{x}_m) \end{bmatrix} \tag{3.17}$$

Lembrando que todos os elementos $\boldsymbol{\Phi}_{ij} = \phi_j(\boldsymbol{x}_i)$ são operações que retornam um valor escalar, ou seja, $\phi_j : \mathbb{R}^n \to \mathbb{R}$. Esta matriz $\boldsymbol{\Phi}$ pode ser interpretada como uma nova matriz dos atributos obtida, sendo equivalente à matriz $\boldsymbol{X}$ utilizada anteriormente para bases lineares.

Com a matriz obtida, a função erro pode ser reavaliada considerando a transformação realizada como:

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{m} (y_i - \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{x}_i))^2 = (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^T (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}) \tag{3.18}$$

Utilizando o mesmo procedimento de obtenção dos valores de $\boldsymbol{\theta}$ que minimizem o erro apresentado anteriormente para as bases lineares, temos que:

$$\boldsymbol{\theta}_{min} = (\boldsymbol{\Phi}\boldsymbol{\Phi}^T)^{-1}\boldsymbol{\Phi}^T\boldsymbol{y} \tag{3.19}$$

Assim como para o caso da bases lineares, para que seja possível calcular a inversa presente na expressão anterior, o posto da matriz $\boldsymbol{\Phi}$ deve ser igual ao seu número de colunas (K), portanto, as transformações aplicadas por $\boldsymbol{\phi}$ não podem ser transformações lineares.

Sobreajuste na Regressão Linear

A possibilidade de aplicar uma transformação $\boldsymbol{\phi} : \mathbb{R}^n \rightarrow \mathbb{R}^K$, onde K é um parâmetro definido pelo usuário, permite a obtenção de uma regressão com uma quantidade virtualmente infinita de parâmetros (lembrando que $\boldsymbol{\theta} \in \mathbb{R}^K$). Porém, como comentado na Seção 2.2.3 para os problemas de classificação, modelos muito complexos tendem a se ajustar muito bem aos dados de treinamento, porém possuem baixa capacidade de generalização, não sendo adequados para prever novos valores. A mesma lógica se aplica para os problemas de regressão.

Para determinar a qualidade do ajuste aos dados de treinamento, diferentes métricas podem ser aplicadas. Uma das mais usadas é a raiz do erro médio quadrático (RMSE), definida como:

$$RMSE = \sqrt{\frac{1}{m}||\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}||^2} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \boldsymbol{\phi}^T(\boldsymbol{x}_i)\boldsymbol{\theta})^2} \tag{3.20}$$

Para ilustrar a influência da função base no comportamento do sistema, considere o exemplo mostrado na Figura 3.3, onde é apresentado o ajuste obtido para um conjunto de 10 dados considerando polinômios com ordem gradativamente maiores, desde um polinômio de ordem zero (constante) até ordem 9.

Como pode ser visto, para ordens muito baixas ($M = 0 - 3$) a curva não se ajusta bem aos dados de treinamento (subajuste). Para valores intermediários ($M = 4 - 6$) o ajuste é adequado, enquanto que para valores muito altos claramente observa-se um sobreajuste. O polinômio passa por todos os pontos, porém oscila de forma errática entre eles, indicando que pontos não presentes no conjunto de treinamento não serão estimados corretamente.

Assim como no caso da classificação, para avaliar a presença de sobreajuste pode-se dividir o conjunto de dados em conjunto de treinamento e de teste. Para o sistema apresentado na

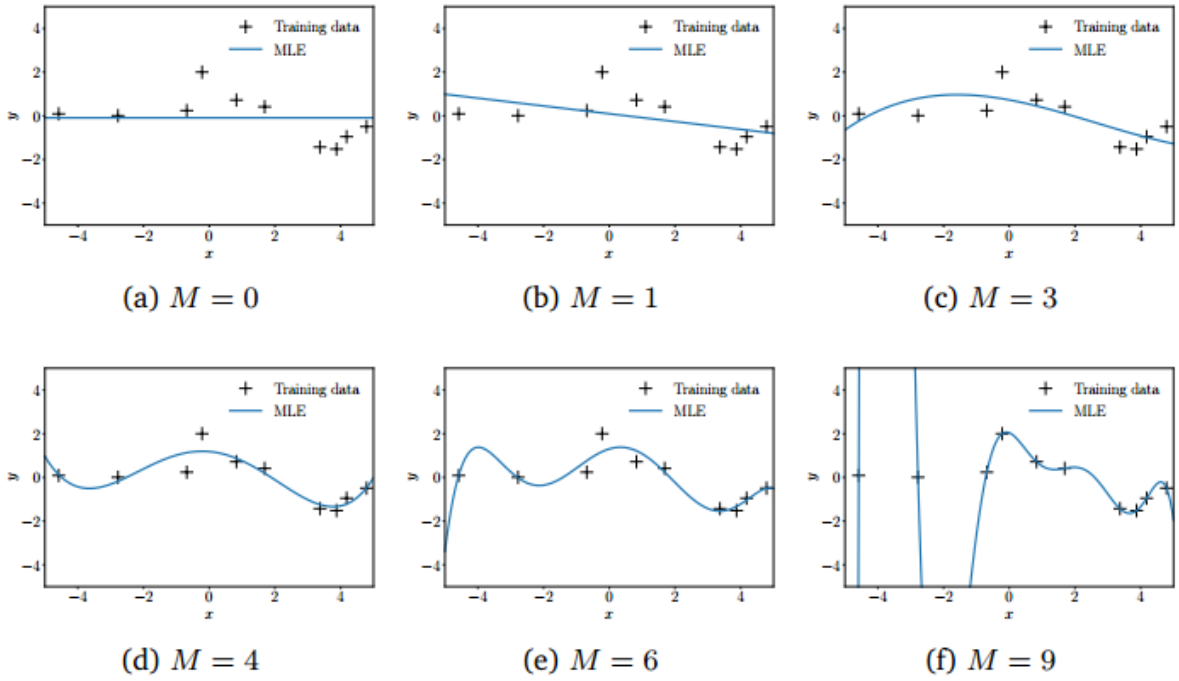


Figure 3.3: Ajuste de regressão considerando polinômios de diferentes ordens M. Fonte: (Deisenroth et al., 2020)

Figura 3.3, pode-se esperar que o erro associado aos conjuntos de treinamento e de teste possuam um comportamento como o mostrado na Figura 3.4, onde o erro associado ao conjunto de teste em um ponto de mínimo para $M = 4$, sendo portanto este o valor com melhor desempenho.

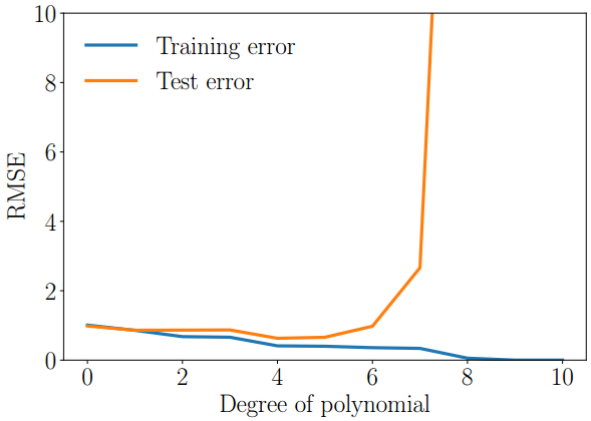


Figure 3.4: Erro associado com os conjuntos de treinamento e de teste para em função da ordem do polinômio de ajuste. Fonte: (Deisenroth et al., 2020)

Novamente, uma maneira de evitar o sobreajuste é a utilização de um parâmetro de regularização. Neste caso, a função erro passa a ser avaliada como:

$$E(\boldsymbol{\theta}) = ||(\boldsymbol{y} - \boldsymbol{\Phi\theta})||^2 + \lambda||\boldsymbol{\theta}||^2 \tag{3.21}$$

E os parâmetros que minimizam esta função serão dados por

$$\boldsymbol{\theta}_{min} = (\boldsymbol{\Phi\Phi}^T + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}^T\boldsymbol{y} \tag{3.22}$$

Um exemplo de aplicação de regressão linear é apresentado neste vídeo[1].

3.2 Regressão Utilizando Redes Neurais Artificiais

O termo Redes Neurais Artificiais (ANN) é utilizado para designar um grande número de algoritmos de aprendizagem que possuem em comum a utilização de elementos intermediários para relacionar um vetor de entrada, contendo n atributos, com uma ou mais variáveis objetivo de saída. Estes elementos intermediários, chamados de "camadas ocultas", permitem estabelecer uma relação não-linear entre os elementos de entrada e saída, ao custo de um aumento significativo na quantidade de parâmetros que precisa ser estimada. O termo "aprendizagem profunda" (*deep learning*) costuma ser utilizado para identificar ANN's com um grande número de camadas ocultas.

Os princípios básicos de aplicação das redes neurais são muito antigos, tendo sido propostos ainda nos anos 50. Nos anos 90 houve um grande interesse na aplicação de redes neurais em engenharia, mas os resultados foram em geral pouco satisfatórios devido à baixa capacidade de processamento e obtenção de modelos complexos. Recentemente, as ANN voltaram a receber atenção devido ao rápido avanço da área de inteligência artificial e seu grande potencial de aplicação na análise e automação de processos.

A utilização de redes neurais costuma ser justificada quando a quantidade de dados disponíveis é muito grande e o usuário tem acesso a uma capacidade de processamento elevada, já que a etapa de treinamento da rede pode ser muito custosa (Aggarwal, 2018). Neste material, será considerado o caso mais simples de rede neural, chamadas usualmente de *rede perceptron multicamadas* (MLP) ou *redes neurais feed-foward*. Diversos outros modelos, como por exemplo redes neuras recorrentes e redes neurais convolucionais (usadas em processamento de imagens), tem sido amplamente aplicados nos últimos anos

[1] https://www.youtube.com/watch?v=_T36ZnacC78

Desbancando mitos: *Nos últimos anos, observou-se a existência de interlocutores incautos, na maioria das vezes mal informados, que tratam redes neurais como algoritmos com propriedades mágicas de aprendizado e potencial de escravizar a humanidade em um futuro próximo. Apesar de lúdica, esta informação é, para a surpresa de muitos, totalmente falsa. Como será visto aqui, redes neurais nada mais são do que uma generalização do conceito de regressão linear para incluir transformações não-lineares, com potencial de destruição equivalente a uma planilha do Excel. O termo "neural" é mantido por uma questão histórica, hoje sabe-se que isto tem muito pouca relação com a forma como organismos vivos dotados de sistema nervoso central processam informação.*

A seguir serão apresentados alguns conceitos básicos introdutórios ao estudo de redes neurais, bem como serão definidos alguns termos utilizados na área.

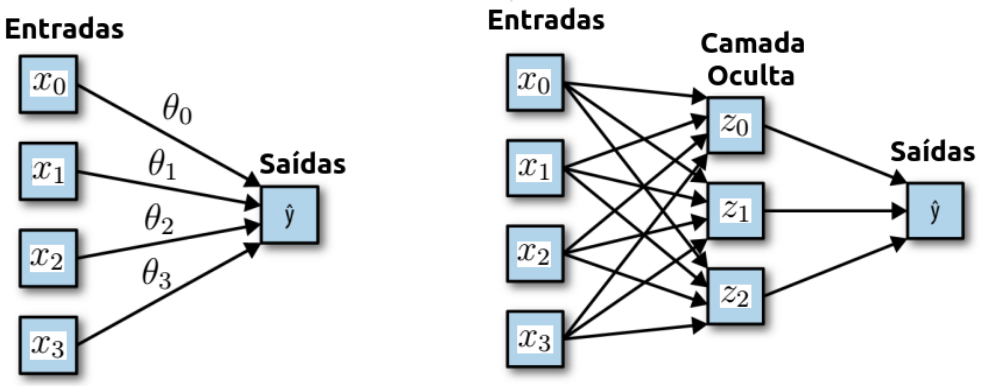
3.2.1 Conceitos Básicos e Terminologia

Considere inicialmente um processo simples de regressão linear utilizando bases lineares, como apresentado anteriormente, para relacionar um vetor de atributos $\boldsymbol{x} = (x_1, x_2, x_3, \ldots, x_n)$ com uma variável objetivo y. Para simplificar a simbologia, será acrescentado também um elemento $x_0 = 1$ no conjunto de dados $\boldsymbol{x}$. Desconsiderando por enquanto a presença de ruído nos dados ($\varepsilon = 0$, de modo que $y = \hat{y}$), a relação entre a saída e a entrada pode ser expressa como:

$$y = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \sum_{i=0}^{n} \theta_i x_i \tag{3.23}$$

Esta relação indica que a variável objetivo é calculada como uma soma *ponderada* dos atributos de entrada, sendo que os valores são ponderados por um conjunto de parâmetros $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$. Este procedimento é ilustrad0 na Figura 3.5(a) para $n = 3$. Como estes parâmetros ponderam a relação entre a entrada e a saída, no contexto de redes neurais são chamados normalmente de ***pesos***.

A estrutura básica de uma rede perceptron multicamadas (MPL) consiste em repetir este processo de somas ponderadas múltiplas vezes, através da adição de etapas intermediárias (camadas ocultas), como ilustrado na Figura 3.5(b). Cada elemento individual da rede é tratado como um *neurônio*, sendo por isso uma combinação em rede deles chamada de *rede neural*. Neste exemplo, uma soma ponderada dos atributos de entrada é utilizada para calcular elementos ocultos z_0, z_1 e z_2. A partir de uma ponderação destes elementos, a



(a) Regressão Linear (b) ANN

Figure 3.5: Ilustração da estrutura básica de (a) um problema de regessão linear com 4 atributos e uma variável objetivo, parametrizado por $\boldsymbol{\theta}$ e (b) uma ANN com uma única camada oculta. Adaptado de: (Muller and Guido, 2017)

variável objetivo y é então calculada. Os valores intermediários são chamados de "ocultos" pois não possuem um significado físico, conhecer estes valores não é necessário para aplicar o modelo.

No exemplo ilustrado, para aplicar a regressão linear, é preciso determinar quatro pesos, enquanto que para a ANN são 15 parâmetros a serem calculados. (12 para ponderar a relação entre $\boldsymbol{x}$ e $\boldsymbol{z}$ e três para relação de y com $\boldsymbol{z}$).

Se os valores $\boldsymbol{z}$ forem calculados como uma combinação linear de $\boldsymbol{x}$ e y for calculado simplesmente como uma combinação linear de $\boldsymbol{z}$, no final das contas y será uma combinação linear de $\boldsymbol{x}$, o que é equivalente à regressão linear simples, não trazendo nenhuma vantajem. Para acrescentar uma dependência mais complexa entre os elementos de entrada e saída, uma função não-linear é aplicada na soma ponderada para obter os elementos $\boldsymbol{z}$, permitindo com que a rede se ajuste a problemas muito mais complexos.

Para o problema ilustrado na Figura 3.5, este procedimento pode ser descrito da seguinte forma:

1. Inicialmente, é obtida uma soma ponderada dos elementos $\boldsymbol{x} = (x_0, x_1, x_2, x_3)$ para obter um valor intermediário a_i associada a cada termo z_i. Por exemplo, o elemento a_0 é calculado como:

$$a_0 = w_{0,0}x_0 + w_{0,1}x_1 + w_{0,2}x_2 + w_{0,2}x_3 \tag{3.24}$$

onde $w_{i,j}$ é o parâmetro peso que relaciona a_i e x_j.

2. Para obter z_i, uma função não-linear, chamada de *função de ativação* é aplicada em a_i. Uma das funções de ativação mais utilizada é a tangente hiperbólica. Usando ela como exemplo, z_0 seria calculado como:

$$z_0 = \tanh(a_0) \tag{3.25}$$

3. Após determinar os elementos ocultos, a variável objetivo é calculada como uma soma ponderada dos destes elementos:

$$y = v_0 z_0 + v_1 z_1 + v_2 z_2 \tag{3.26}$$

onde v_i é o parâmetro peso que relaciona z_i com y.

Neste problema, os parâmetros $w_{i,j}$ e v_i precisam ser determinados a partir dos dados de treinamento. O número de elementos ocultos utilizado também precisa ser definido pelo usuário, sendo que este possui uma grande influência no resultado final e no gasto computacional. Além disso, os elementos ocultos não precisam estar todos em uma única camada, pode-se ter diversas camadas ocultas com diferentes elementos em cada uma delas. Isto gera uma possibilidade de combinações de número de camadas e de elementos muito grande, sendo por isso a eficácia das MPL muito dependentes de um ajuste adequado.

A seguir serão apresentados mais detalhes sobre como os parâmetros podem ser determinados e como o exemplo ilustrado nesta seção pode ser generalizado.

3.2.2 Formulação Geral de uma MLP

Considere o diagrama apresentado na Figura 3.6 demonstrando a topologia de uma rede neural com uma camada de elementos ocultos. Esta rede relaciona os neurônios de entrada ($\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$) com os de saída ($\boldsymbol{y} = (y_1, y_2, \ldots, y_K)$). Para isto, são utilizados M elementos ocultos $\boldsymbol{z} = (z_1, z_2, \ldots, z_M)$, além de parâmetros de viés (*bias*) adicionados em cada camada (exceto a última), denotados por b_0 e b_1.

Esta estrutura é normalmente chamada de uma rede neural com três camadas, considerando também as camadas relacionadas com os elementos de entrada e saída. Observe que não existe ligação entre neurônios em uma mesma camada.

A ponderação entre um elemento z_i e um elemento x_j é feita por um parâmetro peso $w_{ij}^{(1)}$, enquanto que a ponderação entre um elemento y_i e um elemento z_j é feita por um parâmetro peso $w_{ij}^{(2)}$. Os pesos que multiplicam os parâmetros de viés serão identificados como $w_{i,0}^{(k)}$,

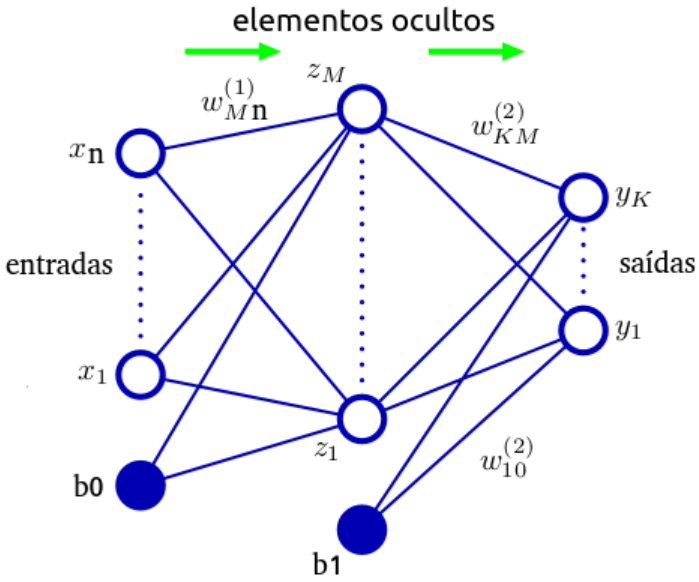


Figure 3.6: Diagrama geral de uma MLP contendo uma camada de elementos ocultos. Adaptado de: (Bishop, 2016)

ponderando o viés da camada k com o elemento i da camada $k+1$. Estes parâmetros peso devem ser estimados com base em um conjunto de treinamento relacionando $\boldsymbol{x}$ com $\boldsymbol{y}$.

Os parâmetros de viés também precisam ser estimados, porém, como estes irão multiplicar parâmetros peso específicos, é conveniente definir $b_0 = b_1 = 1$, de modo a reduzir o número de parâmetros que precisa ser estimado. Assim, ao invés de estimar $b_i w^{(k)}_{j,0}$, é estimado somente $w^{(k)}_{j,0}$, o que é equivalente já que estes termos não aparecem em nenhuma outra operação.

Como comentado anteriormente, inicialmente é realizada uma combinação linear das variáveis de entrada para obter as variáveis a_j, chamadas de variáveis de ativação (ou variáveis de pré-ativação):

$$a_j = \sum_{i=1}^{n} w^{(1)}_{j,i} x_i + w^{(1)}_{j,0} \qquad j = 1, 2, 3, \ldots, M \tag{3.27}$$

Os elementos da camada oculta, z_i, são obtidos aplicando uma *função de ativação* não-linear $h(\cdot)$ em a_i. As funções de ativação mais utilizadas normalmente seguem uma distribuição aproximadamente sigmoidal, como será discutido a seguir.

Escolha da Função de Ativação

A escolha de uma função de ativação adequada é uma etapa fundamental do ajuste das MLP. Dada uma função de ativação adequada e um número suficiente de elementos, uma rede MLP é capaz de representar qualquer função. Algumas das funções mais que podem

ser utilizadas são apresentadas a seguir, com o comportamento destas funções ilustrado na Figura 3.7.:

- Função identidade:

$$h(x) = x \tag{3.28}$$

- Função sigmoidal (logística):

$$h(x) = \frac{1}{1 + e^{-x}} \tag{3.29}$$

- Tangente hiperbólica:

$$h(x) = tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{3.30}$$

- Unidade linear retificada (ReLU):

$$h(x) = max(0, x) \tag{3.31}$$

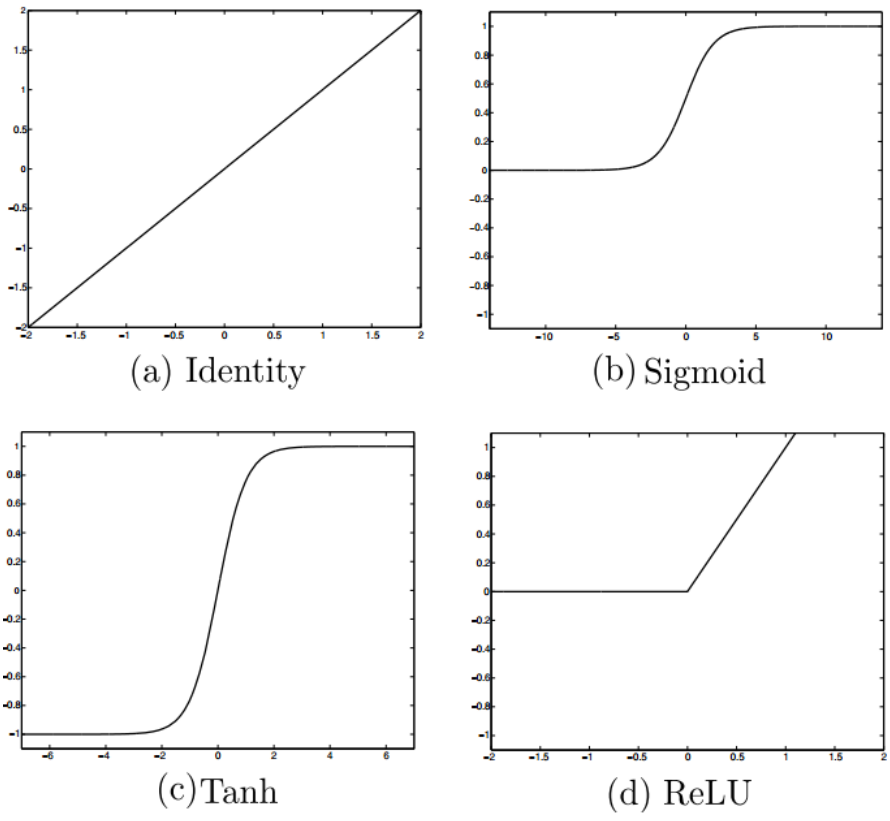


Figure 3.7: Exemplos de funções de ativação. Adaptado de: (Aggarwal, 2018)

A função identidade, como comentado anteriormente, não introduz nenhuma não-linearidade no sistema, reduzido a rede neural a um problema de regressão linear, por isso sua utilização é limitada. Em problemas de regressão, as variáveis objetivo são calculadas diretamente como

uma combinação linear dos elementos da última camada oculta. Então, pode-se imaginar que uma função de ativação do tipo identidade é aplicada neste processo.

A função sigmoidal retorna valores entre 0 e 1, o que é interessante para modelos probabilísticos, enquanto que a tangente hiperbólica é mais adequada para problemas onde as variáveis podem assumir valores negativos. Historicamente, estas duas funções foram muito utilizadas, mas recentemente têm sido substituídas por funções que facilitam o processo de otimização pra obtenção dos parâmetros, como por exemplo a função ReLU.

A função ReLU é muito parecida com a identidade, com a diferença que irá resultar em 0 para $x < 0$, o que a torna não-linear. Uma grande vantagem desta função é que sua derivada é definida e constante para todo o intervalo (0 para $x < 0$ e 1 para $x > 0$. Em $x = 0$ pode-se adotar um dos dois valores). Isto facilita a aplicação de algoritmos de otimização como o método do gradiente descendente, muito utilizado para estimar os parâmetros de redes neurais.

Obtenção das Variáveis Objetivo

Após a definição de uma função de ativação, pode-se então calcular os valores dos M elementos ocultos:

$$z_i = h(a_i) \qquad\qquad i = 1, 2, 3, \ldots, M \tag{3.32}$$

Caso a rede possua mais de uma camada de elementos ocultos, pode-se repetir este processo. Uma nova ponderação dos elementos z_i é realizada para obter novas variáveis de ativação e a função de ativação é aplicada nestas variáveis para obtenção dos elementos da segunda camada oculta. Neste exemplo, será considerado somente uma camada oculta, de modo que as variáveis objetivo são diretamente calculadas como uma combinação linear dos elementos de $\boldsymbol{z}$:

$$\hat{y}_j = \sum_{i=1}^{M} w_{j,i}^{(2)} z_i + w_{j,0}^{(2)} \qquad\qquad j = 1, 2, 3, \ldots, K \tag{3.33}$$

Combinando as expressões anteriores, pode-se expressar as variáveis objetivo como:

$$\hat{y}_k(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=1}^{M} w_{k,j}^{(2)} h\left(\sum_{i=1}^{n} w_{j,i}^{(1)} x_i + w_{j,0}^{(1)} \right) + w_{k,0}^{(2)} \qquad\qquad k = 1, 2, 3, \ldots, K \tag{3.34}$$

onde o termo $\hat{y}_k$ é utilizado para representar o valor *previsto* pela rede neural.

O número total de parâmetros que deve ser ajustado neste caso será $(n+1) \times (M+1) + (M+1)K$. Observe que os elementos do conjunto de treinamento, x_i, só aparecem no termo

mais interno, referente à primeira etapa. Este resultado é usado para calcular os valores da segunda camada (oculta) e estes são usados então para obter a variável objetivo. Como a informação se propaga somente na direção da primeira para a última camada, este tipo de rede neural também é chamada de *rede neural de alimentação direta (feed foward)*.

Para simplificar a representação anterior, será novamente inserido um termo $x_0 = 1$ no vetor $\boldsymbol{x}$ e um termo $z_0 = 1$ no vetor $\boldsymbol{z}$. Com isso, os pesos associados com os parâmetros de viés podem ser incluídos no somatório e a expressão anterior pode ser reescrita como:

$$\hat{y}_k(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=0}^{M} w_{k,j}^{(2)} h \left(\sum_{i=0}^{n} w_{j,i}^{(1)} x_i \right) \tag{3.35}$$

A aplicação de redes neurais também pode ser estendida para problemas de classificação, através da aplicação de uma função de ativação no resultado do somatório anterior. Por exemplo, considerando uma segunda função de ativação $g(\cdot)$, a variável objetivo passa a ser calculada como:

$$\hat{y}_k(\boldsymbol{x}, \boldsymbol{w}) = g \left(\sum_{j=0}^{M} w_{k,j}^{(2)} h \left(\sum_{i=0}^{n} w_{j,i}^{(1)} x_i \right) \right) \tag{3.36}$$

Por exemplo, se uma função sigmoidal for utilizada, os valores estarão entre 0 e 1. Com base nisso, pode-se dividir os elementos em duas classes (maior que 0.5 e menor que 0.5, por exemplo), permitindo a separação do conjunto de treinamento em grupos. No restante deste material será focado na aplicação para regressão, porém, os princípios para aplicação em classificação são essencialmente os mesmos.

A etapa de obtenção dos parâmetros é chamada de *treinamento da rede*, e será discutida a seguir.

3.2.3 Treinamento da Rede

Para o treinamento da rede, a primeira etapa é definir uma função de perda para representar o erro a ser minimizado. De maneira semelhante ao aplicado para a regressão linear (Eq. 3.3), será utilizado o erro quadrático médio, definido neste caso como:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{m} (\hat{y}(\boldsymbol{x}_i, \boldsymbol{w}) - y_i)^2 = \frac{1}{2} ||\hat{\boldsymbol{y}} - \boldsymbol{y}||^2 \tag{3.37}$$

o fator $1/2$ é adicionado por conveniência, pois irá simplificar expressões obtidas futuramente.

Para o caso da regressão linear é possível obter uma solução que minimize o erro de forma analítica, como discutido anteriormente. Infelizmente, para o caso das redes MLP isto não é possível. O método de otimização mais utilizado para resolver este problema é o *método do gradiente descendente*, discutido brevemente a seguir.[2].

Gradiente Descendente

A ideia geral do método do gradiente descendente é encontrar o ponto de mínimo da função erro $E(\boldsymbol{w})$ utilizando a informação do gradiente desta função. Partindo de um chute inicial para os parâmetros $\boldsymbol{w}^{[0]}$[3], o algoritmo "caminha" na direção do valor negativo do gradiente de $E(\boldsymbol{w})$ avaliado em $\boldsymbol{w}^{[0]}$, $\nabla E(\boldsymbol{w}^{[0]})$, para obter um valor atualizado para os parâmetros, $\boldsymbol{w}^{[1]}$. Este processo é repetido até se obter um valor onde $\nabla E(\boldsymbol{w}) \approx 0$, ou seja, um ponto de inflexão. Como o método avança na direção negativa do gradiente, este ponto será um ponto de mínimo. De forma geral, este processo iterativo pode ser representado como:

$$\boldsymbol{w}^{[i+1]} = \boldsymbol{w}^{[i]} - \eta \nabla E(\boldsymbol{w}^{[i]}) \tag{3.38}$$

onde η é um parâmetro chamado de *taxa de aprendizagem* que controla a taxa de atualização dos parâmetros. Se o valor de η for pequeno suficiente, pode-se esperar que $E(\boldsymbol{w}^{[i]}) > E(\boldsymbol{w}^{[i+1]})$, ou seja, o novo valor encontrado para os parâmetros gera um erro menor. Maiores valores de η aceleram a convergência, porém podem fazer com que o método divirja.

Para fazer a atualização dos parâmetros, o algoritmo pode primeiramente calcular o gradiente com relação à toda a amostra de elementos (gradiente descendente *batch*) ou ir atualizado com base no valor de cada amostra individual (gradiente descendente estocástico). A primeira abordagem pode gerar problemas para garantir que o ponto de mínimo encontrado seja um ponto de mínimo global e não local, especialmente se a função erro não for convexa, o que é comum no caso das redes neurais. Por isso, neste caso é utilizado o método estocástico.

Como visto anteriormente, a função erro a ser minimizada é a soma do erro associada com

[2] A obtenção e aplicação deste algoritmo não serão discutidas com detalhes neste material, porém, trata-se de um método de otimização padrão muito utilizado, podendo ser encontrado em qualquer livro-texto da área de otimização.

[3] O sobrescrito $^{[i]}$ será utilizado para designar a i-ésima iteração, enquanto que $^{(i)}$ representa os parâmetros peso que relacionam a camada i com a camada $i+1$ da rede MLP.

cada ponto, o que pode ser escrito como:

$$E(\boldsymbol{w}) = \sum_{i=1}^{m} E_i(\boldsymbol{w}), \qquad \text{onde} \qquad E_i(\boldsymbol{w}) = \frac{1}{2}(\hat{y}(\boldsymbol{x}_i, \boldsymbol{w}) - y_i)^2 \tag{3.39}$$

No caso do gradiente descendente estocástico, a atualização dos parâmetros é feita então com base em cada amostra i, de modo que:

$$\boldsymbol{w}^{[i+1]} = \boldsymbol{w}^{[i]} - \eta \nabla E_i(\boldsymbol{w}^{[i]}) \tag{3.40}$$

Os parâmetros $w_{j,i}$ representam o peso usado para ponderar a influência de um elemento i sobre um elemento j. A presença de dois índices torna o conjunto de parâmetros $\boldsymbol{w}$ uma matriz, porém, o número de elementos em cada camada não é necessariamente o mesmo, o que torna muito complicado avaliar o gradiente da função erro em relação a esta matriz. Por isso, para a etapa de atualização dos parâmetros, é mais conveniente expressar estes parâmetros como um vetor $\boldsymbol{w} = (w_1, w_2, w_3, \ldots, w_H)$, onde H é o número total de parâmetros. Neste formato, $\nabla E_i(\boldsymbol{w})$ é calculada como:

$$\nabla E_i(\boldsymbol{w}) = \begin{bmatrix} \dfrac{\partial E_i}{\partial w_1} \\ \dfrac{\partial E_i}{\partial w_2} \\ \vdots \\ \dfrac{\partial E_i}{\partial w_H} \end{bmatrix} \tag{3.41}$$

É irrelevante qual parâmetro $w_{j,i}$ é identificado como w_1, qual é identificado como w_2 e assim sucessivamente, desde que a mesma ordem seja mantida no cálculo do gradiente.

Para resolver este problema, basta agora encontrar uma maneira de estimar as derivadas parciais da função erro em relação aos parâmetros $w_{i,j}$. Para isto, é utilizada uma estratégia de *retropropagação*, como será discutido a seguir.

Retropropagação

Como destacado por Bishop (2016), o processo de minimização do erro pode ser dividido em duas etapas:

1. *Determinação das Derivadas:* Inicialmente, é atribuído um valor para os parâmetros $\boldsymbol{w}$. Com este valor, são calculados os valores de todos os elementos e as derivadas da função erro em relação à cada parâmetro $w_{j,i}$ é computada. Nesta etapa, as derivadas são avaliadas no sentido da saída para a entrada, como será discutido a seguir. Por isso, a abordagem é chamada de *retropropagação*;

2. *Atualização dos Parâmetros:* Nesta etapa, as derivadas são utilizadas para atualizar os valores do parâmetro, o que pode ser feito diretamente com a aplicação do método do gradiente descendente estocástico.

Observe que estas duas etapas são independentes, ou seja, é possível avaliar as derivadas usando uma abordagem diferente da retropropagação, bem como seria possível atualizar os parâmetros usando outros algoritmos de otimização. Além disso, uma função de perda diferente também poderia ser utilizada sem alterar a estrutura básica do algoritmo.

Como discutido anteriormente, no método do gradiente descendente estocástico pode-se avaliar os gradientes de cada amostra individualmente, por isso, nesta seção o foco será obter $\nabla(E_i)$. Inicialmente, considere que o valor de todos os parâmetros $\boldsymbol{w}$ seja conhecido (através da definição de um chute inicial, por exemplo).

Considerando uma rede com um número qualquer de camadas e de elementos, vimos anteriormente que a primeira consiste na determinação das variáveis de ativação (como mostrado, por exemplo, na Eq. 3.27). Generalizado este procedimento, podemos dizer que a variável de ativação a_j em uma camada $p+1$ qualquer é calculada com base nos valores dos elementos na camada p, representados aqui de forma geral por z_i, como:

$$a_j = \sum_{i=0}^{P} w_{j,i} z_i = w_{j,0} z_0 + w_{j,1} z_1 + w_{j,2} z_2 + \ldots + w_{j,P} z_P \tag{3.42}$$

onde P é o número total de elementos na camada p. Nesta expressão, já está sendo considerado que a variável de viés está sendo incluída como $z_0 = 1$. A expressão acima pode ser usada em qualquer camada. Por exemplo, se $p = 1$ (primeira camada), então $z_i = x_i$, ou seja, z_i será o valor dos atributos de entrada. Se $p+1$ corresponder à última camada, então $a_j = y_j$, ou seja, as variáveis de ativação serão as próprias variáveis objetivo. Em todos os casos, z_i representa uma variável que estabelece uma conexão com a_j ponderada por $w_{j,i}$.

Na segunda etapa de aplicação das redes MPL, uma função de ativação $h(\cdot)$ é aplicada nas variáveis a_j para obter z_j na mesma camada p:

$$z_j = h(a_j) \tag{3.43}$$

Para problemas de regressão, na última camada esta função será a função identidade ($z_j = a_j$), enquanto que para problemas de classificação pode-se aplicar uma função que separe os pontos em classes. O processo de determinação dos valores de a_j e z_j para todas as camadas também é chamado de *propagação para frente*, pois os valores são calculados no sentido da entrada para a saída.

Conhecendo agora os valores para todos os elementos, pode-se passar para a etapa de estimar a derivada do erro com relação a cada um dos w_{ij} parâmetros. Como visto anteriormente (Eq. 3.39), a função erro $E_i(\boldsymbol{w})$ é calculada com base na diferença entre o valor previsto na saída ($\hat{y}_i$) e o valor real (y_i). Como $w_{j,i}$ não aparece de forma explícita nesta expressão, pode-se utilizar a *regra da cadeia* para relacionar o erro com $w_{j,i}$, utilizando as variáveis de ativação como intermediária:

$$\frac{\partial E_n}{\partial w_{j,i}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{j,i}} \tag{3.44}$$

Para simplificar a simbologia, a primeira derivada do lado direito é chamada de δ_j, :

$$\delta_j = \frac{\partial E_n}{\partial a_j} \tag{3.45}$$

A segunda derivada pode ser relacionada diretamente com a definição das variáveis de ativação (Eq. 3.42):

$$\frac{\partial a_j}{\partial w_{j,i}} = z_i \tag{3.46}$$

Juntando as duas expressões, temos que:

$$\frac{\partial E_n}{\partial w_{j,i}} = \delta_j z_i \tag{3.47}$$

Lembrando, $w_{j,i}$ representa o parâmetro peso entre a variável z_i (elemento de entrada) e a_j (elemento de saída). Assim, esta expressão implica que as derivadas da função erro podem ser calculadas multiplicando o valor do elemento z_i por um termo δ_j associada com cada valor a_j. Como os valores de z_i já foram calculados na etapa anterior, resta agora determinar δ_j para os elementos necessários.

Obtenção de δ_j

Inicialmente, podemos calcular o valor de referente aos elementos na última camada. Para simplificar, os elementos da última camada serão identificados como $j = k$ (de acordo com a simbologia usada na Fig. 3.6). Neste caso, temos que $z_k = \hat{y}_k$, onde $\hat{y}_k$ é o valor previsto pela rede para a variável objetivo y_k. Além disso, no caso de regressão, temos que na última camada $a_k = z_k$ (função de ativação identidade), o que implica que $a_k = \hat{y}_k$. Considerando isto e usando a definição da função erro:

$$\delta_k = \frac{\partial E_n}{\partial a_k} = \frac{\partial E_n}{\partial \hat{y}_k} = \frac{\partial}{\partial \hat{y}_k}\left(\frac{1}{2}(\hat{y}_k - y_k)^2\right) \tag{3.48}$$

Resolvendo a integral acima (por exemplo, fazendo uma mudança de variável $u = \hat{y}_k - y_k$ e lembrando que y_k é constante), obtemos[4]:

$$\delta_k = \hat{y}_k - y_k \tag{3.49}$$

Para determinar os valores de δ_j dos elementos da penúltima camada, pode-se aplicar um procedimento similar. Primeiramente, podemos utilizar a regra da cadeia para relacionar a derivada do erro com relação à a_j com as derivadas que podem ser avaliadas. Neste caso, deve-se utilizar a *regra da cadeia multivariável* para levar em conta todas as conexões entre a_j e os elementos da última camada (a_k). Estas conexões são ilustradas na Figura 3.8 para um elemento z_j qualquer.

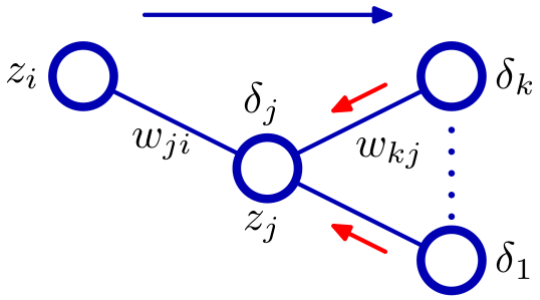


Figure 3.8: Conexões do elemento z_j. Fonte: (Bishop, 2016)

Aplicando a regra da cadeia, chega-se em:

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_{k=1}^{K} \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{3.50}$$

onde K representa o número total de elementos na última camada (como não existe elemento de viés nesta camada, o contador começa em 1).

A primeira derivada no lado direito é simplesmente a definição de δ_k, portanto, já é conhecida. A segunda derivada representa a dependência entre as variáveis de ativação em duas camadas adjacentes. Usando as definições anteriores, a variável a_k é calculada como:

$$a_k = \sum_{j=0}^{P} w_{k,j} z_j \tag{3.51}$$

onde P é o número total de elementos na penúltima camada. Considerando ainda que $z_j = h(a_j)$, esta equação pode ser escrita como:

$$a_k = \sum_{j=0}^{P} w_{k,j} h(a_j) = w_{k,0} h(a_0) + w_{k,1} h(a_1) + \ldots + w_{k,P} h(a_P) \tag{3.52}$$

[4]Este é o motivo pelo qual o fator 1/2 foi acrescentado na definição do erro, para "cortar" o termo da derivada.

Avaliando a derivada parcial de a_k com relação a um elemento a_j qualquer, temos que:

$$\frac{\partial a_k}{\partial a_j} = w_{k,j} h'(a_j) \tag{3.53}$$

onde $h'(a_j)$ representa a derivada da função de ativação avaliada em a_j. Substituindo na definição de δ_j:

$$\delta_j = \sum_{k=1}^{K} \delta_k w_{k,j} h'(a_j) = h'(a_j) \sum_{k=1}^{K} \delta_k w_{k,j} \tag{3.54}$$

Assim, conhecendo-se os valores de δ na última camada, é possível calcular os valores na camada adjacente. Caso a rede neural possua mais camadas ocultas, este processo é repetido para cada uma delas, tratando a última camada calculada como a camada k e determinando δ_j. Este procedimento deve ser repetido para toda as camadas ocultas, mas não é necessário para a primeira camada (com os valores dos atributos de entrada).

Como o cálculo é feito partindo-se da última camada e avançando em direção à primeira, este método é chamado de *retropropagação*. Após obter todos os valore de δ, o gradiente pode ser calculado e os parâmetros $\boldsymbol{w}$ atualizados, repetindo-se este processo até atingir a convergência desejada. Este procedimento pode ser resumido nas seguintes etapas:

1. Um chute inicial é atribuído para os parâmetros $w_{j,i}$, $\boldsymbol{w}^{[0]}$;
2. Utilizando este valor e um vetor com os atributos $\boldsymbol{x}_i$, são calculados os valores de a_j e z_j para todos os elementos da rede (propagação para frente);
3. Comparando os valores obtidos na camada de saída com as variáveis objetivo, determina-se δ_k;
4. Partindo do valor de δ_k, calcula-se os valores de δ_j para todos os elementos ocultos;
5. Com os valores de δ_j e z_j, determina-se a derivada da função erro em relação aos parâmetros $w_{i,j}$ (Eq. 3.47);
6. Com os valores das derivadas, atualiza-se os parâmetros $\boldsymbol{w}^{[0]}$ para obter $\boldsymbol{w}^{[1]}$ usando a Eq. 3.40;
7. Repete-se as etapas 2 - 6 o vetor com os atributos $\boldsymbol{x}_i$ até se obter a convergência desejada.

Ao final do processo, a rede está treinada e pronta para uso!

Um exemplo de aplicação de regressão utilizando redes MLP é apresentado neste vídeo[5].

[5] https://www.youtube.com/watch?v=N00uEaJONio

Bibliography

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning*. Springer New York.

Bishop, C. (2016). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer New York.

Deisenroth, M. P., Faisal, A. A., and Ong, C. S. (2020). *Mathematics for Machine Learning*. Cambridge University Press.

Géron, A. (2019). *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Incorporated.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer.

Hulsen, T., Jamuar, S. S., Moody, A. R., Karnes, J. H., Varga, O., Hedensted, S., Spreafico, R., Hafler, D. A., and McKinney, E. F. (2019). From big data to precision medicine. *Frontiers of Medicie*, 34(6):1–14.

Kant, I. (2012). *Crítica da razão pura*. F. C. Mattos (trad.), Ed. Vozes, 3a edition.

Kubat, M. (2017). *An Introduction to Machine Learning*. Springer International Publishing.

Le, H., Tran, T., and Tran, L. (2018). Automatic heart disease prediction using feature selection and data mining technique. *Journal of Computer Science and Cybernetics*, 34:33–48.

Muller, A. and Guido, S. (2017). *Introduction to Machine Learning with Python*. O'Reilly Media, Incorporated.