



Guia de estudos Data Science Machine Learning

Neuron/USP



GRUPO DE ESTUDOS EM *Data Science* NEURON/USP

FACEBOOK.COM/NEURONUSP

Este material, em conjunto com os Notebooks do Jupyter que o acompanham, é de propriedade do grupo Neuron/USP e, legalmente, não pode ser reproduzido para fins comerciais (sujeito a processos judiciais). Qualquer dúvida ou sugestão entre em contato conosco pela nossa página do Facebook ou e-mail: neuron.conteudo@gmail.com.

Autores: Leonardo Ernesto, Samuel Henrique, Felipe Maia Polo e Matheus Faleiros.

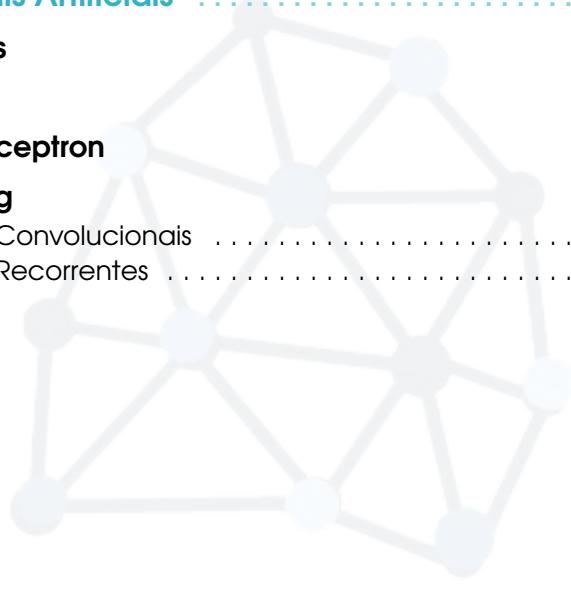
Edição: 9 de outubro de 2018



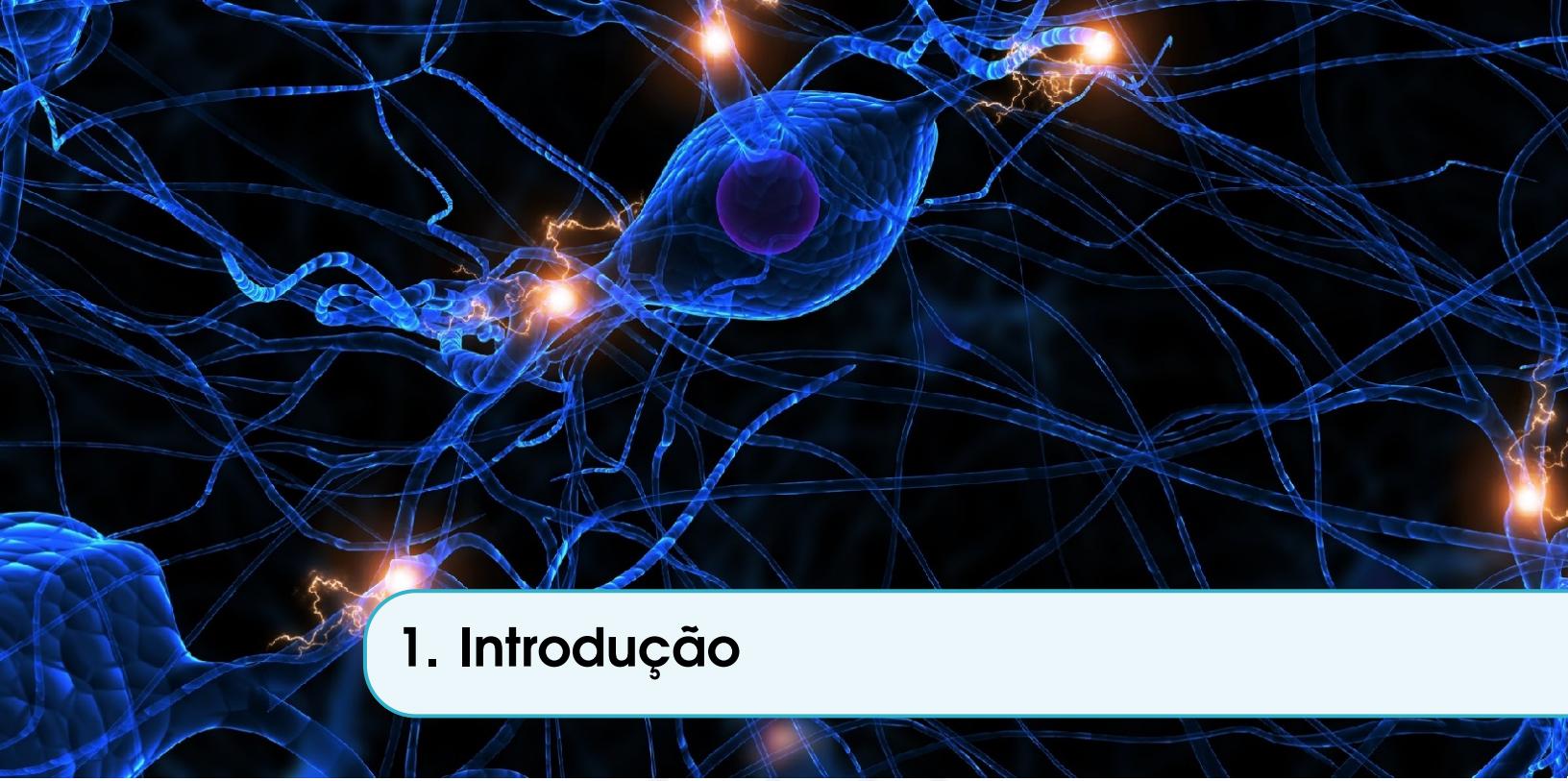
Sumário

1	Introdução	5
1.1	O grupo	5
1.2	O conteúdo	5
1.3	Os autores	6
1.4	Gestão 2018	8
2	Introdução ao Aprendizado de Máquina	9
2.1	Tipos de Aprendizado de Máquina	9
2.2	O Aprendizado Não Supervisionado	9
2.2.1	Métodos de Aprendizado Não Supervisionado	9
2.2.2	Clusterização Hierárquica	10
2.2.3	Métodos de Particionamento	13
2.2.4	<i>Principal Components Analysis - PCA</i>	15
2.2.5	Materiais complementares para Aprendizado Não Supervisionado	16
2.3	O Aprendizado Supervisionado	16
2.3.1	Introdução às Árvores de Decisão	17
2.3.2	Introdução ao Random Forest	18
3	Aprendizado de Máquina Supervisionado	21
3.1	Modelos de Regressão	21
3.1.1	Regressão Linear	21
3.1.2	Árvore de Regressão	26
3.1.3	Random Forest	27

3.1.4	Como avaliar a performance de modelos de regressão?	27
3.2	Modelos de Classificação	28
3.2.1	Árvores de Classificação	28
3.2.2	Random Forest	29
3.2.3	Naive Bayes	29
3.2.4	K-Vizinhos Próximos (KNN)	32
3.2.5	Máquina de vetores de suporte	34
3.2.6	Perceptron	35
3.2.7	Regressão Logística	37
3.2.8	Como avaliar a performance de modelos de classificação?	39
3.2.9	Materiais Complementares	40
4	Redes Neurais Artificiais	41
4.1	Redes Neurais	41
4.2	Perceptron	43
4.3	Multilayer Perceptron	44
4.4	Deep Learning	45
4.4.1	Redes Neurais Convolucionais	46
4.4.2	Redes Neurais Recorrentes	48



neuron



1. Introdução

1.1 O grupo

O grupo Neuron surge com o principal intuito de cobrir um *gap* existente na realidade dos brasileiros e até mesmo das principais universidades brasileiras: o estudo, discussão e desenvolvimentos de projetos relacionados ao *Data Science*, que engloba programação, análise de dados e o emprego das tecnologias de Aprendizado de Máquina com fins práticos ligados tanto à pesquisa quanto aos negócios. Como atividade principal, o grupo Neuron abrigará um grupo de estudos em *Data Science* a partir do ano de 2018 localizado no campus da Universidade de São Paulo (USP) em Ribeirão Preto, São Paulo. No que tange às atividades extras, traremos periodicamente profissionais e estudiosos da área para conversar com nossos membros e daremos apoio a alunos que queiram desenvolver projetos próprios. Além disso, fechamos parceria com o grupo HAIT, mais importante grupo de estudos em Inteligência Artificial formado por alunos das melhores universidades japonesas. Essa parceria possibilitará grande troca de experiência e *networking* entre os membros dos dois grupos, sendo que o desenvolvimento de projetos entre alunos de diferentes nacionalidades será o ponto alto. Espero que aproveite seu tempo no Neuron. Grande abraço,

Felipe Maia Polo - Presidente 2018

1.2 O conteúdo

O conteúdo do Neuron foi desenvolvido com a finalidade de maximizar o seu aprendizado em *Data Science*, utilizando a linguagem de programação Python com o enfoque em Aprendizado de Máquina. Os assuntos que serão abordados foram divididos em quatro grandes tópicos: Introdução a Programação em Python, Fundamentos de Matemática, *Machine Learning* e *Deep Learning*. Esses tópicos serão divididos em subtópicos, para melhor explicação dos mesmos. Os conteúdos serão abordados em aulas expositivas com exercícios práticos, que serão disponibilizados antes de cada aula. Também contaremos com palestras de profissionais das diversas áreas, com o intuito de propiciar aos membros do grupo um conhecimento amplo sobre as aplicações das tecnologia no estudos em *Data Science*. Todo o conteúdo foi feito com muito esforço e dedicação com a

colaboração do Felipe Maia Polo e do Samuel Henrique. Espero que o conteúdo seja completo e de fácil compreensão. Atenciosamente,

Leonardo Ernesto – Vice-Presidente 2018

1.3 Os autores

Este espaço é dedicado para que você conheça um pouco dos nossos autores.

Felipe Maia Polo

Estudante de Economia pela Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto da Universidade de São Paulo. Tem experiência com Estatística, Econometria e Aprendizado de Máquina. Trabalhou no Laboratório de Estudos e Pesquisas em Economia Social (LEPES) focando majoritariamente em avaliação de políticas públicas na área da educação. Concluiu parte da graduação na Universidade de Tóquio, onde teve contato com grupos voltados ao ensino e treinamento de pessoas em *Data Science*, estes que foram grandes inspirações para a concretização do Grupo de Estudos em *Data Science* Neuron/USP.

Leonardo Ernesto

Estudante de Informática Biomédica pela Faculdade de Medicina de Ribeirão Preto e pela Faculdade de Filosofia Ciências e Letras de Ribeirão Preto na Universidade de São Paulo. Exerce o cargo de diretor de assuntos acadêmicos no Centro Estudantil da Informática Biomédica - CEIB, o qual está sob sua responsabilidade desde 2016 e já foi representante Discente na Comissão Coordenadora da IBM em 2016. É aluno de iniciação científica desde 2014 no Laboratório de Tráfego Intracelular de Proteínas, onde desenvolve pesquisa de interação de proteínas. Possui amplo conhecimento nas áreas de Bioinformática, Inteligência Artificial, Reconhecimento de Padrões e *Data Mining*.

Matheus Faleiros

Formado em Informática Biomédica e mestrando do programa Interunidades em Bioengenharia da Universidade de São Paulo. Trabalha com classificação e reconhecimento de padrões em imagens médicas. Possui conhecimento em técnicas de *Machine Learning* e *Deep Learning*.

Samuel Henrique

Estudante de Informática Biomédica pela Faculdade de Medicina de Ribeirão Preto e pela Faculdade de Filosofia Ciências e Letras de Ribeirão Preto na Universidade de São Paulo. Conhecimento avançado na linguagem Python 3.6, algumas de suas bibliotecas e Programação Orientada a Objetos.

Pablo Leonardo

Graduando em engenharia de produção pela faculdade Pitágoras. Graduando em licenciatura plena em matemática pela Universidade Federal do Maranhão. Amante de neuromatemática e matemática computacional.

Murilo Henrique Soave

Estudante de Economia Empresarial e Controladoria na Faculdade de Economia, Administração e Contabilidade da Universidade de São Paulo. Possui conhecimento avançado em Python e C, além de experiência na programação de sistemas embarcados. Atualmente, desenvolve trabalho de iniciação científica vinculado ao Instituto Federal de São Paulo.

Carlos Henrique Lavieri Marcos Garcia

Estudante de Economia na Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto (FEA-RP/USP), com interesse em ciência de dados e Aprendizado de Máquina.

Mateus Padua

Bacharel em Ciência da Computação pela UniSEB COC. Trabalha como desenvolvedor WEB e Engenheiro de Software a 18 anos. Possui conhecimento avançado em Python, Django, JavaScript, OOP, Banco de dados, AWS. Sempre em busca de novos desafios.

Thiago M. Carvalho

Graduado e mestre em estatística pela Universidade de Brasília, trabalha no Banco do Brasil com a linguagem SAS. Aprendiz de R e Python, é um cientista de dados em formação. Atualmente estuda estatística bayesiana.

The logo consists of a faint, light gray watermark-like graphic. It features a central hexagonal shape formed by overlapping circles, with several smaller circles connected by lines to create a network or mesh pattern extending towards the edges. Below this geometric design, the word "neuron" is written in a large, lowercase, sans-serif font. The letters are slightly faded, giving it a watermark appearance.

1.4 Gestão 2018

Presidente
Felipe Maia Polo

Vice-Presidente
Leonardo Ernesto

Conteúdo

<i>Diretor:</i>	Samuel Henrique	<i>Membro:</i>	Matheus Faleiros
<i>Membro:</i>	Pablo Leonardo	<i>Membro:</i>	Murilo Soave
<i>Membro:</i>	Carlos Henrique	<i>Membro:</i>	Charles Chen
<i>Membro:</i>	Thiago Carvalho	<i>Membro:</i>	Bruno Comitre
<i>Membro:</i>	Mateus Padua		

Projeto

<i>Diretor:</i>	Leonardo Ernesto	<i>Membro:</i>	Eduardo Junqueira
-----------------	------------------	----------------	-------------------

Desenvolvimento de pessoas

<i>Diretor:</i>	Gustavo Ribeiro	<i>Membro:</i>	Edvaldo Santos
<i>Membro:</i>	Henrique Nogueira		

Financeiro

<i>Diretora:</i>	Natasha Freitas	<i>Membro:</i>	Beatriz Machado
<i>Membro:</i>	Murilo Soave		

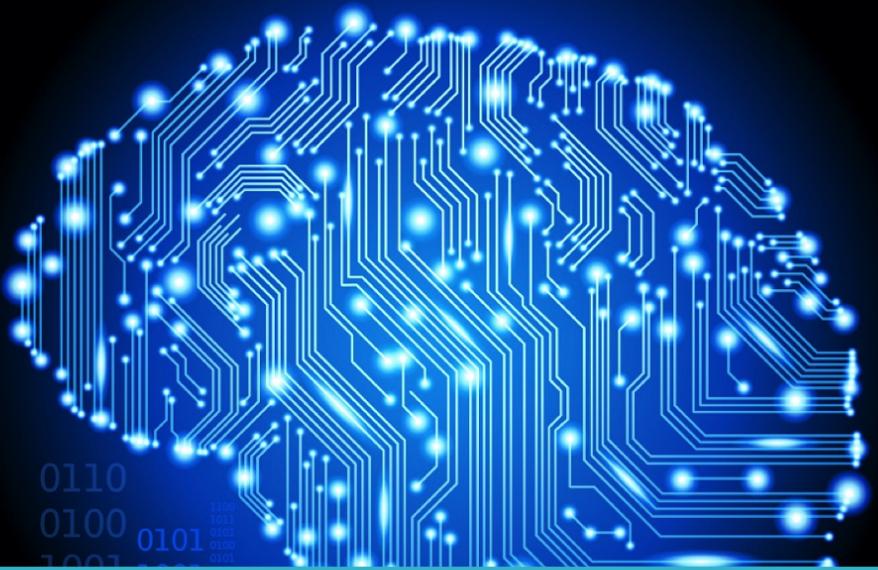
Marketing

<i>Diretor:</i>	Jonathan Batista	<i>Membro:</i>	Cassio Vilela
<i>Membro:</i>	Willy Oliveira	<i>Membro:</i>	Gustavo Santos

Relações Públicas

<i>Diretor:</i>	Felipe Maia Polo	<i>Membro:</i>	Eduardo Heitor
-----------------	------------------	----------------	----------------

neuron



2. Introdução ao Aprendizado de Máquina

1001 0100
0101 1010
1011 0101
0101 0101
0100 1001
1001 0101
0110 0100
0100 0101
1001 0101
0101 0100

0100
1001
0101
0101

1001 0100
0101 0101
0110 0100

Com o avanço da tecnologia, o poder de processamento computacional está cada vez maior e, em função disso, os computadores começaram a aprender com algoritmos a automatização de extração de conhecimentos com muito mais poder e rapidez. Hoje em dia o Aprendizado de Máquina (ou *Machine Learning*) está presente em nossos celulares e computadores, usando algoritmos que aprendem interativamente a partir de dados. A partir de agora, vamos entrar mais a fundo no aprendizado de máquina e como ele pode ser utilizado. Veremos quais são os principais métodos de aprendizado mais a frente.

2.1 Tipos de Aprendizado de Máquina

Os principais tipos de aprendizado de máquina são: aprendizagem supervisionada, não supervisionada e semi-supervisionada. A seguir observamos a característica de cada uma delas. De certa forma, todos os tipos de aprendizado são baseados em experiências passadas (dados) e infere algo sobre a realidade, tentando resolver um problema. Resumidamente, os algoritmos utilizam reconhecimento de padrões e grande quantidade de dados para obter respostas precisas. Exemplos: recomendações do Netflix, Reconhecimento de face do Facebook, carro autônomo da Google.

2.2 O Aprendizado Não Supervisionado

Neste tipo de aprendizado não existe supervisão ou “professor” que fornece os resultados corretos de saída, apenas os dados de entrada. O algoritmo busca por padrões e procura saber quando a característica ocorre e quando não ocorre, ou seja, tem liberdade de classificar como quiser. O conjunto das técnicas não supervisionadas e mais utilizadas para agrupar os dados que são semelhantes se chama *clustering*, por exemplo.

2.2.1 Métodos de Aprendizado Não Supervisionado

Os principais objetivos do aprendizado não supervisionado, são encontrar estruturas em dados não classificados, padrões inesperados e classificar os dados em grupos por alguma similaridade.

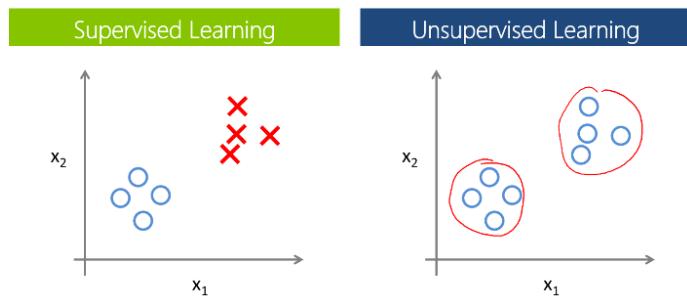


Figura 2.1: Exemplo das diferentes abordagens de aprendizado, a primeira mostra o supervisionado: onde as figuras são separadas de acordo com um atributo já conhecido; enquanto a segunda mostra o aprendizado não supervisionado, onde ocorre a classificação de acordo com características em comum. Disponível em: <https://lakshaysuri.files.wordpress.com/2017/03/sup-vs-unsup.png?w=648>

Nesse estudo explicaremos como funcionam os algoritmos de algumas técnicas de Clustering e redução de dimensionalidade, que englobam classificação atributos em grupos e a diminuição do número de variáveis usadas sem uma perda de informação relevante. Dentre esses métodos abordaremos; o Métodos de Particionamento (*Partitioning Method*), Clusterização Hierárquica (*Hierarchical Clustering*) e Análise do Componente Principal (*Principal Components Analysis – PCA*).

2.2.2 Clusterização Hierárquica

Dentro do Machine Learning são inúmeras as naturezas que um conjunto de dados a ser utilizado em um modelo pode assumir. Assim sendo, tão diverso quanto os âmagos de cada conjunto, são os métodos de agrupá-los. Afinal, características distintas demandam algoritmos - também distintos - para lograr um maior êxito nas previsões feitas pelo modelo implementado.

Por exemplo, imaginemos uma situação na qual temos informações acerca de um aglomerado de compradores e queremos dividí-los em função do seu perfil de compra. Por tratar-se de um conjunto cujo número de clusters é desconhecido, o método de Clusterização Hierárquica apresenta-se como uma excelente opção. Pois, esse método de classificação tem como um de seus resultados a construção de um dendrograma, que é um instrumento capaz de nos mostrar a relação de proximidade entre os elementos.

O algoritmo hierárquico de clusterização pode ser empregado de duas formas:

1. Modo aglomerativo: O algoritmo inicia com todos os elementos da matriz separados em grupos distintos e, a cada passo, os dois grupos mais próximos são agrupados. Este processo repete-se até que haja somente um único grupo como objeto final.
2. Modo divisivo: Aqui há o processo inverso do modo supracitado, o algoritmo começa com um único grupo e os vai dividindo em pares até que restem os objetos isolados da matriz inicial.

O modo adotado na literatura clássica – e que também adotaremos aqui – é o primeiro modo. Segue uma demonstração gráfica do funcionamento do algoritmo:

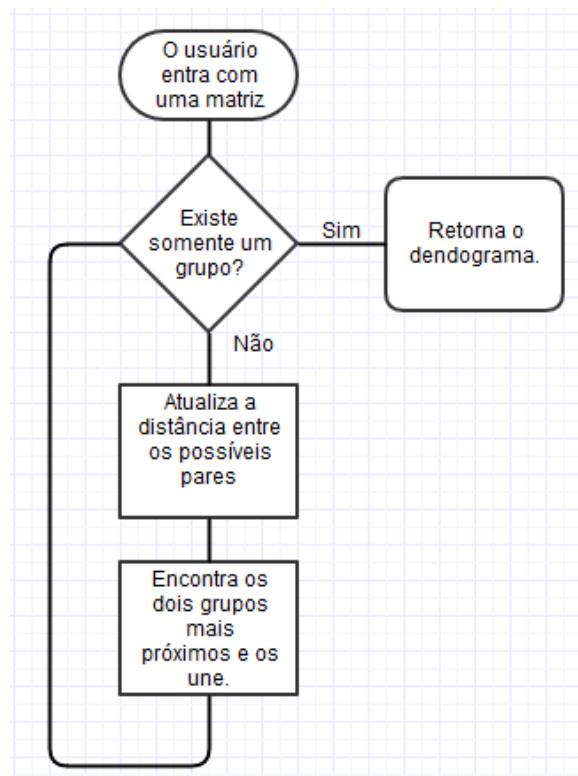


Figura 2.2: Fluxograma do algoritmo

Vale notar que o processo de encontrar os dois grupos mais próximos é feito através de um dos métodos abaixo:

- Single: A distância entre os grupos é mensurada através da distância mínima encontrada entre os objetos que os compõem.
- Complete: Diferente do primeiro método, o critério é a distância máxima entre os objetos.
- Average: Já aqui, o que é levado em consideração é a distância média entre todos os objetos.
- Centroid: É considerada a distância entre os centroides dos dois clusters.
- Ward: Caso os dois clusters selecionados unam-se, medirá como mudará a distância total em relação aos centroides originais.

Para mais detalhes sobre a teoria, ver http://www.saedsayad.com/clustering_hierarchical.htm. Agora que os fundamentos teóricos básicos foram introduzidos, vamos a um exemplo prático para fixá-los. Trabalharemos com um dataset que contém informações sobre compradores¹.

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt #Plotagem dos gráficos
import scipy.cluster.hierarchy as shc #Construção do dendrograma
from sklearn.cluster import AgglomerativeClustering #formação dos clusters
  
```

¹O download do dataset encontra-se no drive junto com o material da apostila

```
data = pd.read_csv("/home/l4b/Downloads/shopping_data.csv")
```

O dataset é composto pelas colunas CustomerID, Genre, Age, Annual Income(Mensurado em milhares de reais) e Spending Score. Destas, restringir-nos-emos as duas últimas que apresentam as inputs úteis ao nosso processo. A primeira aponta o rendimento anual e a segunda a propensão a consumir, sendo 0 o mínimo e 100 o máximo.

```
infos = data.iloc[:,3 :5].values #Seleção das duas referidas colunas.
```

```
plt.figure(figsize=(10,10)) #Dimensões do gráfico que será plotado.
```

```
link = shc.linkage(infos, method='ward') #Realiza a união dos grupos utilizando o método wa
```

```
dendrogram = shc.dendrogram(link) #Constroi o dendrograma.
```

```
plt.show() #Exibe-o.
```

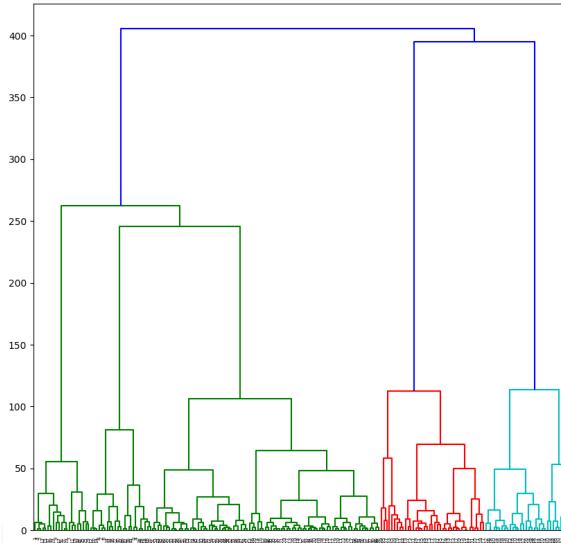


Figura 2.3: Dendrograma

Algo interessante a se notar no dendrograma é o fato do tamanho das linhas verticais representarem a distância demandada no agrupamento. Ou seja, o ideal é aguardar surgir extensas linhas para escolher o número de clusters. No caso deste dataset, é notável que isso ocorre após a formação de 5 grupos. Com o número em mãos, resta-nos partir para a clusterização.

```
cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
cluster.fit_predict(infos)
```

A função terá como retorno uma lista nos informando o grupo que cada elemento da matriz pertence, vamos, portanto, plotar o gráfico final já com essa divisão.

```
#O eixo X será representado pelo rendimento anual e o Y pela propensão a consumir.
```

```
plt.scatter(infos[:,0], infos[:,1], c=cluster.labels_, cmap='rainbow')
```

```
plt.show()
```

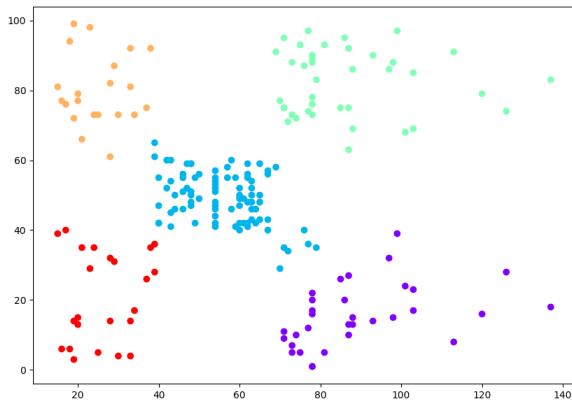


Figura 2.4: Resultado da clusterização

No agrupamento acima, é possível visualizar claramente os cinco clusters. Por exemplo, o vermelho representa indivíduos que ganham pouco e também tendem a consumir pouco, já o roxo é formado por pessoas que ganham muito, mas, consomem pouco.

2.2.3 Métodos de Particionamento

Esses métodos são baseados na separação dos indivíduos em subgrupos, sendo que uma parte importante desse tipo de algoritmo é o número de subgrupos k que serão formados e que deve ser informado pelo usuário. Para encontrar o melhor k , usamos métricas que descrevem a melhor estimativa dos grupos em relação aos dados. Dentre os métodos de particionamento, abordaremos o *k-means*.

Podemos fazer o uso do algoritmo de *K-means* quando possuímos um conjunto de dados não rotulados e gostaríamos de agrupá-los, para assim poder observar padrões nos dados. Posteriormente, poderia ocorrer uma análise mais aprofundada das características de cada grupo após o agrupamento, caso este aconteça.

Para mostrar a implementação do método, iremos abordar o mesmo exemplo utilizado na parte de clusterização hierárquica, porém desta vez em utilizaremos a idade também como variável. Abaixo as bibliotecas que usaremos.

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

Para facilitar nossa análise iremos tirar a variável Costumer ID pois ela não é relevante para a demonstração.

```
dataset.drop(["CustomerID"], axis=1, inplace= True)
```

O algoritmo do *K-means* funciona basicamente calculando a distância de um ou mais centros, os centróides, até os pontos do dataset usado para o aprendizado não supervisionado da máquina, a clusterização então é feita em relação a estes centros, ou seja, se um ponto X está mais próximo de um determinado centro falamos que este ponto pertence ao cluster que este centro representa.

Sempre ao iniciarmos o algoritmo definimos os números de centros que queremos gerar e como nosso dataset já é conhecido e sabemos que existem 5 clusters nele, então iremos inicializar com este valor. Se fossemos utilizar outros datasets os quais não conhecemos o número de clusters, teríamos que fazer uma análise para poder descobrir o números de clusters que mais se adequa ao dataset - mais a frente abordaremos como proceder nesses casos.

```
kmeans = KMeans(n_clusters = 5, init = 'k-means++')
```

Na chamada do método do *K-means* vemos também, além do número de clusters, a maneira que usaremos para gerar os centros. Existem três maneiras disponibilizadas pelo pacote:

- k-means++: Método mais utilizado, sendo que na documentação não fica muito claro seu funcionamento porém sabemos que este método facilita a convergência do algoritmo;
- random : Neste método os centros são gerados randomicamente dentro do espaço ocupado pelo dataset;
- Por último, podemos passar um 'ndarray' com os pontos onde desejamos que os centros sejam inicialiados.

A utilização dos centros no algoritmo serve para fazer o agrupamento dos dados. Centros iniciais são gerados de acordo com um dos métodos acima e a partir dele é feito um agrupamento inicial, sendo que o algoritmo usa a distância euclidiana para decidir a qual centro o ponto pertence - ele pega a menor distância. Após este primeiro agrupamento, os centróides são recalculados como uma média dos pontos pertencentes a cada um dos clusters. Os dois processos se repetem n vezes até que os centróides tenham收敛ido. É importante dizer que os centróides finais dependem dos centróides iniciais, ou seja, podemos repetir o processo algumas vezes com diferentes centros para checar a robustez do agrupamento.

Seguindo com nosso código, agora com um entendimento maior do método e com ele já inicializado em nosso código, vamos fazer com que a máquina aprenda a agrupar os pontos:

```
kmeans.fit(dataset)
```

Pronto, agora temos nosso modelo K-means treinado e pronto para uso, vamos ver se o resultado obtido pelo *k-means* é similar ao algoritmo anterior. Para isso vamos fazer uma plotagem. O código abaixo mostra uma forma diferente de manipular os dados para plotar, neste código usamos três variáveis, porém vamos plotar apenas duas para facilitar a visualização:

```
data = 0
X = np.array(dataset['Annual Income (k$)'])
Y = np.array(dataset['Spending Score (1-100)'])
X.shape =(-1,1)
Y.shape =(-1,1)
data = np.concatenate((X,Y),axis=1)
```

Agora podemos plotar.

```
plt.scatter(data[:,0], data[:,1], s = 50, c = kmeans.labels_)
plt.title('Final')
plt.xlabel('X')
```

```
plt.ylabel('Y')
plt.legend()

plt.show()
```

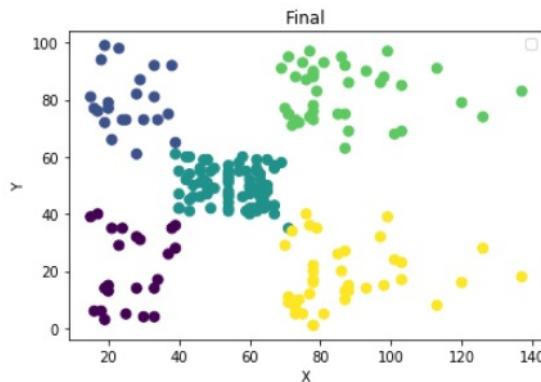


Figura 2.5: Resultado da clusterização

Podemos ver que os resultados são bem parecidos, isso nos diz que o agrupamento foi robusto ao método utilizado. O método PAM que foi falado no começo da sessão é também conhecido como k-medoids. A única diferença entre os dois modelos é que o PAM utiliza a distância Manhattan para calcular as distâncias dos pontos até os centros, que nesse caso é a mediana geométrica dos pontos e não a média (o centro é definido de modo a minimizar a soma das distâncias Manhattan de todas as observações pertencentes àquele grupo até o ponto) - isso é feito para que o modelo seja menos sensível a outliers, ou seja, pontos que estão muito "fora da curva" e que podem criar empecilhos para se chegar em um bom resultado.

2.2.4 Principal Components Analysis - PCA

O método PCA é utilizado quando há a necessidade de se reduzir a quantidade de atributos de dados numéricos enquanto tenta-se manter a estrutura dos dados. Podemos usar o PCA quando trabalhamos com análise de genes diferencialmente expressos, onde a quantidade de dados é muito grande e não queremos que ocorra a perda de informações, por exemplo. Podemos observar a figura abaixo, para uma visão mais geométrica do modelo²:

²<http://alexhwilliams.info/itsneuronalblog/2016/03/27/pca/>

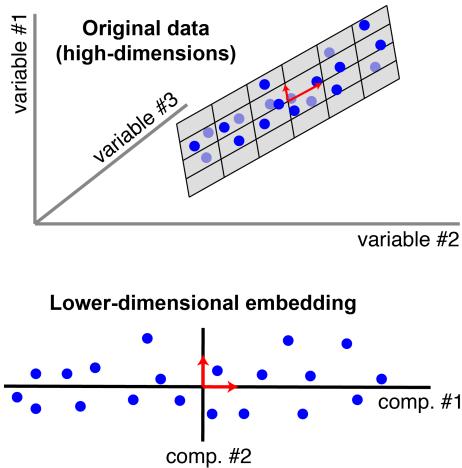


Figura 2.6: Exemplo de utilização do PCA para diminuir o número de variáveis.

Intuitivamente, o método do PCA encontra as direções que os dados têm maior variação e "reconstrói" as variáveis nessas direções. Na imagem acima, temos 3 variáveis, no entanto, percebe-se que os dados têm grande parte de suas variações em apenas duas direções. Assim sendo, podemos reconstruir as variáveis nas direções que explicam grande parte das variações nos dados. As duas novas variáveis são independentes entre si e são calculadas a parte de combinações das três originais de modo a maximizar a variância desses dados. É importante dizer que antes de se realizar o PCA, é necessário reescalar os dados para que todas as variáveis tenham a mesma escala. O número de componentes (ou variáveis novas) utilizados deve ser escolhido pelo usuário de acordo com o quanto da variação pode ser explicada pelos novos componentes em relação às variáveis originais. Mais detalhes podem ser acessados em https://pt.wikipedia.org/wiki/Análise_de_componentes_principais.

2.2.5 Materiais complementares para Aprendizado Não Supervisionado

Se você achou o material muito básico até aqui, sugiro que leia o capítulo <http://www.ee.columbia.edu/~vittorio/UnsupervisedLearning.pdf>. Outros materiais de apoio:

- <https://medium.com/machine-learning-for-humans/unsupervised-learning-f45587588294>
- https://www.youtube.com/playlist?list=PL2Wzg8U7YXS9HjwRCy_1kcAMHLFUoV9tu
- <http://stanford.edu/~cziech/cs221/handouts/kmeans.html>
- <https://www.cs.utah.edu/~piyush/teaching/4-10-print.pdf>
- http://aquilesburlamaqui.wdfiles.com/local--files/logica-aplicada-a-computacao/texto_fuzzy.pdf
- <https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis/>
- <https://www.toptal.com/machine-learning/clustering-algorithms>
- <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-19082004-092311/>.

2.3 O Aprendizado Supervisionado

Neste tipo de aprendizagem o algoritmo observa inúmeros exemplos de entrada, ou seja, observa o histórico ou característica dominante desses dados e cria um modelo que prevê sua saída, como se existisse um professor para ensiná-la. A máquina precisa ter muitos exemplos e utilizar os critérios e

informações corretas para fazer boas previsões com os novos dados. Quando falamos de aprendizado supervisionado, temos dois tipos principais de objetivos: (i) prever resultados com significados quantitativos, que dá origem ao problema de **regressão**, e (ii) prever resultados com significados qualitativos, ou seja, de se classificar um novo indivíduo em classes pré-existentes nos dados, que dá origem ao problema da **classificação**. Dois métodos que podem ser usados tanto para resolver problemas de regressão quanto de classificação são as Árvores de Decisão e Floresta Aleatória (Random Forest), por exemplo.

2.3.1 Introdução às Árvores de Decisão

Árvores de Decisão para Classificação ou Regressão são métodos que particionam os dados e realizam previsões condicionais às partições geradas. Tal contexto é empregado dentro da Mineração de Dados - *Data Mining*. Nas Árvores de Decisão, o primeiro nó da árvore é sempre o atributo mais importante, e os menos importantes são mostrados nos outros nós. Ao escolher e apresentar os atributos em ordem de importância, as Árvores de Decisão permitem aos usuários conhecer quais fatores são os mais importantes em seus trabalhos³. A vantagem principal das Árvores de Decisão é a tomada de decisões levando em consideração os atributos mais relevantes, além de compreensíveis para a maioria das pessoas⁴.

O computador não tem a capacidade de reconhecer padrões óbvios, como um ser humano. Esta situação tem gerado demandas por novas técnicas e ferramentas que, com eficiência, transformem os dados armazenados e processados em conhecimento⁵. As Árvores de Decisão são representações simples do conhecimento e um meio eficiente de construir classificadores e regressores que predizem classes/atributo quantitativo baseados nos valores de atributos de um conjunto de dados⁶. Uma Árvore de Decisão utiliza a estratégia chamada Dividir Para Conquistar, ou seja, um problema complexo é decomposto em subproblemas mais simples. Recursivamente, a mesma estratégia é aplicada a cada subproblema⁷.

Uma árvore é composta por nós, sendo que os nós podem representar tanto variáveis quantitativas quanto qualitativas. Ramos são as subseções da árvore. Nó interno (ou nó de decisão) é um nó cujo propósito é direcionar o objeto para o caminho apropriado por comparação de valores. Nó terminal (também chamado de nó folha) é o nó que não possui “filhos”, sendo o último nó do segmento. O nó terminal, quando usamos árvores de decisão/regressão, indica o valor ou classe que será atribuída ao objeto que percorrer aquele caminho. Abaixo é demonstrado um exemplo da estrutura da árvore⁸.

Existem vários algoritmos para criar uma árvore de decisão, sendo os mais comuns o ID3, CART, C4.5, entre outros. A biblioteca scikit-learn que estamos usando para os métodos de *machine learning* implementa uma versão da árvore CART na API. Para entender melhor como funciona uma árvore de decisão ou regressão veja o seguinte vídeo <https://www.youtube.com/watch?v=gYSWrUP4aB0>.

³Análisis de crédito bancario por medio de redes neuronales y árboles de decisión: una. Revista de Administração da Universidade de São Paulo, p225 - 234.

⁴http://www.coc.ufrj.br/teses/doutorado/inter/2005/Teses/CARVALHO_DR_05_t_D_int.pdf

⁵Data Mining and Knowledge Discovery: A Review of Issues and Multistrategy Approach. In: R. Michalski, I. Bratko, and M. Kubat, Machine Learning and Data Mining: Methods and Applications (pp. p71-112). London: John Wiley and Sons.

⁶O uso de árvores de decisão na descoberta de conhecimento na área da saúde. In: SEMANA ACADÊMICA, 2000. Rio Grande do Sul: Universidade Federal do Rio Grande do Sul, 2000.

⁷<http://www.liacc.up.pt/~jgama/Mestrado/ECD1/Arvores.html>

⁸<https://dx.doi.org/10.5935/0103-104.20140016>

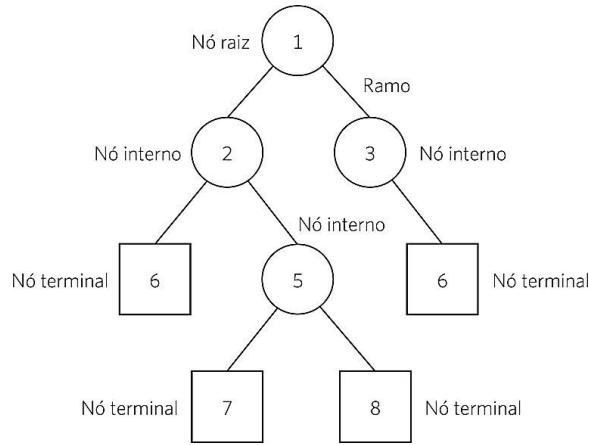


Figura 2.7: Representação da estrutura de uma árvore

2.3.2 Introdução ao Random Forest

Random Forest ou floresta aleatória é caracterizado por sua fácil utilização, flexibilidade e por conseguir trabalhar com poucos dados. Pode, assim como a árvore de decisão/regressão, ser usada para a regressão e para a classificação⁹.

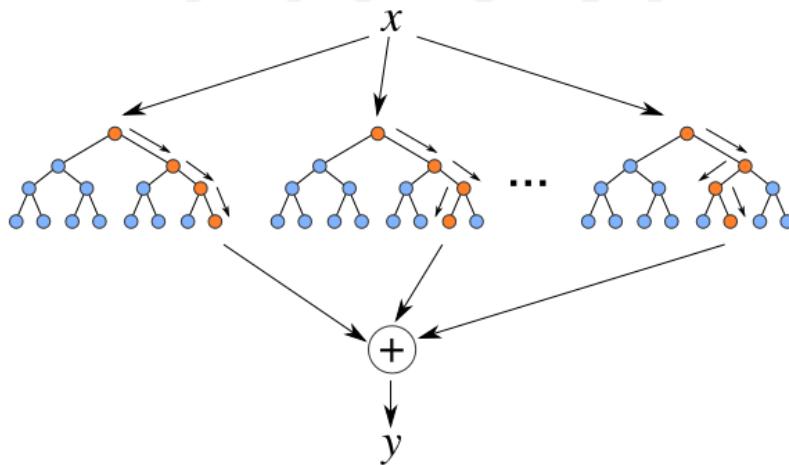


Figura 2.8: Representação de uma floresta aleatória com diversas árvores

O algoritmo de Random Forest consiste-se na criação e combinação de várias árvores de decisão para obter uma predição mais estável e com maior precisão¹⁰. Ou seja, o algoritmo tenta encontrar as melhores características em um subconjunto de características em uma grande diversidade de dados, o que resulta em um modelo melhor.

O modelo de Random Forest consiste-se basicamente nos seguintes passos¹¹: (i) definir o número de árvores que haverá na floresta; (ii) para cada uma das árvores aplicar uma amostragem com

⁹https://cdn-images-1.medium.com/max/800/0*tG-IWcxL1jg7RkT0.png

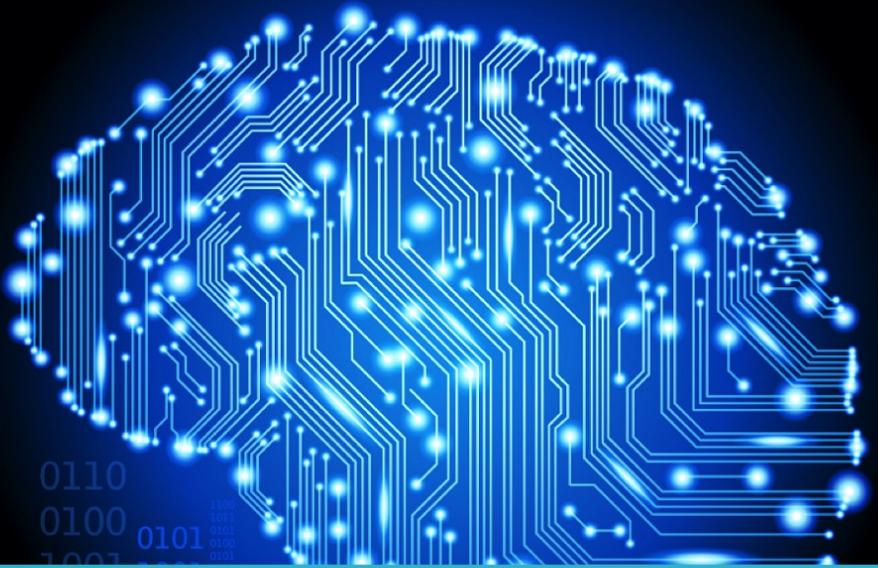
¹⁰<https://medium.com/machina-sapiens/o-algoritmo-da-floresta-aleat%C3%B3ria-3545f6babdf8>

¹¹https://www.youtube.com/watch?v=D_2LkhMJcfY

reposição do tamanho da sua base de treinamento (processo *bootstrapping*); (iii) escolher um número m menor ou igual ao número de variáveis explicativas; (iv) para cada uma das árvores escolher m (*randomizing*) variáveis explicativas de forma aleatória para cada uma das árvores; (v) fazer as árvores crescerem até uma profundidade pré-definida ou até quando não der mais; (vi) levando em consideração todas as previsões de um indivíduo j , realizar uma votação (pela maioria) no caso da classificação ou uma média na caso da regressão para obter a previsão final.

Como visto anteriormente, as técnicas de *Machine Learning* são divididas em aprendizado supervisionado e não supervisionado. Dentro do aprendizado supervisionado podemos subdividir as abordagens em Classificação e Regressão.





3. Aprendizado de Máquina Supervisionado

1001 0100
0101 1011
0101 0101
1001 0100
0101 0101
0101 0100

3.1 Modelos de Regressão

O problema de regressão na campo do Aprendizado de Máquina comporta diversas empreitadas nas quais os objetivos compreendem o cálculo de um valor de saída com qualidade quantitativa. Assim sendo, problemas de regressão no campo do Aprendizado de Máquina comportam situações como por exemplo de predição de preços de imóveis, salários de indivíduos, a quantidade de chuva em um determinado dia (em mm) ou mesmo o peso das pessoas dadas algumas pistas de seus estilos de vida. Existem diversos métodos para a resolução de problemas de regressão e não existe um que sempre se sobressairá sobre os outros, portanto é muito importante sempre fazer uma comparação dos resultados obtidos por diversos modelos.

3.1.1 Regressão Linear

A Regressão Linear é um modelo muito conhecido e usado no mundo da Estatística e do Machine Learning, pela sua simplicidade, facilidade de interpretação e poder preditivo. Usamos modelos de Regressão Linear quando queremos resolver problemas de regressão, ou seja, quando temos um conjunto de dados dotados de características e queremos prever que número real está associado àquelas características. Por exemplo, podemos querer prever qual o salário de uma pessoa dadas suas características pessoais e características do ambiente no qual as pessoas estão inseridas.

Especificação matemática

Os modelos padrão de regressão linear geralmente são definidos da seguinte maneira:

$$y_i = b + w_1x_{1i} + w_2x_{2i} + \dots + w_kx_{ki} + \varepsilon_i = b + \mathbf{x}_i^T \mathbf{w} = \hat{y}_i + \varepsilon_i \quad (3.1)$$

Na equação acima, y_i é comumente conhecida como variável explicada (poderia ser o salário, por exemplo) do indivíduo i ; o conjunto $\{x_{ji}\}$ é o conjunto de características observáveis do indivíduo i , que é conhecido como conjunto de variáveis explicativas; o b é o que chamamos de viés ou *bias*, que

é também conhecido como coeficiente linear ou intercepto¹; por fim, \hat{y}_i é a previsão de y_i feita pelo modelo e ε_i é o erro cometido pelo modelo, ou seja, a parte não explicada pelos dados.

A utilização de variáveis categóricas como explicativas no modelo

A inclusão de variáveis contínuas (e.g. altura, idade) no modelo é direta, ou seja, não há a necessidade de mudar sua forma antes de incluí-las na equação a ser estimada. No entanto, quando queremos incluir variáveis categóricas (e.g. nível de escolaridade, gênero) como explicativas é necessário fazer algumas mudanças. Por exemplo, suponha que temos a variável *python*, que mede a habilidade das pessoas na programação em Python, com as seguintes categorias "Iniciante", "Intermediário" e "Avançado". Sendo que "Iniciante" é representado pelo número 0, "Intermediário" é representado pelo número 1 e "Avançado" é representado pelo número 2. O padrão a ser seguido é: criar 3 novas variáveis (uma para cada categoria), sendo que estas assumem 0 ou 1, dependendo das respostas, sendo que 0 é negativo e 1 positivo para tal categoria. Após a criação das variáveis, deve-se excluir uma delas² e incluir as restantes no equação a ser estimada.

A interpretação dos parâmetros estimados

Suponha que você tenha um conjunto de indivíduos na sua base de dados os quais não têm seus salários expostos. Então você gostaria de predizer quais seriam os salários dessas pessoas com base nas variáveis disponíveis em sua base de dados: você tem acesso aos anos de experiência e o nível de inglês das pessoas, sendo que essa última variável tem as categorias "Básico" ou 0, "Intermediário" ou 1 e "Avançado" ou 2. Pelo o que foi explicado anteriormente, pelo fato de a variável experiência ser contínua, podemos incluí-la diretamente na equação. Em relação à outra variável, é necessário transformá-la antes de colocá-la na equação. Vamos então transformá-la em 3 variáveis binárias e excluir uma delas. No caso, optei por excluir a variável relativa à categoria "Iniciante" e a equação estimada foi:

$$\widehat{\text{salar}}_i = 100 + 15 * \text{exp}_i + 90 * \text{interm}_i + 150 * \text{avanc}_i \quad (3.2)$$

Vamos primeiramente começar com o *b*, que nesse caso assumiu o valor 100. Nossa modelo estimado está nos dizendo que se todas as variáveis forem nulas, ou seja, a pessoa não tem experiência e tem um nível iniciante de inglês, seu salário predito é 100. Os parâmetros estimados também me dizem que a cada ano adicional de experiência, é esperado que os indivíduos tenham 15 de acréscimo em seu salário e que, em relação às pessoas que tem nível iniciante de inglês, ter nível intermediário aumenta o salário esperado em 90 e ter nível avançado de inglês aumenta o salário em 150. Se por um acaso omitíssemos a variável *interm* e não a que denota o nível iniciante, a equação estimada seria:

$$\widehat{\text{salar}}_i = 190 + 15 * \text{exp}_i - 90 * \text{inic}_i + 60 * \text{avanc}_i \quad (3.3)$$

Isso acontece porque agora mudou-se o benchmark: como a categoria de inglês intermediário foi omitida dessa vez, os outros parâmetros devem mudar para se adaptarem à essa mudança. Se

¹É uma constante geralmente compartilhada por diversos indivíduos da base de dados e que, no contexto do Machine Learning, aumenta o poder de predição do modelo.

²Para evitar o problema da Multicolinearidade: <https://en.wikipedia.org/wiki/Multicollinearity>, que causa instabilidade nas previsões.

a pessoa tiver inglês iniciante, espera-se que ela ganhe 90 a menos que as pessoas que têm inglês intermediário e se a pessoa tiver inglês avançado, espera-se que ela ganhe 60 a mais que as pessoas que têm inglês intermediário. Observe que a mesma pessoa, com as mesmas características, tem o mesmo salário predito pelas duas equações.

Estimando os parâmetros

A estimação

Para a estimação dos parâmetros é necessário voltarmos ao conceito do Gradiente Descendente ou *Gradient Descent*. Primeiramente é necessário definir nossa função perda ou *loss function*, que nesse caso é o Erro Quadrático Médio cometido pelo modelo, e depois minimizá-la. O Erro Quadrático Médio é definido pela equação:

$$\text{EQM} = \frac{1}{N} \sum_{i=1}^N \varepsilon_i^2 = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \frac{1}{N} \sum_{i=1}^N (y_i - b - \mathbf{x}_i^T \mathbf{w})^2 \quad (3.4)$$

Na equação acima, os símbolos têm o mesmo significado já comentados. Na teoria, seria necessário realizar o cálculo de $\frac{\partial \text{EQM}}{\partial b}$ e $\nabla_w \text{EQM}$ e depois montar o algoritmo do Gradiente Descendente. Na prática, essas operações e algoritmos já estão implementadas nas bibliotecas do Python, então não é necessário se prender a esse problema (a não ser que você tenha vontade de entender a fundo como as coisas funcionam).

Procedimentos

Os procedimentos da estimação de um modelo de regressão linear se assemelha aos procedimentos que tivemos quando trabalhamos com o KNN. Primeiramente, separamos nossa base de dados em 3 subconjuntos: o conjunto de treinamento, o de validação e o de teste. A estimação dos parâmetros é conduzida no conjunto de **teste**, sendo que podemos estimar mais de um modelo (e.g. tirando ou colocando variáveis). Após o processo de estimação, os diversos modelos estimados são comparados pelo seu erro cometido, ou seja, o EQM. Quando menor o EQM no conjunto de validação, melhor. Então basta escolher o modelo que comete o menor erro e testá-lo no conjunto de teste.

O modelo é bom?

Após a realização do teste do modelo, é possível usar a métrica do R^2 para medir o quanto bom o meu modelo é explicando a variável de interesse. O R^2 é melhor interpretável do que o EQM, por isso geralmente o usamos na hora de reportar um resultado. A medida R^2 é definida da seguinte maneira:

$$R^2 = \frac{\text{Var}(\hat{\mathbf{y}})}{\text{Var}(\mathbf{y})} = \frac{\text{Var}(\hat{y}_1, \hat{y}_2, \dots, \hat{y}_N)}{\text{Var}(y_1, y_2, \dots, y_N)} \quad (3.5)$$

O R^2 varia de 0 a 1, sendo que se $R^2 = 0$, o modelo consegue explicar zero da variação dos dados reais e se $R^2 = 1$, toda a variação dos dados é explicada pelo modelo - o modelo tem o máximo de aderência aos dados.

O problema do *Overfitting*

Vamos supor que você esteja treinando seu modelo com os dados referentes ao conjunto de teste. Se você tornar seu modelo mais flexível e complexo (e.g. adicionando mais variáveis explicativas), muito provavelmente o erro cometido pelo seu modelo no conjunto de treinamento vai cair e o seu R^2

no mesmo conjunto vai se aproximar de 1. No entanto, isso não é necessariamente verdade quando testamos o modelo no conjunto de dados que não foi utilizado para o treinamento. Inicialmente, isso até pode ocorrer, mas conforme vamos aumentando a complexidade do modelo, corremos o perigo de cometer o que chamamos de *overfitting*, ou seja, seu modelo tem uma aderência tão boa no conjunto de treinamento que é incapaz de fazer generalizações. Em outras palavras, o modelo "decora" os dados do conjunto de treinamento e é incapaz de reproduzir um bom resultado em dados que ainda não foram vistos.

Vamos a um exemplo mais visual: suponha que queremos treinar dois modelos da forma $y = b + w_1x + w_2x^2 + \dots + w_kx^k$, sendo que no primeiro $k = 1$ e no segundo $k = 9$, para prever o nível de vendas de automóveis no Brasil no trimestre $t + 1$ em função do PIB no trimestre t . Na imagem³, o primeiro modelo é dado pela reta preta e o segundo é dado pela curva polinomial azul. É evidente que o polinômio de alto grau é muito mais flexível e tem um erro muito menor do que o de menor grau, no entanto, isso vale enquanto estamos falando do conjunto de teste. É possível ver que o segundo modelo "decorou" os dados e provavelmente terá um resultado insatisfatório quando testado em outra base de dados.

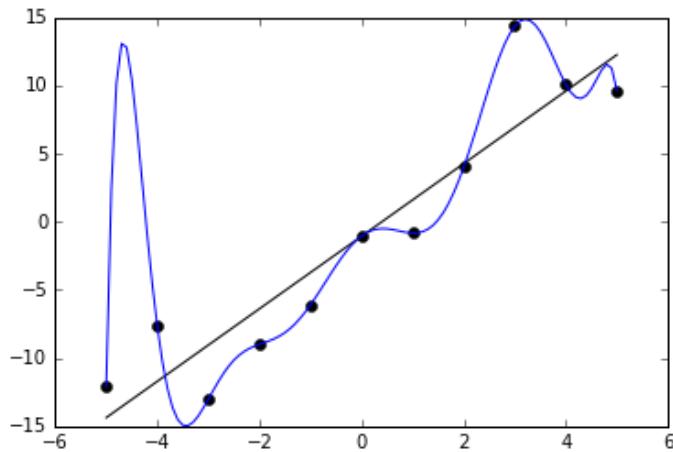


Figura 3.1: Venda de automóveis ($t+1$) Vs. PIB (t)

Como resolver o problema

Existem duas maneiras principais de resolver o problema do *Overfitting* nesse caso: (i) podemos interromper o treinamento do modelo quando percebermos que os erros no conjunto de treino e no conjunto de validação estão descolando muito ou (ii) diminuir a complexidade do modelo. Para ficar mais clara essa ideia de interromper o treinamento quando há o descolamento, preste atenção na seguinte imagem⁴. No eixo x temos o número de passos dados pelo algoritmo do Gradiente Descendente e no eixo y o erro cometido pelo modelo nos conjuntos de teste (azul) e de validação (vermelho). É possível ver que em certo ponto é necessário parar o processo de treinamento para que o erro no conjunto de validação não comece a subir. Ficará a cargo do leitor pesquisar sobre técnicas para interromper o treinamento (*early stopping*).

Como dito, há outra maneira de conter o problema do *overfitting*, que é baixando a complexidade

³<https://en.wikipedia.org/wiki/Overfitting>

⁴<https://en.wikipedia.org/wiki/Overfitting>

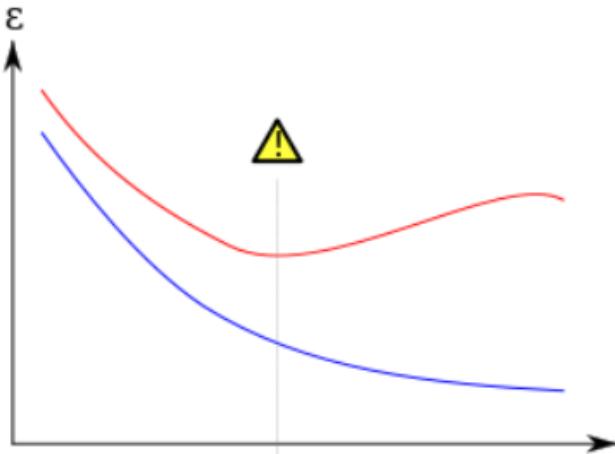


Figura 3.2: Venda de automóveis (t+1) Vs. PIB (t)

do modelo. Há basicamente duas maneiras de se fazer isso: (i) retirar variáveis do modelo ou (ii) aplicar Regularização no modelo. Vamos focar na segunda maneira. A Regularização é quando introduzimos novas informações ao processo de estimação dos parâmetros. Por exemplo, quando estamos estimando os parâmetros de um modelo de regressão linear, temos que a função perda é dada por $L = \text{EQM} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$, que é o erro quadrático médio cometido pelo modelo. Se mudarmos a função perda para

$$L = \text{EQM} + \lambda \sum_{j=1}^k w_j^2 \quad (3.6)$$

Percebemos que estamos penalizando o modelo (aumentando o erro) pela magnitude dos parâmetros e isso força o modelo a jogar os parâmetros que tem menor importância para zero, tornando o modelo mais simples. Esse método se chama *Ridge Regression*, ou seja, Regressão Rígida. A Regressão Rígida é um caso particular da *Elastic Net Regression*, que faz uma combinação entre os regularizadores da *Ridge Regression* e a *LASSO*, que inclui outro regularizador. A função perda da *LASSO* seria:

$$L = \text{EQM} + \lambda \sum_{j=1}^k |w_j| \quad (3.7)$$

E a da *Elastic Net* é:

$$L = \text{EQM} + \lambda_1 \sum_{j=1}^k |w_j| + \lambda_2 \sum_{j=1}^k w_j^2. \quad (3.8)$$

É importante dizer que as constantes λ , λ_1 e λ_2 são hiperparâmetros e devem ser definidos no processo de validação do modelo testando diversos valores. O material encontrado no link <https://www.slideshare.net/ShangxuanZhang/ridge-regression-lasso-and-elastic-net> aprofunda um pouco nos modelos apresentados, apresentando os prós e contras.

3.1.2 Árvore de Regressão

Quando árvores de decisão são utilizadas para técnicas de regressão podem ser chamadas de árvores de regressão. Sua utilização é feita quando temos as variáveis dependentes/explicadas quantitativas. Em contrapartida, as árvores de decisão são utilizadas quando as variáveis dependentes são categóricas⁵. Abaixo é possível ver uma árvore de regressão a qual tem a variável dependente/explicada quantitativa (mm de chuva).

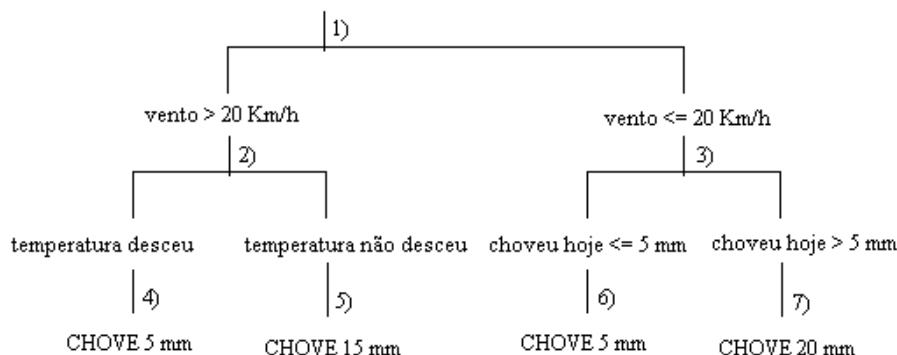


Figura 3.3: Representação da estrutura de uma árvore de regressão

Na imagem temos um exemplo de árvore de regressão que prevê quantos milímetros de chuva haverá em um dia em função dos dados anteriores. Podemos usar a árvore de regressão ao invés de regressão linear quando os dados apresentarem uma alta não linearidade e existir uma relação complexa entre as variáveis explicativas, sendo que os modelos em árvore são simples para compreensão e explicação.

Como já vimos anteriormente, as variáveis que aparecem mais acima são as mais importantes em poder explicativo, ou seja, a variável vento tem um poder explicativo maior que a variável temperatura. É importante dizer que os pontos de corte (no caso do vento o único ponto de corte é 20Km/h) também são calculados internamente pelo modelo de forma a minimizar o erro cometido pelo modelo na base de treinamento.

Uma outra coisa importante a ser evidenciada é como a variável explicada é prevista: como já dito, o modelo de árvore partitiona a base de dados de acordo com valores das variáveis explicativas criando "caixinhas" de indivíduos. No caso acima, temos quatro "caixinhas", pois há 4 nó folhas na árvore. Se um indivíduo "cai" em uma determinada folha, digamos no nó folha 6, a variável explicada predita daquele indivíduo será a média dos valores das variáveis explicativas que estão na "caixinha" do nó 6 e fazem parte da base de treino. No exemplo acima, se um indivíduo cai na terceira partição (nó 6), a valor previsto será 5mm.

Na figura abaixo, apesar de se tratar de um problema de classificação, é possível ver a árvore formada a partir das variáveis X_1 e X_2 . Note que cada nó folha da árvore corresponde a uma partição da base de dados, ou seja, uma "caixinha"⁶:

⁵<https://www.fep.up.pt/docentes/pcosme/Pedagogico/Arvores/20de/20Regressao.doc>

⁶<https://www.r-bloggers.com/regression-tree-using-ginis-index/>

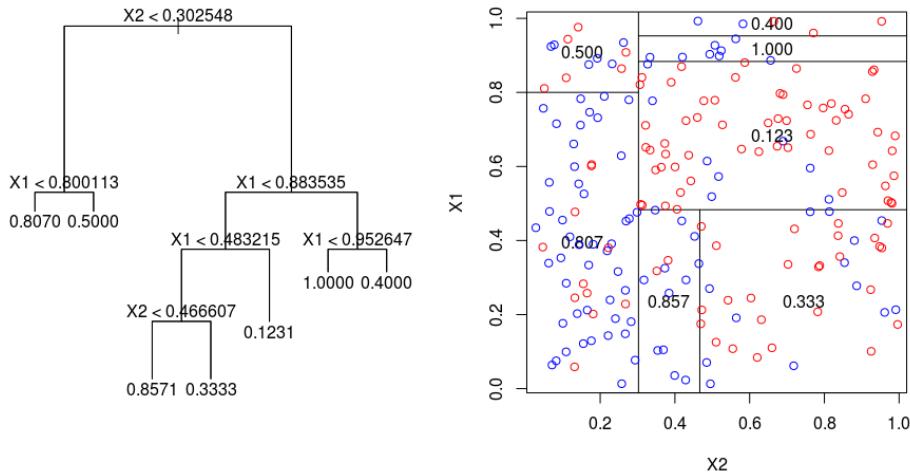


Figura 3.4: Árvore a particionamento da base de dados

3.1.3 Random Forest

Usado para as mesmas situações da árvore de regressão, ou seja, quando os dados das variáveis dependentes são contínuas, e principalmente quando temos um conjunto pequeno de amostras. São utilizadas técnicas computacionais para garantir os melhores resultados. Um bom exemplo para utilização para a realização da regressão, quando gostaríamos de inferir resultados futuros, por exemplo: o quanto o valor de uma determinada ação variará no instante seguinte. O método será abordado com maior detalhamento no Jupyter Notebook de Regressões.

O método de Random Forest para regressão tem um funcionamento básico já descrito nos passos (i) ao (vi). Vamos dizer que nossa floresta tem 100 árvores, após a predição do valor de uma variável y para um indivíduo j nas 100 árvores, realizamos uma média desses valores e assim temos nossa previsão final. O momento de Random Forest tende a dar um resultado melhor do que árvores individuais pois é capaz de reduzir a variância do estimador de y .

3.1.4 Como avaliar a performance de modelos de regressão?

Após a realização das validações ou testes do modelos, é necessário aplicar certas métricas para avaliar o poder preditivo dos métodos de regressão. Para realizar a avaliação dos métodos de regressão existem basicamente duas maneiras clássicas: (i) realizar o cálculo do Erro Quadrático Médio (quanto menos melhor) cometido pelo modelo ou (ii) utilizar a métrica do R^2 (quanto mais melhor) para se ter noção da porcentagem das variações nos dados explicada pelo método em questão. Abaixo a formalização do Erro Quadrático Médio cometido pelo modelo, ou seja, quanto menor for o EQM no conjunto de dados X , melhor o modelo em fazer previsões no conjunto de dados X :

$$\text{EQM} = \frac{1}{N} \sum_{i=1}^N \varepsilon_i^2 = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (3.9)$$

Na fórmula acima, y_i é o resultado real do indivíduo i e \hat{y}_i é o resultado previsto para o indivíduo i . O R^2 é melhor interpretável do que o EQM, por isso geralmente o usamos na hora de reportar um

resultado. A medida R^2 é definida da seguinte maneira:

$$R^2 = \frac{Var(\hat{\mathbf{y}})}{Var(\mathbf{y})} = \frac{Var(\hat{y}_1, \hat{y}_2, \dots, \hat{y}_N)}{Var(y_1, y_2, \dots, y_N)} \quad (3.10)$$

O R^2 varia de 0 a 1, sendo que se $R^2 = 0$, o modelo consegue explicar zero da variação dos dados reais e se $R^2 = 1$, toda a variação dos dados é explicada pelo modelo - o modelo tem o máximo de aderência aos dados.

3.2 Modelos de Classificação

Ainda dentro de Aprendizado Supervisionado temos o problema de classificação. Nesse contexto, os valores de entrada podem ser tanto quantitativos (discretos ou contínuas) quanto qualitativos assim como no problema de regressão, porém os valores de saída são sempre qualitativos/categóricos (e.g. 0 ou 1, "spam" ou "não spam", etc). Esses valores de saída também são chamados de classes, rótulos ou *labels*. O leque de métodos existentes para classificação é extenso, variando de como árvores de decisão até Máquina de Vetores de Suporte e Regressão Logística.

3.2.1 Árvores de Classificação

Uma árvore de decisão busca separar os dados calculando a relevância dos atributos e gerando regras de decisão para cada uma de suas divisões - *split*. Um modelo de árvore para classificação é mais intuitivo que sua aplicação para regressão: cada nó intermediário contém uma determinada regra de decisão enquanto cada nó folha contém um desfecho, ou seja, a classe de determinada entrada após satisfazer um conjunto de regras.

Para a realização de um *split* otimizado não buscamos mais minimizar o Erro Quadrático Médio, pois agora não precisamos saber o quão distante nossa predição está do valor real e sim saber quantas vezes acertamos ou não a classe real. Para isso, vários autores propuseram implementações de árvores de decisão utilizando métodos que acharam mais adequados para calcular a relevância dos atributos. Breiman⁷, que propôs a árvore de decisão CART - *Classification and Regression Tree*, utiliza o índice Gini, que mede a impureza de determinado conjunto de dados. Em contraste, o autor da árvore de decisão C4.5 Ross Quinlan⁸, propôs o uso do método ganho de informação, ou entropia, que reflete a aleatoriedade das partições formadas (para detalhes matemáticos dos dois métodos veja⁹ e¹⁰).

Outro fator a se atentar com árvores de decisões é o tamanho da árvore. Para ajustar isso existem técnicas de poda, ou seja, técnicas que definem quantos nós intermediários devem preceder um nó folha. Isso pode vir a ser importante pois um modelo muito grande de árvore pode causar sobreajuste - *overfitting* nos dados-, enquanto uma árvore muito pequena pode gerar regras muito generalizadas, reduzindo a capacidade preditiva do modelo.

As vantagens de um modelo de árvore para classificação é a sua fácil implementação e interpretação, sendo um dos poucos modelos onde podemos saber quais as decisões tomadas para predizer a classe de um valor de entrada. No entanto, este é um método sensível a ruídos (aleatoriedade que

⁷Breiman, Leo, Jerome Friedman, R. Olshen and C. Stone (1984). Classification and Regression Trees. Belmont, California: Wadsworth.

⁸C4.5: Programs for Machine Learning, Ross Quinlan, 1993

⁹https://en.wikipedia.org/wiki/Decision_tree_learning

¹⁰<http://each.uspnet.usp.br/sarajane/wp-content/uploads/2015/03/aula04.pdf>

atrapalham o aprendizado), desbalanceamento de classes e raramente convergem para ótimos globais¹¹.

3.2.2 Random Forest

Vimos também de forma geral o funcionamento de uma Floresta Aleatória. Novamente, o processo de predição de uma floresta aleatória é mais intuitivo para o problema de classificação do que para o problema de regressão: As N árvores geradas pelo modelo fazem suas classificações e, sem seguida duas abordagens podem ser utilizadas: (i) votação¹², onde escolhemos para a predição final a classe mais prevista entre as árvores, ou (ii) a média¹³, onde uma média das predições é calculada e a classe escolhida é aquela cujo o valor mais se aproxime dessa média.

O método de Floresta Aleatória é vantajoso em relação às árvores pois reduz a sensibilidade aos ruídos e a *outliers*. No entanto, seu tempo computacional é diretamente proporcional ao número de árvores e elas nem sempre possuem regras de decisão de forma que as tornem boas para predizer uma determinada classe.

3.2.3 Naive Bayes

Vamos falar um pouco sobre um algoritmo supervisionado de classificação que é dos mais simples, mas também dos mais utilizados. Estamos falando do algoritmo **Naive Bayes**.

Vamos relembrar rapidamente o que é um algoritmo supervisionado de classificação.

Um algoritmo de aprendizado supervisionado é aquele que conta com um conjunto de preditores e variável resposta que são utilizados no treinamento de um modelo, isto é, tem sua utilidade quando os 'resultados' estão disponíveis para um dado conjunto de 'variáveis explicativas' do problema a ser solucionado. Esses 'resultados' são importantes no momento em que o modelo está aprendendo como trabalhar com os dados. E o modelo é de classificação quando seu objetivo é separar os elementos em classes, binárias ou não.

Pra ficar mais claro essa questão de aprendizado supervisionado e modelo de classificação, imagine o seguinte exemplo:

Exemplo

Suponha que Thiago, um jovem ávido por andar de skate, mora longe da pista de skate que mais gosta. Além disso, a pista não é coberta. Portanto, nos dias de chuva ele não pode andar. Nos dias nublados, as vezes, fica receoso de chegar ao skate park e estar chovendo. Entretanto, conseguimos seguir os passos de Thiago por alguns dias para entender um pouco sobre sua rotina e seus dilemas para andar de skate dadas as condições do tempo na região próxima de sua casa. Os dados coletados são mostrados logo abaixo, seguidos do resultado, isto é, se Thiago conseguiu ou não andar de skate naqueles dias.

O exemplo abaixo é uma clara situação onde poderíamos empregar um algoritmo de classificação, pois o que queremos é poder responder a pergunta: **Thiago andou ou não de skate em um dado dia?** Para respondermos a essa pergunta temos que avaliar as condições de tempo e informá-las ao nosso modelo, para que este possa treinar (aprender como relacionar as condições do tempo e o resultado, a partir dos dados da tabela acima) e então conseguir responder a novas questões baseado apenas nas características oferecidas como informação. Portanto, a tarefa do modelo é classificar um

¹¹<http://scikit-learn.org/stable/modules/tree.html>

¹²L. Breiman, "Random Forests", Machine Learning, 45(1), 5-32, 2001.

¹³<http://scikit-learn.org/stable/modules/ensemble.html>

registro (com as diversas informações sobre tempo) quanto à duas classes: 'andou' ou 'não andou' de skate.

Dia	Vento	Nublado	Sol	Chuva	Resultado
1	sim	sim	nao	sim	nao andou
2	nao	sim	nao	nao	andou
3	sim	nao	sim	nao	andou
4	nao	sim	nao	nao	andou
5	sim	nao	sim	nao	andou
6	nao	sim	nao	sim	nao andou
7	sim	nao	nao	sim	nao andou
8	nao	sim	nao	sim	nao andou
9	sim	nao	sim	nao	andou
10	nao	nao	nao	nao	andou

Tabela 3.1: Informações sobre as condições de tempo e os dias de acompanhamento do Thiago e sua tentativa de andar de skate

O Algoritmo

O algoritmo Naive Bayes é um classificador probabilístico baseado no 'Teorema de Bayes', o qual foi criado por Thomas Bayes (1701-1761) para tentar provar a existência de Deus [curiosidade]. A seguir apresentamos a regra de Bayes onde o algoritmo baseia seu funcionamento.

$$p(b|a) = \frac{p(a,b)}{p(a)} \quad (3.11)$$

que (pela lei da multiplicação de probabilidades) também pode ser escrita como:

$$p(b|a) = \frac{p(a|b) \cdot p(b)}{p(a)} \quad (3.12)$$

O Naive Bayes é um algoritmo muito utilizado na área de Aprendizado de Máquina para categorizar/classificar textos baseado na frequência das palavras usadas, e assim podendo ser usado para identificar se determinado e-mail é um SPAM ou sobre qual assunto se refere determinado texto, por exemplo.

Por ser muito simples e rápido, possui um desempenho relativamente maior do que outros classificadores. Além disso, o Naive Bayes só precisa de um pequeno número de dados de teste para concluir classificações com uma boa precisão.

A principal característica do algoritmo, e também o motivo de receber 'naive' (ingênuo) no nome, é que ele desconsidera completamente a correlação entre as variáveis. Ou seja, se o tempo, no exemplo da tabela acima, é dividido em "sol", "nublado", "chuva" ou "vento", o algoritmo não vai levar em consideração a correlação entre esses fatores, mas vai tratar cada um de forma independente.

Se o problema for classificar um texto ou algo do gênero, o Naive Bayes é uma das melhores alternativas, mas se a correlação entre os fatores for extremamente importante, o Naive Bayes pode falhar na predição da nova informação.

Dessa forma, temos que esse algoritmo é muito útil para "análise de sentimentos", "filtragem de spam", "classificação/filtragem de textos" e além disso, pode ser utilizado para previsões em tempo real, dada sua velocidade e a necessidade de poucos dados para realizar as classificações.

Determinando Probabilidades

Para entender um pouco melhor como funciona o classificador, vamos a um exemplo rápido: Digamos que estamos trabalhando no diagnóstico de uma nova doença, e que fizemos testes em 100 pessoas distintas. Após coletarmos os dados e fazer as primeiras análises, descobrimos que 20 pessoas possuíam a doença (20%) e 80 pessoas estavam saudáveis (80%), sendo que das pessoas que possuíam a doença, 90% receberam Positivo no teste da doença, e 30% das pessoas que não possuíam a doença também receberam o teste positivo.

Listando esses dados de uma forma mais clara, temos: - 100 pessoas realizaram o teste. - 20% das pessoas que realizaram o teste possuíam a doença. - 90% das pessoas que possuíam a doença, receberam positivo no teste. - 30% das pessoas que não possuíam a doença, receberam positivo no teste.

A pergunta neste caso seria: se uma nova pessoa realizar o teste e receber um resultado positivo, qual a probabilidade de ela possuir a doença?

O algoritmo Naive Bayes consiste em encontrar uma probabilidade a posteriori (possuir a doença, dado que recebeu um resultado positivo), multiplicando a probabilidade a priori (possuir a doença) pela probabilidade de “receber um resultado positivo, dado que tem a doença”.

Devemos também calcular a probabilidade a posteriori da negação (Não possuir a doença, dado que recebeu um resultado Positivo). Ou seja:

- $P(\text{doençalpositivo}) = 20\% * 90\%$
- $P(\text{doençalpositivo}) = 0,2 * 0,9$
- $P(\text{doençalpositivo}) = 0,18$
- $P(\text{não doençalpositivo}) = 80\% * 30\%$
- $P(\text{não doençalpositivo}) = 0,8 * 0,3$
- $P(\text{não doençalpositivo}) = 0,24$

Após isso precisamos normalizar os dados, para que a soma das duas probabilidades resulte em 1 (100%). Para isso, dividimos o resultado pela soma das duas probabilidades.

Exemplo:

- $P(\text{doençalpositivo}) = 0,18/(0,18+0,24) = 0,4285$
- $P(\text{não doençalpositivo}) = 0,24/(0,18+0,24) = 0,5714$

Dessa forma, podemos concluir que se o resultado do teste da nova pessoa for positivo, ela possui aproximadamente 57% de chance de não estar doente.

Exemplo

Vamos ver na prática como esse negócio funciona. Para isso, vamos utilizar uma base de dados chamada **Iris**. Trata-se de um conjunto de dados muito conhecido e muito utilizado no aprendizado de modelos de classificação. Este BD se refere às flores Iris. Nele encontramos 150 registros de medidas de largura e comprimento de sépalas e pétalas das flores. Vou importar os dados aqui e

mostrar a carinha deles para que fiquemos familiarizados.

Para uma visão completa desse exemplo, consulte nosso notebook sobre o Naive Bayes.

Para aprofundar no assunto, sugiro muito a página da biblioteca Scikit-Learn.

3.2.4 K-Vizinhos Próximos (KNN)

O método K-Vizinhos próximos - *K-Nearest Neighbours KNN* - pode ser utilizado para resolver um problema de classificação. Esse modelo é um dos mais simples e intuitivos: seu treinamento consiste em armazenar cada uma das instâncias (ou exemplos) da base de treino e, através de uma função de distância¹⁴ e do número de vizinhos mais próximos, é capaz de prever qual a classe de um novo indivíduo.

A simplicidade desse método tem um preço: ele é sensível a ruídos (aleatoriedade dos dados) e *outliers*, não possui uma capacidade alta de generalização e é altamente sensível a seu hiperparâmetro: o número de vizinhos mais próximos. Em situação de alta dimensionalidade não é muito efetivo e pouco eficiente. Seu tempo de treino costuma ser rápido, porém o tempo para predição envolve calcular a distância entre a instância (indivíduo) nova e todas as outras instâncias, sendo assim, em bases de dados muito grandes, esse método pode ter um tempo de execução alto. Em bases de dados balanceadas e representativas, esse modelo costuma ter bons resultados.

Agora trazendo um exemplo mais concreto: imagine a seguinte situação, uma exposição de veículos antigos está ocorrendo em sua cidade e é informado em uma placa próxima a cada veículo, se este foi fabricado nos Estados Unidos, Japão ou Europa. Os carros estão espalhados pelo salão sem uma lógica aparente e em algum lugar existe um veículo que não tem uma placa de nacionalidade e você quer descobrir onde este carro foi fabricado. Seguindo a lógica do KNN você faria o seguinte: olharia os vizinhos mais próximos a este carro e veria onde eles foram fabricados, digamos que você decidiu olhar os cinco carros mais próximos, três deles eram dos Estados Unidos, um do Japão e por fim um da Europa e como existem mais carros fabricados nos Estados Unidos próximos ao seu, você induziria que o carro também foi fabricado lá. É importante dizer que a distância entre os carros pode ser física ou algo que mede similaridade entre atributos levando em consideração características dos carros. Vemos que a lógica do modelo é bem simples, porém bastante poderosa, com isso em mente podemos agora partir para codificação.

Continuando com a situação acima em mente, vamos aplicar nosso modelo em forma de código. Vamos fazer apenas algumas modificações para que o método faça mais sentido - na situação acima informamos que os carros estavam dispostos sem uma lógica aparente e para o método fazer sentido vamos imaginar que os carros estejam dispostos segundo as seguintes variáveis: Milhas por litros (**mpg**), Número de cilindros (**cylinders**), Volume nos pistões (**cubicines**, Cavalos no motor(**hp**), Peso em libras (**weightlbs**), Tempo de aceleração até 60 milhas por hora (**time-for-60**) e por fim o ano do carro (**year**).

Escolhidas as variáveis, vamos continuar primeiramente importando as bibliotecas que vamos usar:

```
import pandas as pd
import numpy as np
import seaborn as sns #Usaremos esta biblioteca para plotar desta vez
from sklearn.model_selection import train_test_split# criar grupo de treino e teste
from sklearn.preprocessing import StandardScaler # normalizar os dados
```

¹⁴Ver Seção de Álgebra Linear.

```
from sklearn.neighbors import KNeighborsClassifier # biblioteca do metodo
from sklearn.metrics import classification_report, confusion_matrix # acuracia do método.
```

Então podemos importar nosso dataset e separar as classes de cada carro para facilitar a manipulação dos dados:

```
dataset= pd.read_csv("cars_.csv")

classe= dataset['brand']
dataset.drop(['brand'],axis=1,inplace= True)
```

Uma vez separadas as classes das variáveis, temos que criar nosso grupo de treino e nosso grupo de teste, para isso utilizamos uma função do python que é muito útil:

```
X_train, X_test, y_train, y_test = train_test_split(dataset,classe, test_size=0.20,random_s
```

Com os dados separados, só precisamos normalizá-los antes de aplicar o KNN:

```
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Como foi dito no exemplo o método induz qual a classe do elemento olhando as classes dos seus vizinhos próximos, então é muito importante definir qual é o numero de vizinhos que se deve olhar para conseguir chegar a um número no qual o erro é minimizado. Para isso utilizamos o código abaixo para ver a média dos erros e usar o K (número de vizinho próximos) que gera menor número de erros na base de teste:

```
error=[]
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Erro do K')
plt.xlabel('Valor K')
plt.ylabel('Média do erro')
```

Com isso temos o seguinte gráfico:

Nele fica bem claro os números que geram os menores erros para o método. Neste caso usaremos 6 como o número de vizinhos próximos, porém poderíamos usar o outro número com tranquilidade. Com o melhor K em mãos basta agora aplicar o método e testá-lo:

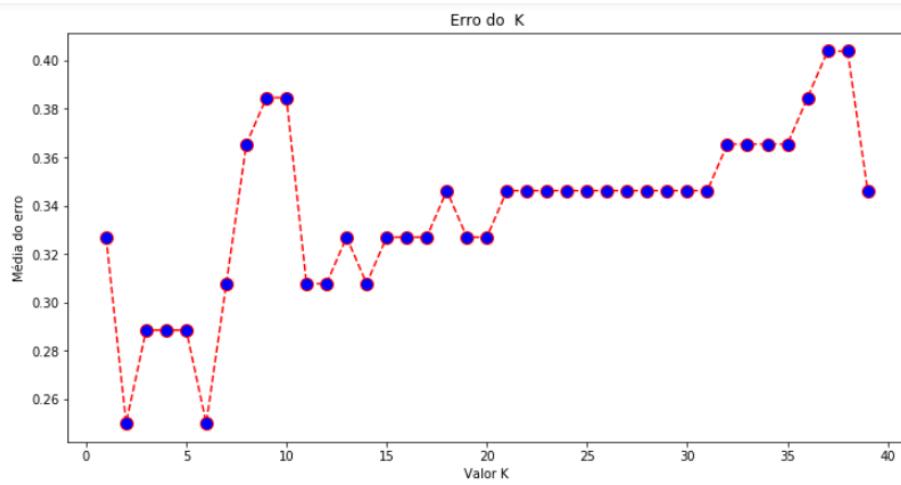


Figura 3.5: Gráfico de Erros para K vizinhos Proximos

```
classif = KNeighborsClassifier(n_neighbors=6)
classif.fit(X_train, y_train)

y_pred = classif.predict(X_test)

classif.score(X_test,y_test)
```

Com o modelo apresentado temos uma precisão de 75% em nosso grupo de teste. Porém, para validarmos o método precisamos de mais informações e para isso iremos usar uma matriz de confusão: ela nos permite ver se nosso método está acertando bem em todas as classes de nosso dataset, por exemplo. Você pode entender melhor este topico clicando aqui¹⁵. Além da matriz vamos utilizar um reporte de classificação que o sklearn nos oferece:

```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

Com esses métodos vemos que nosso modelo conseguiu acertar muito bem para os veículos dos Estados Unidos e Europa e acertando um pouco menos para veículos do Japão.

3.2.5 Máquina de vetores de suporte

Podemos dizer que o estado-da-arte em classificação possui dois métodos: redes neurais artificiais e máquinas de vetores de suporte (também chamados de *Support Vector Classifier - SVC*). É um método versátil que consiste em buscar um hiperplano de acordo com uma função, chamada de *kernel*, que consiga separar as classes de forma ótima utilizando as instâncias mais próximas a ele, ou vetores de suporte. Sua versatilidade se dá principalmente ao tipo de *kernel* utilizado: para problemas lineares pode-se escolher uma função linear $ax + b$, para problemas não lineares, funções de base radial como $e^{-\gamma r^2}$ e até mesmo funções personalizadas. É comum algumas bibliotecas, como

¹⁵<http://minerandodados.com.br/index.php/2018/01/16/matriz-de-confusao/>

sklearn¹⁶ e Weka¹⁷, implementarem diferentes tipos de SVC, variando o tipo de regularização (C , ν ou permitindo uma escolha personalizada), complexidade do algoritmo, funções de perda, etc. A figura¹⁸ abaixo mostra como diferentes *kernel* se comportam em um problema bidimensional utilizando o conjunto de dados Iris.

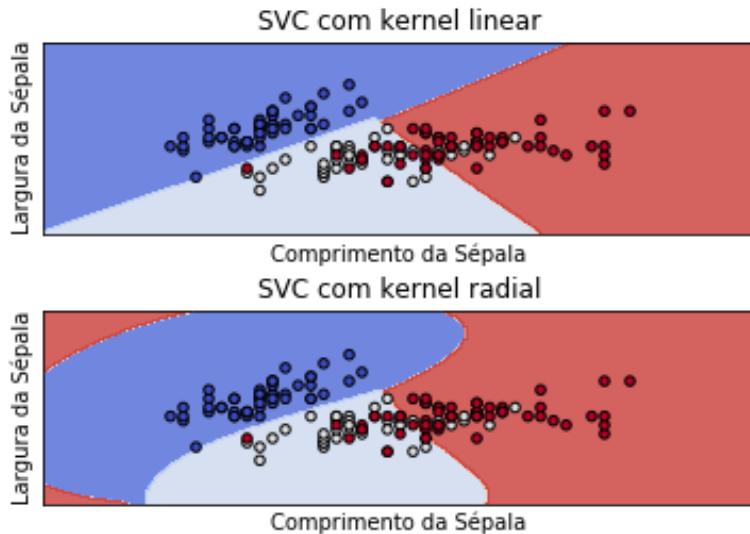


Figura 3.6: Máquina de vetores de suporte com kernel linear e de base radial. Note a diferença entre os formatos das fronteira de decisão.

3.2.6 Perceptron

O modelo Perceptron é um dos primeiros modelos de classificação de que se tem notícia e teve sua formulação feita em meados do século passado. O objetivo na hora de se formular o Perceptron por parte de seus criadores foi simular o funcionamento dos neurônios. Mas como essa simulação foi implementada? O Perceptron funciona como uma 'função de ativação' da seguinte forma: o valor saída do Perceptron é 1 se o sinal que chega no Perceptron é positivo, o valor saída do Perceptron é -1 se o sinal que chega no Perceptron é negativo e o valor saída do Perceptron é 0 se o sinal que chega no Perceptron é nulo. Em termos matemáticos dizemos que o Perceptron é modelado com uma função $sign(\cdot)$. Se z é o sinal recebido pelo Perceptron, então temos sua especificação matemática da seguinte forma:

$$\text{sign}(z) = \begin{cases} 1 & \text{se } z > 0 \\ 0 & \text{se } z = 0 \\ -1 & \text{se } z < 0 \end{cases} \quad (3.13)$$

¹⁶<http://scikit-learn.org/stable/modules/svm.html>

¹⁷<https://weka.wikispaces.com/LibSVM>

¹⁸Adaptado de http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html

De forma ilustrada, a especificação do Perceptron pode ser representada da seguinte maneira¹⁹:

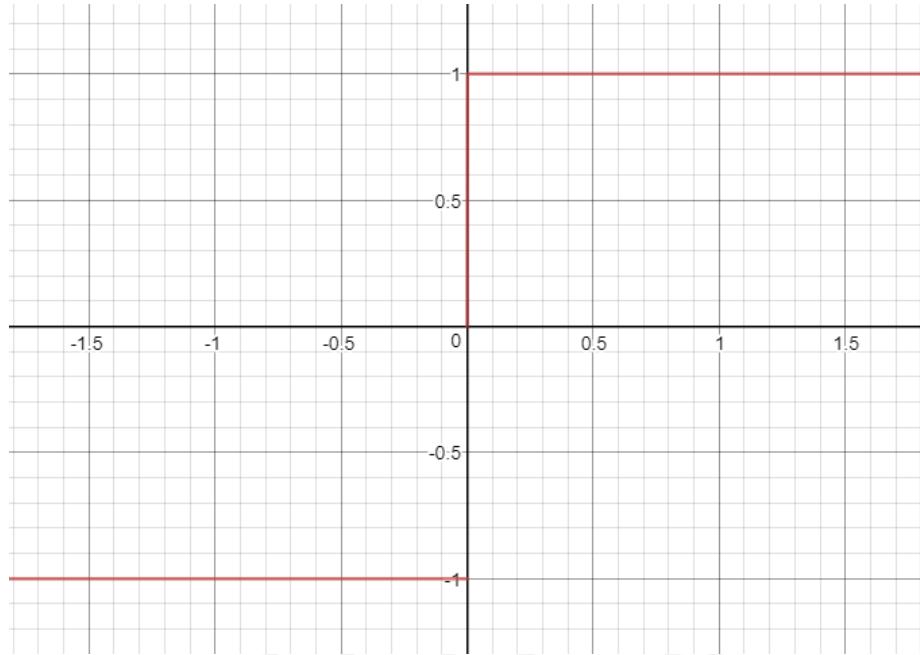


Figura 3.7: Função $\text{sign}(\cdot)$

Usando tal modelagem matemática, como fazemos nossa classificação? É importante dizer que nessa parte trataremos apenas de classificações binárias (duas classes) e que devemos começar a saber o que é o sinal z recebido pelo Perceptron. Para simplificar, imagine que em nossa base de dados temos apenas duas características de cada indivíduo: as variáveis X_1 e X_2 . O sinal z para um indivíduo (observação) qualquer i na nossa base de dados é $z_i = b + w_1x_{1i} + w_2x_{2i}$, sendo que b , w_1 e w_2 são parâmetros a serem estimados. Os parâmetros são estimados de forma a minimizar o erro de classificação cometido pelo modelo na base de treinamento²⁰. Para se fazer a predição para um indivíduo j a partir de parâmetros dados ou estimados devemos aplicar a regra vista anteriormente:

$$\hat{y}_j = \begin{cases} 1 & \text{se } z_j = b + w_1x_{1j} + w_2x_{2j} \geq 0 \\ -1 & \text{se } z_j = b + w_1x_{1j} + w_2x_{2j} < 0 \end{cases} \quad (3.14)$$

A equação acima nos diz que a predição para a classe do indivíduo j (\hat{y}_j) será a primeira classe (1) se o sinal produzido por esse indivíduo é maior ou igual a zero e a predição será uma segunda classe, representada pelo número -1 , se o sinal produzido pelo indivíduo j é menor que zero. Observe que juntamos os casos $>$ e $=$ e isso pouco importa pois com probabilidade zero teremos $z_j = 0$ calculado pelo computador. O caso em que temos apenas 2 variáveis pode ser facilmente visto no plano bidimensional²¹ e é possível perceber que uma reta divide as classes e por esse motivo dizemos

¹⁹Imagem gerada em <https://www.desmos.com/calculator>.

²⁰Nesse caso se chama "Hinge Loss- para mais detalhes acessar <http://web.engr.oregonstate.edu/~xfern/classes/cs534/notes/perceptron-4-11.pdf>

²¹Em um caso de maior dimensão, teríamos um plano ou hiperplano dividindo o espaço em dois.

que o Perceptron é um classificador linear²²:

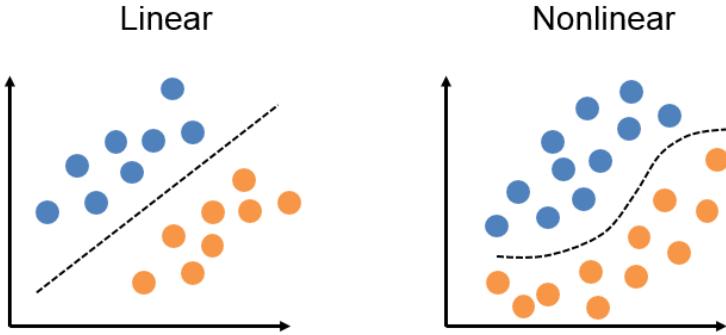


Figura 3.8: Classificador Linear Vs. Não Linear

Se estou usando dados médicos, poderia imaginar que a classe representada por -1 é uma classe "Doente" e a classe representada por 1 é a classe "Saudável". Na verdade, isso pouco importa e não terá influência no resultado final²³.

3.2.7 Regressão Logística

Vamos começar com uma motivação clássica envolvendo o modelo de Regressão Logística, que é conhecida como motivação da Variável Latente: imagine Y^* é uma variável que denota a motivação dos indivíduos e, como não podemos mensurá-la diretamente, dizemos que é uma variável latente. Assuma também que em nossa base de dados tenhamos 3 variáveis: (i) Y - binária e que diz que se os indivíduos terminaram (1) ou não (0) um curso online -, (ii) X_1 e (iii) X_2 , que são atributos mensuráveis e que impactam na motivação dos indivíduos Y^* . Imagine que se a motivação dos indivíduos não chegam a um certo limiar, os indivíduos desistem do curso. Assumindo que o limiar seja zero temos para um indivíduo j :

$$Y_j = \begin{cases} 1 & \text{se } Y_j^* = b + w_1X_{1j} + w_2X_{2j} + \varepsilon_j \geq 0 \\ 0 & \text{se } Y_j^* = b + w_1X_{1j} + w_2X_{2j} + \varepsilon_j < 0 \end{cases} \quad (3.15)$$

É visível pela equação acima que o fato de o indivíduo j continuar ou não no curso depende diretamente da motivação dele ultrapassar o limiar proposto - considere que b , w_1 e w_2 são parâmetros a serem estimados e que ε_j é um erro aleatório não perceptível nos dados e que segue uma distribuição específica arbitrária. Pela equação acima, chegamos ao fato que de a probabilidade de o indivíduo j continue no curso é dada por:

$$P(Y_j = 1|X_1, X_2) = P(Y_j^* \geq 0|X_1, X_2) = P(\varepsilon_j \geq -b - w_1X_{1j} - w_2X_{2j}) \quad (3.16)$$

²²Imagen retirada de <https://jtsulliv.github.io/perceptron>.

²³Não é necessário sequer fornecer essa informação para os pacotes no Python.

Se considerarmos que ε_j tem distribuição logística²⁴ centrada no zero (implica que seja uma distribuição simétrica no zero) a probabilidade $P(Y_j = 1|X_1, X_2)$ iguala-se a:

$$P(Y_j = 1|X_1, X_2) = P(\varepsilon_j \leq b + w_1 X_{1j} + w_2 X_{2j}) = \frac{1}{1 + e^{-(b + w_1 X_{1j} + w_2 X_{2j})}} \quad (3.17)$$

A última expressão da direita também é conhecida como Função Sigmoid, especialmente no ramo das Redes Neurais Artificiais. Assim como o Perceptron, a Regressão Logística é um classificador linear, ou seja, o modelo encontra os parâmetros que melhor traçam um hiperplano dividindo as classes minimizando assim a função perda - "Cross Entropy"²⁵. Até agora focamos mais na classificação binária, no entanto, é muito natural expandirmos esse modelo de Regressão Logística para um modelo mais geral conhecido por Regressão Logística Multinomial ou Softmax, que é o equivalente da Sigmoid para um número maior de classes. A probabilidade da classe h condicional a um vetor fixo de características para o indivíduo i é $\mathbf{x}_i = (x_{1i}, x_{2i}, \dots, x_{ki})$:

$$P(y_i = h|\mathbf{x}_i) = \frac{e^{b_h + \mathbf{x}_i^T \mathbf{w}_h}}{\sum_{j=0}^J e^{b_j + \mathbf{x}_i^T \mathbf{w}_j}} \quad (3.18)$$

Na equação acima, os parâmetros b_j e \mathbf{w}_j , para qualquer j indo de 0 a J , são parâmetros estimados na base de treinamento. Perceba que no exemplo acima existem $J + 1$ classes, sendo que J é um número natural, maior que zero e finito.

Qual a principal vantagem de se utilizar o modelo Regressão Logística Multinomial em relação aos modelos já visto?

Além da simplicidade da função perda a ser otimizada, um grande ponto alto em se utilizar este modelo é conseguir extrair quase que naturalmente os efeitos marginais de variações nas variáveis na probabilidade de uma classe específica. Mas o que isso quer dizer de forma prática? Imagine que tenhamos duas classes apenas "1=ganhar da máquina no xadrez" ou "0=perder da máquina no xadrez". Pense também que ganhar ou não da máquina é função de uma única variável que são as horas dedicadas ao treino e aprendizado de xadrez. Assuma que a função que descreve a probabilidade de ganhar dadas as horas de treinamento pode se desenhada da seguinte maneira²⁶:

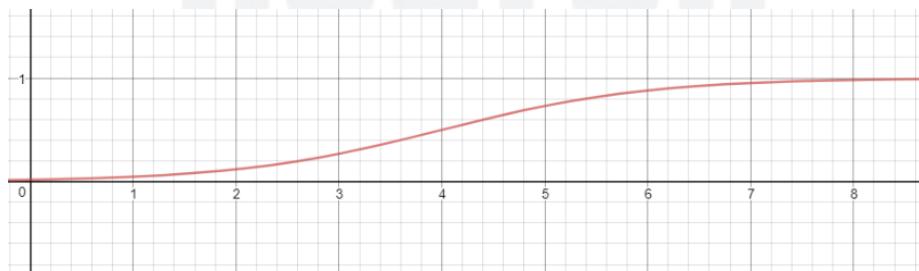


Figura 3.9: Horas de treinamento e Probabilidade de ganhar

²⁴https://en.wikipedia.org/wiki/Logistic_distribution.

²⁵Minimizar a "Cross Entropy" é equivalente a maximizar a função de verossimilhança nesse caso. Para mais detalhes a respeito dessa função, ver <http://cs231n.github.io/linear-classify/>.

²⁶Imagen gerada em <https://www.desmos.com/calculator>.

É perceptível que com poucas horas de treino, é quase impossível ganhar da máquina e que com muitas horas de treino, é quase certo que vai ganhar o jogo. Quando falamos em efeitos marginais, estamos pensando no efeito na probabilidade de se ganhar o jogo dada uma variação positiva ou negativa no montante de horas dedicadas à prática do xadrez. É importante notar que um incremento na probabilidade de se ganhar o jogo dada uma pequena variação nas horas dedicadas depende da bagagem do jogador - em outras palavras: um jogador que tem 3 horas de treino é recompensado por uma hora extra de treino (na probabilidade de se ganhar) de maneira diferente do que um jogador que tem 6h de treino e se dedica uma hora extra.

Se quisermos sabermos o impacto de uma hora extra sobre a probabilidade de se ganhar um jogo, devemos a experiência do jogador então, como fazemos isso? A experiência escolhida é um tanto arbitrária e pode ser, por exemplo, a experiência média ou mediana da amostra (indivíduo médio ou mediano). É comum também avaliar o efeito marginal em todos os indivíduos para saber o efeito médio. Quando se tem mais de um atributo e queremos avaliar o efeito marginal do incremento de uma variável na probabilidade condicional, o procedimento é igual utilizando as médias ou medianas de todas as variáveis ou calculando o efeito médio.

3.2.8 Como avaliar a performance de modelos de classificação?

O método mais intuitivo para avaliar a performance de um classificador é a acurácia, que é dada por: $\frac{\text{acertos}}{\text{total}}$, valor indo de 0 a 1. Porém esse método possui problemas: quando trabalhamos com conjuntos de dados desbalanceados seu resultado não é representativo (ex: 90 dados de classe A e 10 de classe B, utilizando essa métrica o classificador teria 0.9 de acerto se classificasse todos como classe A) e ele não fornece informações sobre a performance individual de cada classe. Sendo assim, outras métricas são propostas na literatura (considere VP como verdadeiros positivos, VN como verdadeiros negativos, FP como falsos positivos e FN como falsos negativos, tendo como referência uma determinada classe):

- Acurácia: $p = \frac{VP+VN}{Total}$, taxa de verdadeiros positivos preditos.
- Precisão: $p = \frac{VP}{VP+FP}$, taxa de verdadeiros positivos preditos.
- Recall ou Sensibilidade: $r = \frac{VP}{VP+FN}$, taxa de verdadeiros positivos.
- Especificidade: $e = \frac{VN}{VN+FP}$, taxa de verdadeiros negativos.
- F1: $\frac{2}{p^{-1}+r^{-1}}$, que considera recall e precisão para uma medida mais geral considerando as taxas de acertos para determinada classe.

Também é comum utilizar uma matriz de confusão para avaliar em conjunto com essas métricas. A figura²⁷ a seguir ilustra a matriz de confusão e sua relação com as métricas descritas.

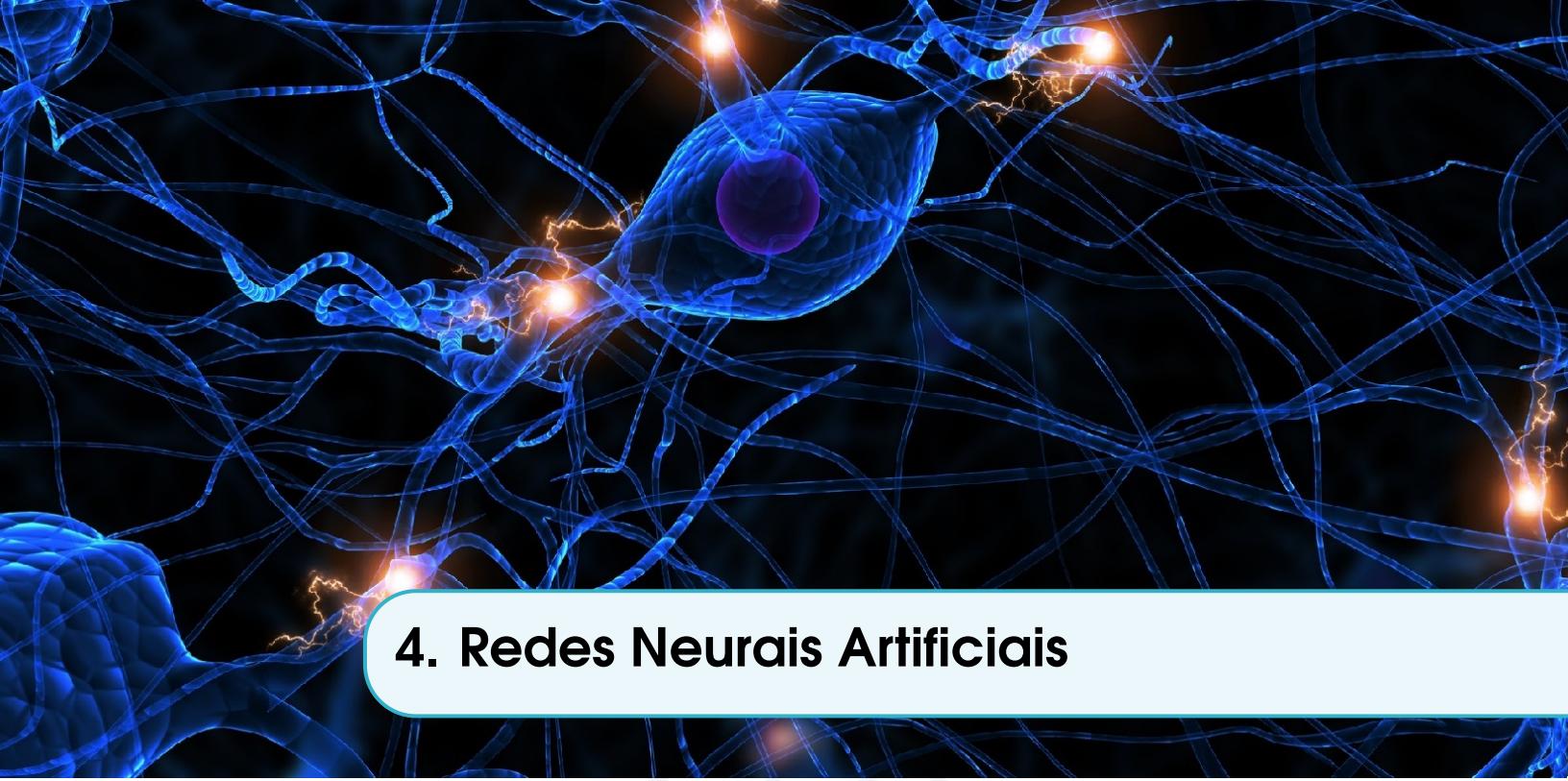
²⁷<http://developerdeveloper.blogspot.com.br/2013/11/matriz-confusao.html>

		Valor Verdadeiro (confirmado por análise)	
		positivos	negativos
Valor Previsto (predito pelo teste)	positivos	VP Verdadeiro Positivo	FP Falso Positivo
	negativos	FN Falso Negativo	VN Verdadeiro Negativo

Figura 3.10: Matriz de confusão

3.2.9 Materiais Complementares

- <https://www.youtube.com/watch?v=gYSWrUP4aB0&t=2s>
- https://rstudio-pubs-static.s3.amazonaws.com/123284_bf5c001edce8407cbf893acaffe3eb3a.html
- <https://towardsdatascience.com/random-forest-in-python-24d0893d51c0>
- <https://www.vooo.pro/insights/um-tutorial-completo-sobre-a-modelagem-baseada-em-tree-e-random-forest/>
- <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>
- http://www.saedsayad.com/decision_tree.htm
- <http://www.semspirit.com/artificial-intelligence/machine-learning/regression/random-forest-regression/random-forest-regression-in-python/>



4. Redes Neurais Artificiais

A demanda de melhoramento dos algoritmos de aprendizado de máquina e o aumento da capacidade de processamento propiciaram que algoritmos que até então não eram usados com frequência se tornassem mais populares, como é o caso da Redes Neurais Artificiais. Tais métodos têm ganhado grande destaque, em vista da sua versatilidade e seus resultados muitas vezes são melhores do que outras técnicas de classificação e regressão, tendo sido utilizada por grupos de investimento para analisar o mercado financeiro, por hospitais para auxílio na tomada de decisão médica, principalmente na parte de imagens, na robótica entre outros.

4.1 Redes Neurais

As Redes Neurais Artificiais (RNA) foram desenvolvidas tendo como inspiração o funcionamento o cérebro humano, ou seja, a forma como processamos informações, uma vez que os computadores, tradicionalmente, processam informações de uma maneira bem diferente.

O cérebro é um computador altamente complexo, não linear e paralelo, tendo a capacidade de organizar seus constituintes estruturais, conhecidos como neurônios, de forma a realizar certos processamentos, sendo capaz de aprender e de generalizar¹.

Os neurônios são as principais células do sistema nervoso, sendo constituído por três partes: Dendritos, Corpo Celular e Axônio, como podemos ver na imagem abaixo². A maioria dos neurônios possuem seu axônio envolto em bainha de mielina, que auxiliam na condução do impulso nervoso.

A comunicação entre os neurônios é feita por Sinapse, que é a passagem de um impulso elétrico gerado por um neurônio para outro. Essa passagem é feita pelos terminais do axônio para os dendritos de outro neurônio.

O neurônio em repouso tem uma diferença de potencial, ou seja, ele é carregado negativamente do lado interno e positivamente do lado externo. O impulso elétrico do neurônio é realizado pela saída de potássio (K) e a entrada de sódio (Na). O sódio entra quando ocorre o estímulo que consegue

¹Haykin, S. Redes Neurais - Princípios e prática. 2 ed. Porto Alegre: Bookman, 2001.

²<https://giorgiafiorio.org/wp-content/uploads/2018/04/Neuron-Anatomy-neuron-anatomy-and-physiology-Nerve-cell.jpg>

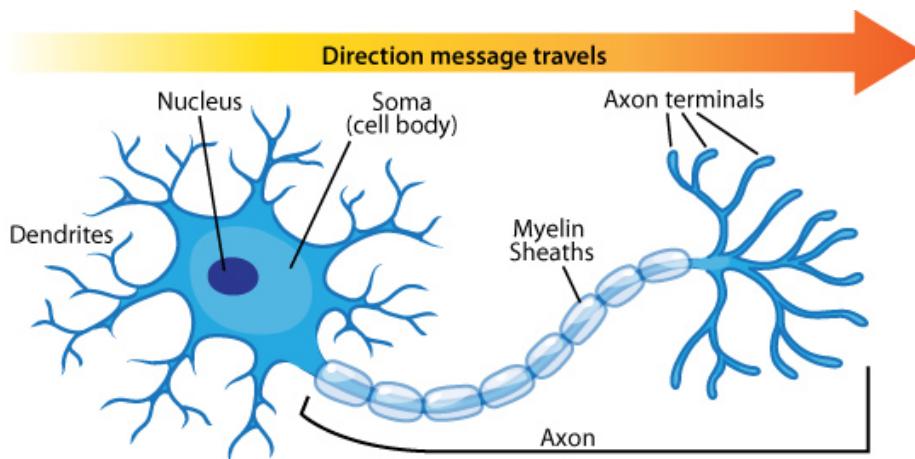


Figura 4.1: Representação esquemática de um neurônio

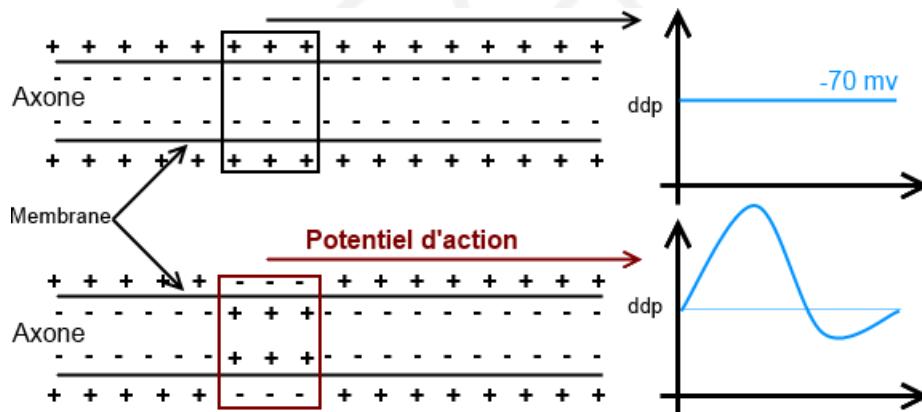


Figura 4.2: Representação esquemática do processo de despolarização de um neurônio

superar o limiar de ativação, ocorrendo a abertura dos canais de sódio do neurônio e a saída do potássio, esse processo chamado de despolarização, como podemos ver na imagem abaixo³.

Assim sendo, gera o impulso elétrico que será propagado por todo o neurônio, seguindo o sentido do Dendrito → Corpo Celular → Axônio.

Logo após a passagem do impulso, ocorre a repolarização, onde o sódio volta para a região externa e o potássio regressa para o interior do neurônio, assim o neurônio volta ao estado de repouso.

O impulso elétrico gerado vai se propagando para vários outros neurônios, podendo ativar regiões específicas do cérebro responsáveis por funções motoras e/ou cognitivas. De forma análoga podemos construir modelos artificiais que utilizam essa arquitetura para assim auxiliar na resolução de diversos problemas complexos. A seguir vamos abordar com maior profundidade ao modelo Perceptron e o Perceptron de Multi Camadas (MLP)

³https://www.bac-s.net/docs/import_html/3142/index_html_m79e1466.png

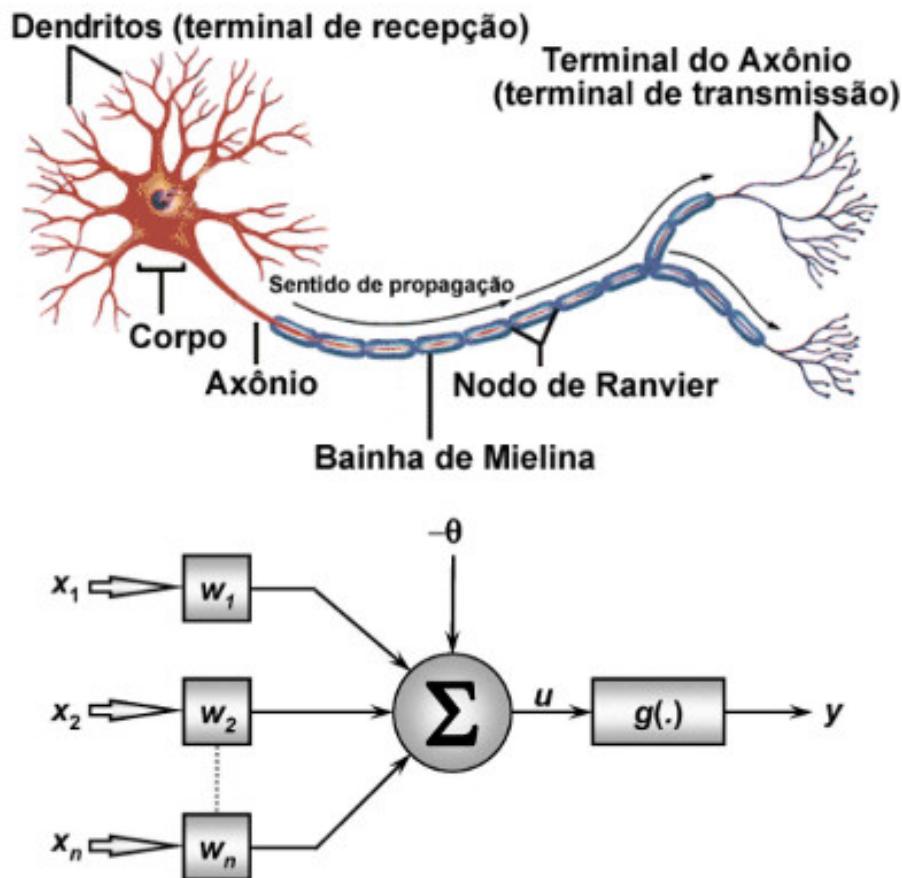


Figura 4.3: Representação esquemática de um neurônio artificial comparado com um neurônio biológico

4.2 Perceptron

Como visto anteriormente, o Perceptron é uma Rede Neural tendo uma ou múltiplas unidades de entrada e uma unidade de saída, sem camadas ocultas, sendo um classificador linear.

Podemos ver abaixo a uma comparação entre um neurônio artificial e um neurônio biológico⁴, assim podemos compreender melhor a analogia entre o processamento do cérebro humano e o processamento artificial.

Na figura que compara o neurônio biológico com o artificial podemos ver as entradas x_1, x_2, x_3 e os respectivos pesos w_1, w_2, w_3 , lembrando que os valores dos pesos são atribuídos no treinamento. O somatório é uma combinação linear dos valores de entrada ponderada pelos pesos. O resultado dessa função é avaliado por uma função de "ativação"(como a função sign já vista) e temos a divisão dos valores em duas classes distintas como output.

⁴<https://www.monolitonimbus.com.br/perceptron-redes-neurais/>

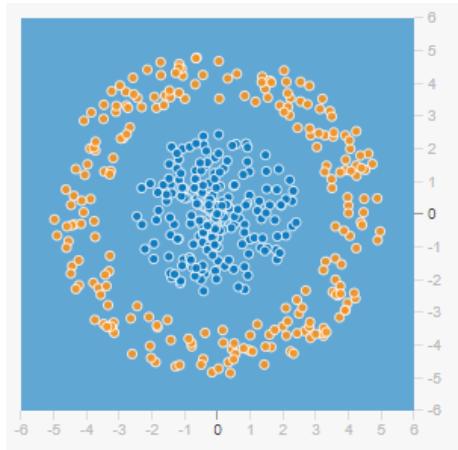


Figura 4.4: Duas classes divididas de forma não linear

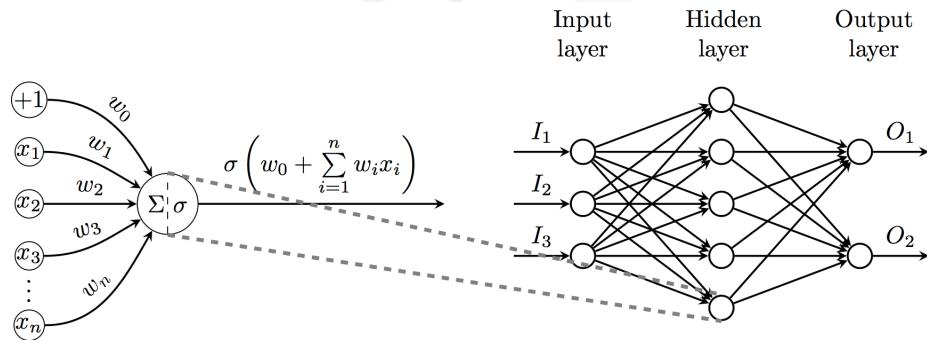


Figura 4.5: Representação esquemática de uma Rede Neural de Múltiplas Camadas

4.3 Multilayer Perceptron

Utilizamos o Multilayer Perceptron para abordar problemas de classificação os quais as classes estão separadas por funções não-lineares, ou seja, hiperplanos deixam de ser capazes de dividir as classes. O exemplo mostrado na figura seguinte⁵, onde temos duas classes, e sabemos que para separar elas, necessitamos de uma curva não linear - logo o Multilayer Perceptron é recomendado.

Essas redes usam mais de uma camada oculta de neurônios, ao contrário do perceptron de camada única⁶, como podemos ver na figura a seguir, sendo que nas camadas ocultas são aplicadas transformações não lineares chamadas de funções de ativação (e.g. sigmóide, ReLU, tanh etc). Perceba que o número de nós na camada de *output* igual ao número de classes:

A camada de entrada (*Input Layer*) consiste-se em neurônios que aceitam os valores de entrada. A saída desses neurônios é igual às entradas. Todos os outros nós recebem combinações lineares dos outputs dos neurônios anteriores, sendo que as combinações são ponderadas por pesos e soma-se uma constante - o bias (b). Os neurônios das camadas ocultas aplicam uma função de ativação antes de soltarem seus outputs, que serão combinados novamente, como descrito. A camada final, de *output*, pode ter ou não um classificador linear (como o SoftMax, caso estejamos interessados em

⁵Figura gerada no site <http://playground.tensorflow.org>

⁶<http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/>

um problema de classificação) ou somente a saída da combinação feita a partir da camada oculta anterior, no caso de regressão. Isso pode ser escrito em forma de multiplicação de matrizes, para o Multilayer com $k - 1$ camadas ocultas, como segue⁷:

$$y(\mathbf{x}_i) = f_k \left(\mathbf{w}_k^T \dots f_2 \left(\mathbf{w}_2^T f_1 \left(\mathbf{w}_1^T \mathbf{x}_i + b_1 \right) + b_2 \right) \dots + b_k \right) \quad (4.1)$$

As funções f_j na equação acima são as funções de ativação não lineares, exceto pela f_k , que é a última. Nesse caso, f_k é aplicada na saída da output layer, sendo um classificador linear (e.g. SoftMax ou sigmoid), no caso de uma classificação, ou uma transformação identidade (multiplica por 1) no caso de regressão. As camadas ocultas (*Hidden Layers*) estão entre as camadas de entrada e saída e normalmente o número de camadas ocultas varia dependendo do problema. Quando um valor chega para um neurônio intermediário, ele é transformado de acordo com uma função de ativação como a sigmóide⁸:

$$f(z) = \frac{1}{1 + e^{-z}} \quad (4.2)$$

Ou a ReLU:

$$f(z) = \max\{0, z\} \quad (4.3)$$

A camada de saída (*Output Layer*) é a camada final de uma rede neural que retorna o resultado de volta ao ambiente do usuário. Com base no projeto de uma rede neural, também sinaliza as camadas anteriores sobre como elas se comportaram no aprendizado da informação e melhoraram suas funções. Os nós na camada de saída podem usar a função SoftMax em caso de uma classificação K classes⁹ ou uma função identidade (multiplica por 1) caso quiséssemos fazer uma regressão.

Agora que vimos mais a fundo como uma estrutura de uma rede neural MLP é, podemos ver como o problema de classificação não linear, já abordado, se resolve com 2 camadas intermediárias ocultas com 4 neurônios cada uma como segue¹⁰:

Da mesma maneira que o Perceptron, o MLP são baseados em formulações matemáticas, um tanto complexas, para entender melhor essas formulações recomendamos que acessem os links do material complementar.

4.4 Deep Learning

*Deep Learning*¹¹, ou aprendizado profundo, é um conceito de redes neurais que difere do tradicional pois adiciona-se neurônios e camadas, possui relações complexas de conexão entre as camadas, exige maior poder computacional para execução e, mais importante, efetua extração de atributos automática. No conceito tradicional, as redes neurais eram totalmente conectadas e necessitavam que uma prévia caracterização, ou extração de atributos fosse efetuada, enquanto as

⁷<https://dzone.com/articles/deep-learning-via-multilayer-perceptron-classifier>

⁸<https://dzone.com/articles/deep-learning-via-multilayer-perceptron-classifier>

⁹<https://dzone.com/articles/deep-learning-via-multilayer-perceptron-classifier>

¹⁰Figura gerada no site <http://playground.tensorflow.org>

¹¹Deep Learning Practitioner's Approach, Josh Patterson and Adam Gibson.

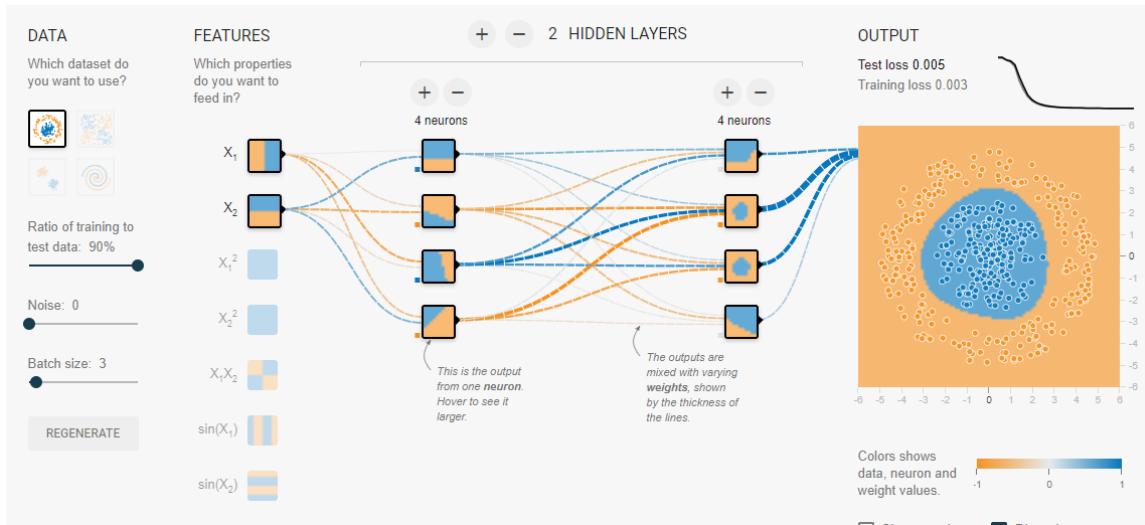


Figura 4.6: Aplicação de um MLP para a classificação não linear

redes profundas trabalham de forma localmente conectada e efetuam a extração de atributos em dados brutos. Contudo, a quantidade de parâmetros do modelo se torna excessivamente alto, fazendo com que o custo computacional e o tempo de treino aumentassem, bem como a necessidade de um conjunto de dados de treino maior.

4.4.1 Redes Neurais Convolucionais

O objetivo de uma rede neural convolucional, ou CNN, é extrair atributos de alto nível no dado via convolução¹². Esse tipo de abordagem é muito comum quando trabalha-se com imagens, seja para classificação ou segmentação.

A inspiração biológica de uma CNN é no córtex visual dos animais: as células visuais são sensíveis a pequenas sub-regiões do campo de visão e as células ativam de acordo com o tipo de padrão encontrado¹³.

De forma intuitiva, a rede organiza os dados de entrada (imagens) como matrizes N-dimensionais com altura, largura e profundidade (ou canais de cor, como o RGB, por exemplo) - os dados constituem a camada de entrada. Na próxima etapa, que é a de extração de atributos (veja figura), a rede se organiza majoritariamente em quatro tipos de camadas intermediárias: Convolucionais, *Pooling*, funções de ativação e Camadas densas com funções de ativação. Uma camada convolucional será a camada onde o processo de convolução realizará os cálculos necessários para a extração de atributos das imagens, enquanto a camada de *Pooling* será utilizada para redução de dimensionalidade com o objetivo de reduzir o *overfitting* do modelo. Já as camadas de função de ativação são colocadas principalmente após as convolucionais e as densas são camadas totalmente conectadas (como as do MLP) geralmente colocadas após as extrações de atributos e antes de se fazer a classificação. Os três primeiros tipos de camadas intermediárias são intercaladas de uma maneira específica e depois pode haver algumas camadas densas com funções de ativação antes do classificador. Por fim a camada de

¹²<https://pt.wikipedia.org/wiki/Convolução>

¹³Eickenberg et al. 2017. "Seeing it all: Convolutional network layers map the function of the human visual system."

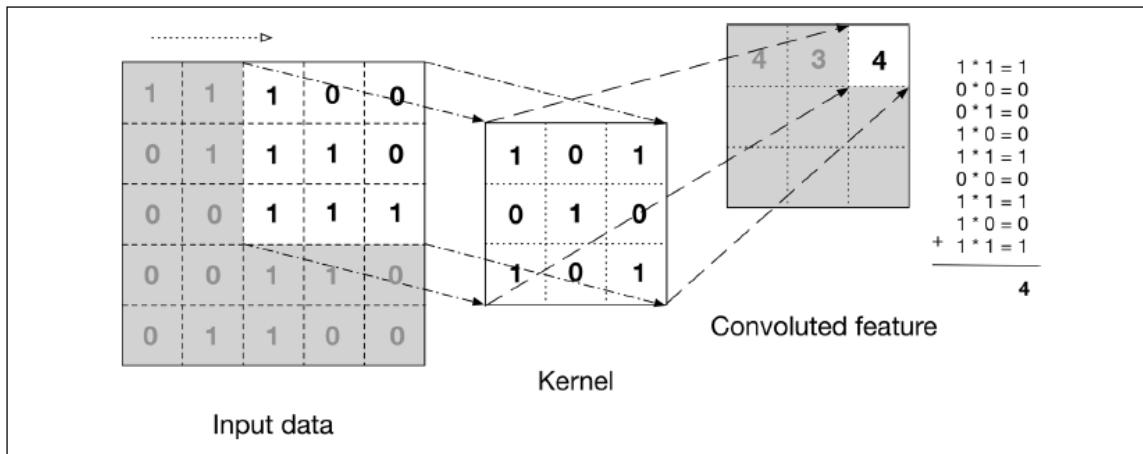


Figura 4.7: Extração de atributos utilizando convolução em CNN

14

classificação computa o *score* de cada uma das classes utilizando um classificador linear já visto (e.g. SoftMax).

Quando falamos de modelos CNN, devemos ficar atentos na configuração de uma grande quantidade de hiperparâmetros (se comparado com outros modelos). A seguir temos uma lista com uma breve explicação de alguns desses hiperparâmetros e possíveis heurísticas para ajuste dos mesmos.

- Tamanho do filtro de convolução: quantidade de pixels computados em cada convolução, estima-se que deve ser um valor baixo, por exemplo 2x2 ou 5x5.
- Inicialização dos pesos: Os pesos de uma rede neural devem ser iniciados de alguma maneira - sendo que esse início muitas vezes pode influenciar o ponto de convergência do modelo. Métodos comuns de inicialização incluem valores aleatórios de acordo com uma distribuição normal padrão.
- Função de ativação: Função que define o formato da saída de cada neurônio. Pode ser diferente em cada camada.
- Tamanho de saída (ou Mapa de Características): Quantidade de filtros gerados por uma camada que irá ser utilizado de entrada para a próxima camada. Não existe técnica bem definida dos valores iniciais, mas especula-se que deve aumentar de acordo com a profundidade.
- Número de camadas ocultas e quantidade de Convoluçãoes/Pooling: Não existe técnica que recomenda esse valor. Quanto mais profunda for a rede, mais profunda vai ser sua capacidade de abstrair informações dos dados de entrada, porém mais complexo será o ajuste dos outros hiperparâmetros, mais parâmetros (pesos) deverão ser estimados e mais sensível a *overfitting* será o modelo. A quantidade de convoluções antes de uma camada de *pooling* também não é definida, mas é comum que seja visto na literatura uma ou duas convoluções seguidas de uma camada de *pooling*.
- Regularização: técnica aplicada para reduzir a sensibilidade do modelo ao *overfitting*. Exemplos de regularizações aplicadas são L1, L2 e Dropout (desativação de neurônios).
- Taxa de aprendizado (Learning Rate): intuitivamente é o quanto rápido seu modelo aprende de acordo com os exemplos. O valor deve ser pequeno o suficiente para permitir convergência da

rede, mas grande suficiente para que o treinamento seja feito em tempo viável. Em CNN as taxas de aprendizado costumam estar entre 0.1 e 0.0001.

- Épocas (epochs): Quantas vezes todos os dados vão iterar no modelo. Também chamado de tempo de treino.

Na prática, a arquitetura da rede utilizada vem da literatura, de redes já publicadas¹⁵, e a maior parte desses parâmetros já vem fixados, como o tamanho da convolução, funções de ativação e até mesmo a taxa de aprendizado utilizada. Porém, mesmo utilizando redes "prontas" alguns hiperparâmetros podem ser otimizados (como a taxa de aprendizado). Outra prática comum é o uso de redes já com pesos treinados, ou *transfer learning*, onde usa-se uma rede já treinada e apenas realiza poucas épocas para ajustar os pesos¹⁶.

Existem diversos *frameworks* para trabalhar com redes neurais profundas. Nessa apostila será usado o Keras¹⁷.

Materiais Complementares

- <https://www.learnopencv.com/understanding-feedforward-neural-networks/>
- <https://www.quora.com/What-are-some-neural-network-architectures>
- <http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/>
- <https://dzone.com/articles/deep-learning-via-multilayer-perceptron-classifier>
- <https://www.youtube.com/watch?v=nirNIh2Vmknk>
- <https://deeplearning4j.org/glossary>

4.4.2 Redes Neurais Recorrentes

Até agora vimos modelos "estáticos": temos diversos atributos de cada um dos indivíduos em um momento (ou mais) do tempo e, mesmo que levássemos em consideração os diversos períodos, não tínhamos a necessidade ou a vontade de modelar esses dados como sendo sequenciais. Os modelos de Redes Neurais Recorrentes são modelos que trabalham com sequências, conhecidas muitas vezes como séries temporais, e as modelam para realizar previsões em diversos pontos da sequência levando em consideração a ordem dos fatos. Por que é tão importante levar em consideração a ordem dos fatos quando estamos trabalhando com sequências? Vamos explorar aqui dois exemplos rápidos.

Primeiramente, vamos supor que queremos prever qual a próxima palavra em uma frase incompleta - é evidente que neste caso estamos trabalhando com uma sequência de dados, pois podemos considerar as palavras como sendo "pontos" nos dados e estas adquirem o sentido que queremos quando são colocadas em uma sequência de palavras, que indicam um contexto. Em relação ao segundo exemplo, considere que queremos saber qual será o comportamento do preço (ou retorno) de um ativo no próximo instante e, que para isso, não bastará olhar os dados referentes somente ao último período mas como esses dados evoluíram no passado e repercutiram na nossa variável de interesse em momentos passados. É importante frisar que em ambos os casos o contexto de toda a sequência pode ter impacto na próxima previsão que queremos fazer.

Uma rede neural recorrente pode ter uma arquitetura e representação como na seguinte imagem:

Na figura que representa uma Rede Neural Recorrente temos a imagem dividida em duas partes, uma antes e outra depois da seta azul. As duas querem dizer exatamente a mesma coisa, no entanto

¹⁵LeNet, VGG16, AlexNet, etc

¹⁶A justificativa intuitiva: Se a rede consegue detectar uma curva em uma imagem de carro ela consegue detectar uma curva em uma imagem de cachorro, então uma rede treinada para carros pode ser um início bom para os valores dos meus pesos

¹⁷<https://keras.io>

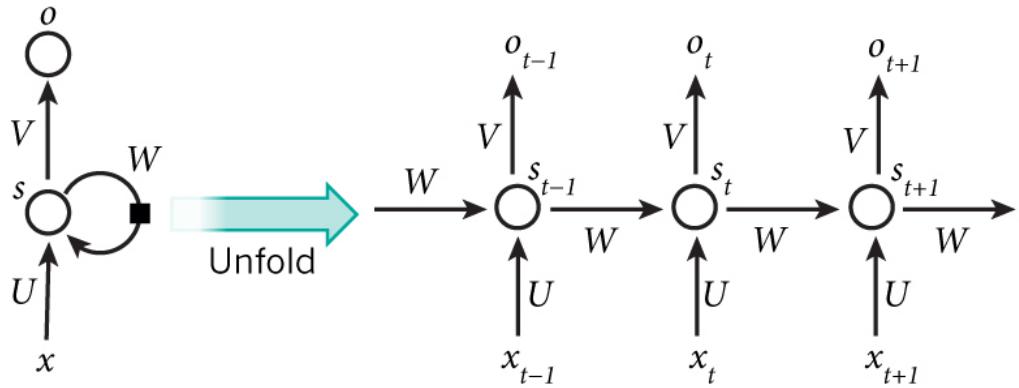


Figura 4.8: Representação de uma Rede Neural Recorrente (RNN)

18

são formas diferentes de se representar uma RNN, sendo que a representação à esquerda da seta é uma forma mais limitada e por conta disso vamos focar na representação à direita. Temos que levar em consideração o fato de que à direita temos um excerto, contendo apenas três períodos, do meio da rede. Vamos considerar aqui que $t - 1 = 0$, sendo esse é o primeiro período e que W , V e S representam um conjunto de pesos (matrizes de pesos) invariantes no tempo.

Para inicializar a rede, assumimos que os pesos em W , V e S começam como números aleatórios e de baixa magnitude, como é feito em modelos que já vimos. Um vetor de características da sequência em $t - 1$, o x_{t-1} , tem seus fatores combinados de forma linear de acordo com os pesos U e somados a uma constante b (bias) gerando um vetor S_{t-1} após a aplicação de uma função de ativação não-linear, como a ReLU ou a tanh. Após a aplicação da função não linear, esse vetor resultante S_{t-1} gera outros dois outputs após a combinação de seus componentes e à aplicação de um função de ativação: (i) O_{t-1} é o primeiro output da rede e é obtido de acordo com os pesos em V e a aplicação de uma função de ativação para classificação (e.g. softmax, sigmoid) ou para regressão (identidade) e (ii) uma outra parte que terá seus elementos combinados pelos pesos em W e levada à próxima "unidade" da rede.

Na segunda "unidade" dessa rede geraremos S_t a partir da aplicação de um função de ativação não-linear (e.g. ReLU, tanh) sobre a soma entre os vetores resultante das operações WS_{t-1} e UX_t e um vetor constante, com todas as entradas iguais, o *bias*. Após a geração de S_t , esse vetor gera dois outputs como o que foi visto anteriormente para S_{t-1} . De forma algébrica, podemos representar as operações que descrevemos aqui como as seguintes equações, sendo que g representa algum tipo de função de ativação como ReLU/tanh e f representa um regressor (identidade) ou um classificador (e.g. softmax ou sigmoid):

$$S_t = g(WS_{t-1} + UX_t + b_s) \quad (4.4)$$

$$O_t = f(VS_t + b_o) \quad (4.5)$$

Como falamos, quando estamos inicializando a rede, a primeira unidade funciona da seguinte

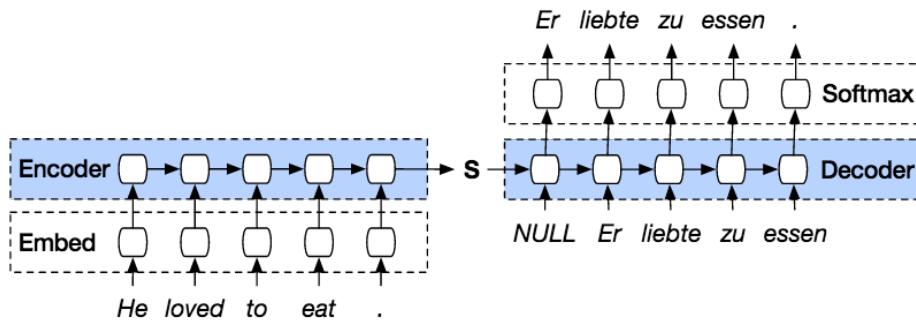


Figura 4.9: Outra representação de uma Rede Neural Recorrente "many to many"(RNN)

20

maneira:

$$S_0 = g(Ux_0 + b_s) \quad (4.6)$$

$$O_0 = f(VS_0 + b_o) \quad (4.7)$$

Como dissemos, os pesos contidos nas matrizes W , U , e V (não podemos deixar o bias de fora!) começam como sendo aleatórios, mas como a rede ajusta os pesos para que faça previsões boas? Essa arquitetura de rede, como muitos outros modelos vistos até ao momento, são modelos de aprendizado supervisionado, ou seja, devem conter os rótulos e os resultados numéricos referentes a cada output para o treinamento do modelo. Então, para cada O_t teremos um resultado que esperaríamos e o resultado que a rede deu. Comparando esses resultados para cada "unidade" da nossa rede, conseguimos definir uma função perda somando todos esses erros cometidos pelo modelo. Depois de sabermos qual é o erro cometido pelo modelo como um todo, aplicamos alguma variação do método do gradiente para minimizar esse erro e encontrar os melhores parâmetros - no modelo RNN, esse algoritmo de ajuste de parâmetros é chamado de "Backpropagation Through Time"¹⁹.

A arquitetura de Rede Neural Recorrente apresentada é apenas uma das arquiteturas possíveis. Vamos abordar mais três tipos de arquiteturas possíveis aqui e algumas de suas aplicações mais famosas. A arquitetura que vimos é do tipo "many to many" pois temos várias entradas x_t e vários outputs O_t . Poderíamos ter ainda um outro exemplo de arquitetura do tipo "many to many", como mostrado na máquina de tradução na imagem.

Essa rede é dividida em duas partes principais, sendo que a primeira é chamada de "encoder" e a segunda de "decoder". Outras arquiteturas podem ser vistas na seguinte imagem:

Na imagem podemos ver diversos tipos de arquiteturas. As redes "many to one" são muito conhecidas por ter bom desempenho na análise de sentimentos de textos e a rede "one to many" pode ser usada para a geração de textos e músicas, por exemplo.

¹⁹Para mais detalhes ver <https://www.coursera.org/learn/nlp-sequence-models/lecture/bc7ED/backpropagation-through-time> e <https://www.youtube.com/watch?v=6niqTuYFZLQ>.

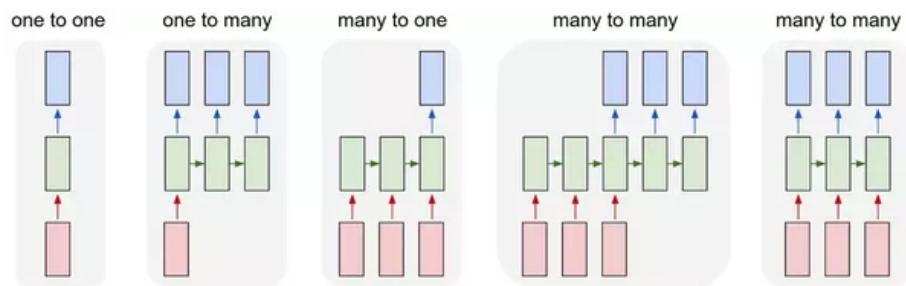


Figura 4.10: Diversas arquiteturas de RNN

21

Materiais complementares

Para ter uma noção melhor que do são as RNN, visitar os seguintes endereços:

- <https://www.coursera.org/learn/nlp-sequence-models/lecture/0h7gT/why-sequence-models>
- <https://www.youtube.com/watch?v=6niqTuYFZLQ>

