Introdução aos Algoritmos de Aprendizagem Supervisionada

PRR Plano de Recuperação e Resiliência
REPÚBLICA PORTUGUESA
Financiado pela União Europeia NextGenerationEU

1 Introdução **3**

1.1 Aprendizagem Supervisionada 3

1.2 Aprendizagem Não-Supervisionada 4

1.3 Aprendizagem por Reforço 4

2 Algoritmos de Classificação **6**

2.1 Aspectos Básicos dos Algoritmos de Classificação 6

2.1.1 Importação dos Dados 6

2.1.2 Divisão em Grupos de Teste e Treinamento 7

2.1.3 Aplicação do Algoritmo de Classificação e Análise dos Resultados . . 8

2.1.4 Extensão para Classificação Multi-Classe 11

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.

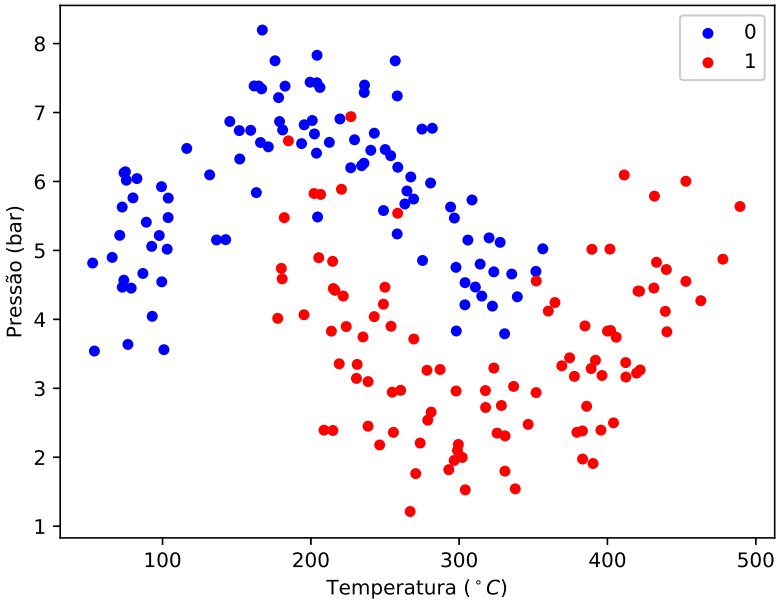


Figure 2.1: Conjunto de dados rotulados.

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.

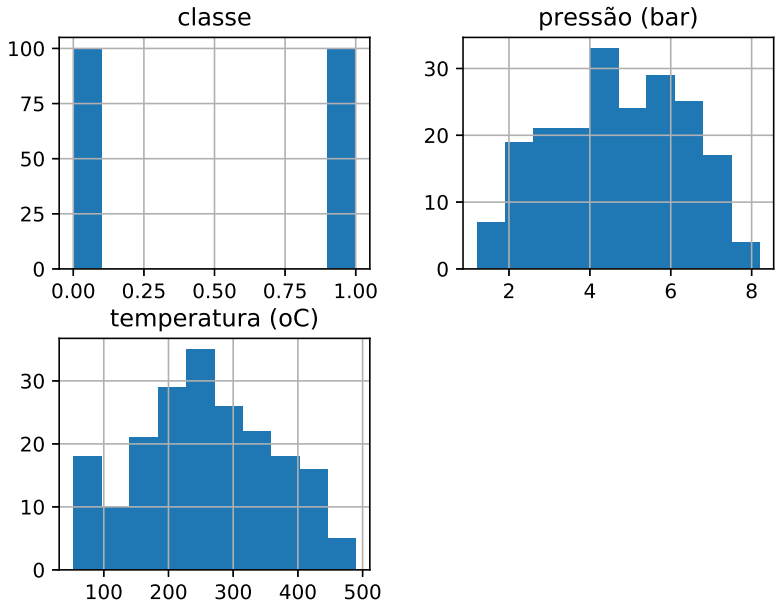


Figure 2.2: Histograma de distribuição dos pontos.

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.

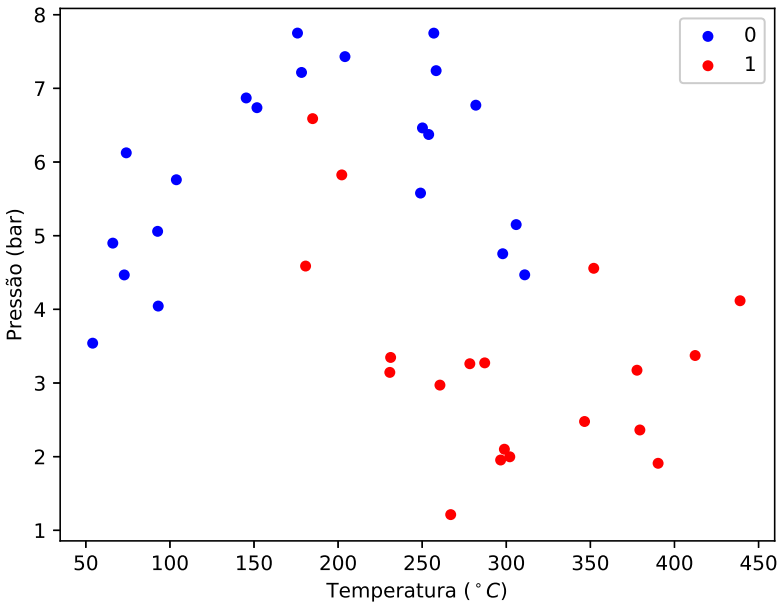


Figure 2.3: Grupo de teste considerando 20% dos dados.

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.

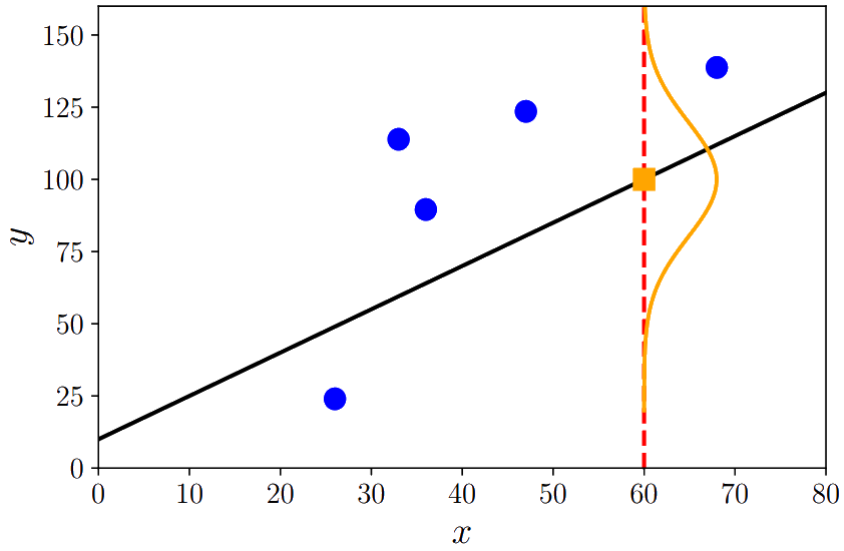


Figure 2.4: Comparação entre uma função preditiva (linha preta) e um modelo probabilístico (linha amarela). Fonte: Deisenroth et al. (2020)

.

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 **??** 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. **??**, 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?

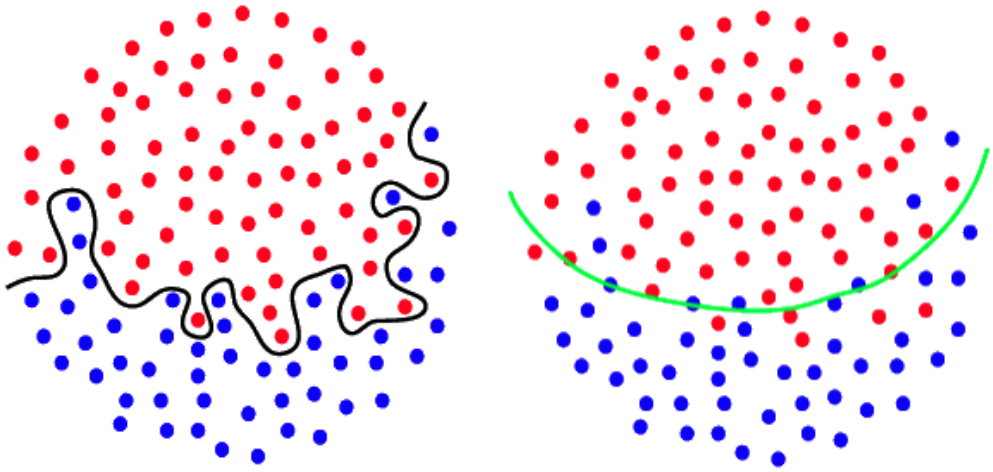
A primeira pergunta está associada com um conceito muito importante na área de aprendizagem que é a noção de sobreajuste (*overfitting*), que pode ser contornada com uma estratégia de *regularização*. A segunda pergunta pode ser respondida utilizado uma estratégia de *validação cruzada*. Estas duas abordagens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

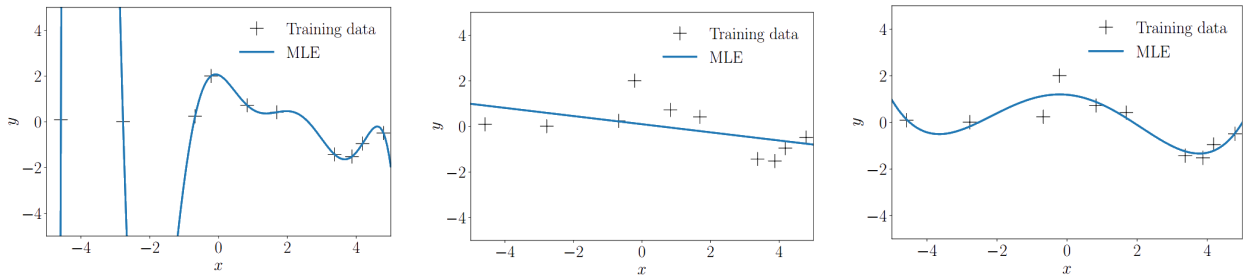
Por exemplo, considere os exemplos de sobreajuste ilustrados na Figura **??** para classificação e na Figura **??** pra regressão. No caso da classificação, o algoritmo é capaz de separar os dados perfeitamente nas duas classes, porém é muito provável que em um processo físico real esta fronteira irregular 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. **??**) 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 **??** para um exemplo de classificação na na Figura **??** 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. **??**.

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 **??** é ilustrada uma divisão em 5 subconjuntos (método 5-fold).

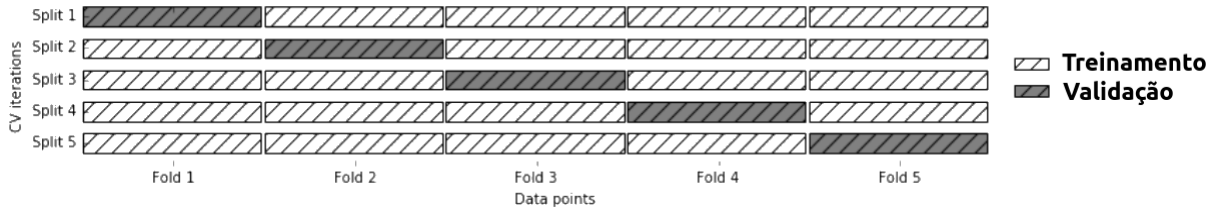


Figure 2.7: Divisão dos dados em conjuntos de treinamento (branco) e de validação (cinza) considerando uma abordagem 5-fold. Fonte: Muller and Guido (2017)

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 **??** 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:

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

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:

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

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:

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

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 **??** é apresentado uma ilustração das barreiras de divisão binária utilizando o algoritmo 1-NN.

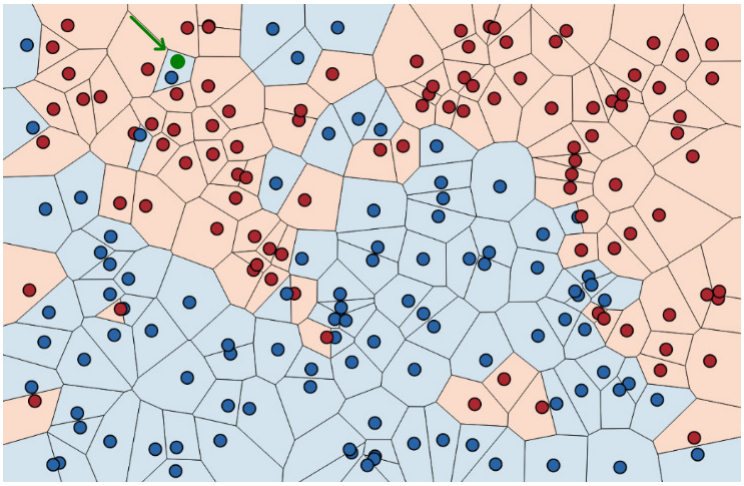


Figure 2.8: Exemplo de classificação usando 1-NN.

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 **??** (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.

Bibliography

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

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.

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