

Objetivo do curso: compreensão básica de modelos de aprendizado de máquina.

1 O que é Machine Learning?

É uma área relativamente antiga que é estudada há várias décadas. O que chamou atenção recentemente foram as boas performances em algumas tarefas de grande interesse. Entre elas, a análise de imagens.

Convolutional Neural Network foi usado em escala pela primeira vez em 2012, o que mostrou sua performance. Desde lá, ela apenas melhorou. O **Deep Learning** está passando a performance humana em alguns casos na análise de imagens.

Outra área que o **Machine Learning** foram os jogos. As máquinas conseguem entender e performar melhor que um humano nos jogos. Por exemplo, um dos melhores jogadores de *Go* perdeu para um humano.

Como realmente funciona? Basicamente, nós vamos ensinar uma máquina a aprender. Damos exemplos para ela (amostras de dados), e o que gostaríamos que ela concluísse com os dados.

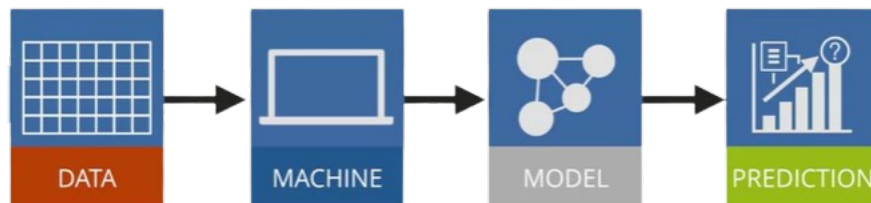


Figura 1: Ordem das ações de Machine Learning.

Para treinar, damos vários exemplos nesse formato: esse conjunto de dados x_1 resulta nesse y_1 , um outro conjunto x_2 em y_2 e assim por diante.

Para cada x_i que ele receber, deve retornar o respectivo y_i ; Porém, se passarmos um conjunto que não foi contemplado, ele deve ser capaz de prever qual será o resultado.

Queremos que ele aprenda os parâmetros do modelo matemático e prever o y_i dado o x_i .

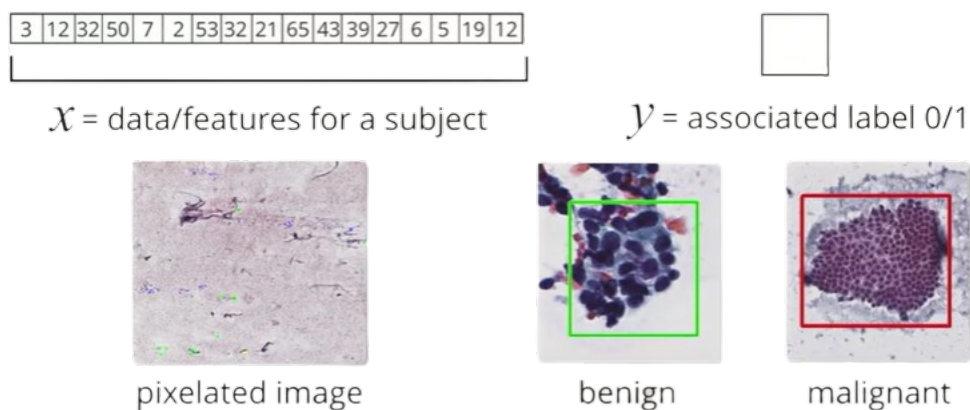


Figura 2: Exemplo de Machine Learning para identificar tumores benignos e malignos.

1.1 Regressão Lógica

Queremos um training set que consiga aprender um modelo e consegue prever o resultado dado um conjunto de dados.

Para fazer o *learning* temos um algoritmo que é feito com vários parâmetros e *learning* significa que gostaríamos de inferir quais parâmetros desse modelo são consistentes com nossos dados de treinamento.

Vamos considerar um dos algoritmos mais básicos: **Logistic Regression**

O objetivo do **Machine Learning** é que, dado N exemplos, data x e outcome y , gostaríamos de construir modelo preditivo que é capaz de prever y dado x .

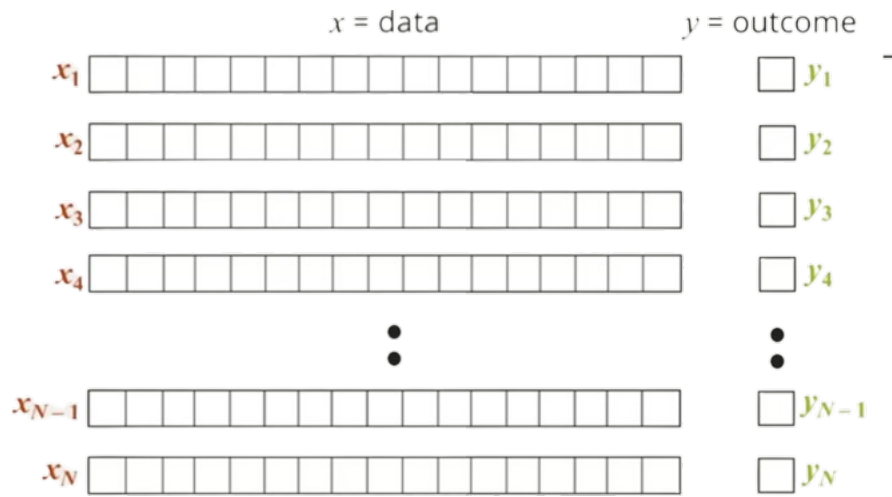


Figura 3: Exemplo Logistic Regression.

Linear Predictive Model: X_{i1} é o primeiro componente do vetor X , X_{i2} o segundo e assim por diante até X_{iM} .

Vamos multiplicar cada componente do vetor X por um parâmetro e somamos um bias:

$$(b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \dots + (b_M \times x_{iM}) + b_0$$

Isso é um mapeamento dos dados X_i para um número Z_i .



$x_i = \text{features for day } i$	features				outcome	$y_i = 1, \text{ yes}$ $y_i = 0, \text{ no}$
	Cloud Cover	Humidity	Temperature	Air Pressure	Did it Rain	
	0.5	80%	75	1.2	1	
	0.2	95%	83	1.3	0	
$z_1 = (b_1 \times 0.5) + (b_2 \times 0.8) + (b_3 \times 75) + (b_4 \times 1.2) + b_0$					$y_1 = 1$	
$z_2 = (b_1 \times 0.2) + (b_2 \times 0.95) + (b_3 \times 83) + (b_4 \times 1.3) + b_0$					$y_2 = 0$	

Figura 4: Conjunto de dados exemplo para prever a chuva.

Muitas vezes é melhor dar uma chance se vai chover ou não em um dia do que afirmar algo. Para fazer isso, usamos uma **Logistic Function** notado por σ :

$$p(y_i = 1|x_i) = \sigma(z_i)$$

z_i : multiplicação dos parâmetros dos dados X com os parâmetros b_1, b_2 até b_M .

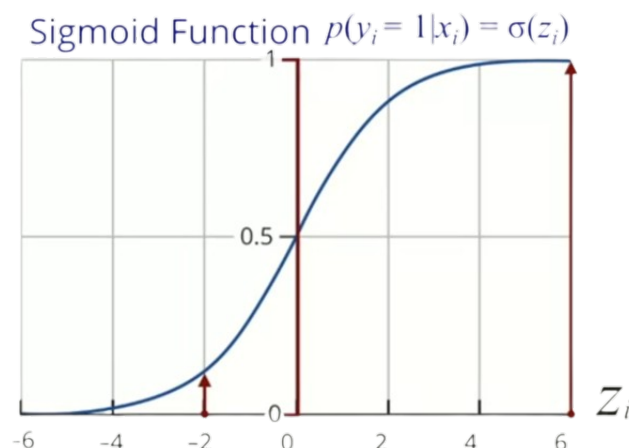


Figura 5: **Sigmoid Function** é uma maneira de converter previsões para uma perspectiva probabilística.

Essa função, a **Sigmoid Function** $p(y_i = 1|x_i) = \sigma(z_i)$, sempre está entre 0 e 1. Quando z_i é grande, como 5 ou 6, a função converte ele para um número perto de 1. Quando é pequeno, -1, -2, -4, converte para perto de 0.

Os parâmetros b dizem o quão importante as variáveis são para a predição.

É um modelo bem simples; é apenas uma combinação linear de multiplicação das variáveis observados pelos parâmetros associados, somando-os, mapeando eles para uma variável z_i e, então, executando-os por meio de uma função Sigmoid Function.

O coração do machine learning é: temos um modelo paramétrico que é caracterizado por um conjunto de parâmetros que queremos aprender. A maneira em que fazemos o aprendizado é ter um conjunto de dados e, para esses dados, temos parâmetros X e resultado Y . Gostaríamos de aprender os parâmetros do nosso modelo de tal forma que as previsões do modelo sejam consistentes com os dados do treinamento.

O que queremos dizer com *Learning* é inferir os parâmetros B_0 até B_M que nos forneçam saídas de mapeamento de X para Y consistente com os dados.

Os conceitos básicos da **Logistic Regression** são bastante usados em Deep Learning.

Logistic Regression é um processo de modelar uma probabilidade discreta de um resultado dado os inputs. O mais comum é um binário, sim ou não.

1.2 Interpretação da Logistic Regression

É um dos conceitos mais simples de Machine Learning.

Exemplo para análise de escrita humana. Dada uma foto de uma letra, queremos que a máquina diga que número que é.

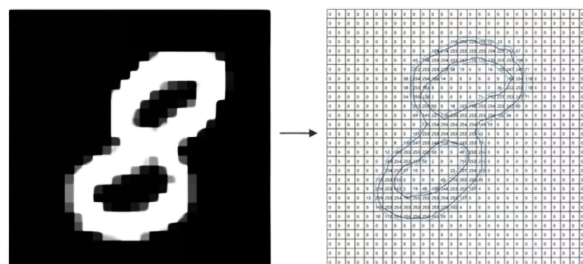


Figura 6: 8 analisado pixel a pixel.

Transformamos a imagem em uma matriz de pixels, onde cada um deles possui um peso em

um espaço. Pegamos esses pixels e eles são o vetor X que foi conversado anteriormente.

Primeiramente, vamos considerar o problema onde os números só podem ser 1s ou 0s para encaixar na Logistic Regression.

Os dados de treinamento são as imagens e o que representam. Seus pixels serão o vetor X . Vamos ponderar os pixels multiplicando pelos parâmetros b_1 até b_M , somamos, conseguimos Z , passamos esse valor pela **Sigmoid Function** e conseguimos um numero entre 0 e 1, que diz a probabilidade de ele ser 0 ou 1.

O que queremos dizer com *learning* é inferir o conjunto de parâmetros b_0 a b_M .

Esse somatório matemático pode ser representado da seguinte maneira:

$$\sum_{m=1}^M x_{im} \times b_m$$

$$x_i \odot b$$

Que significa o produto interno entre x_i e b .

Portanto, temos o valor de nosso z_i dado por:

$$z_1 = b_0 + x_i \odot b$$

Os parâmetros do modelo b são como um filtro para os dados.

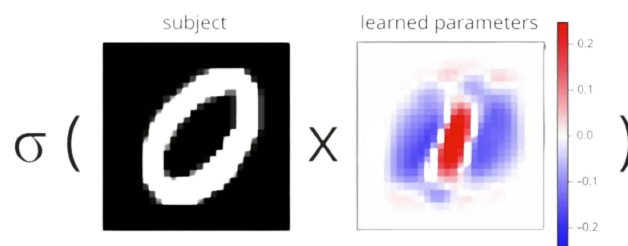


Figura 7: À esquerda, o que foi recebido. À direita, o filtro que será aplicado.

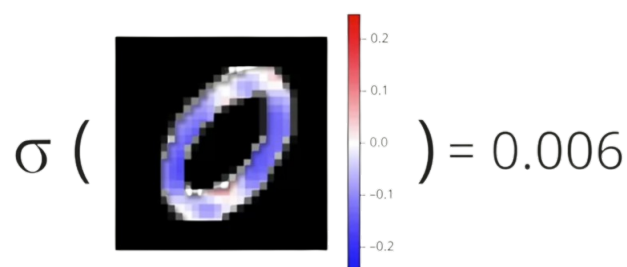


Figura 8: Resultado do filtro; como há mais partes em azul (negativas) o produto interno $x_i \odot b$ resulta em um valor bem próximo de 0.

Esse modelo será usado como base para outros mais complexas como o Deep Learning.

1.3 Motivação para Multilayer Perception

Em **Logistic Regression**, nós possuíamos dados X_i , então fazemos um produto interno $x_i \odot b$, que chamamos de filtro, e ele é somado com b_0 (bias) para conseguir z_i . Então a **Sigmoid Function** converte esse número em uma probabilidade.

Qual é o problema com esse modelo?

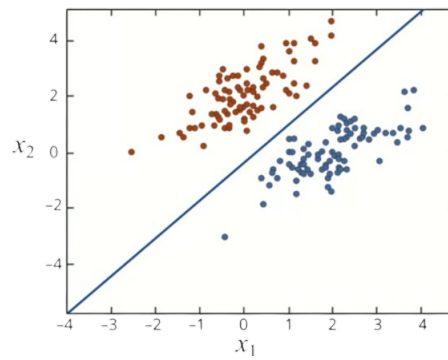


Figura 9: Situação em que é bom usar Logistic Regression.

Um ótimo modelo quando há diferença entre uma classe 0 e 1, que podem ser separados por uma linha. Ele está resolvendo um problema binário.

Problemas as classes 1 e 0 não podem ser separados por uma linha não podem ser resolvidos pelo modelo em questão.

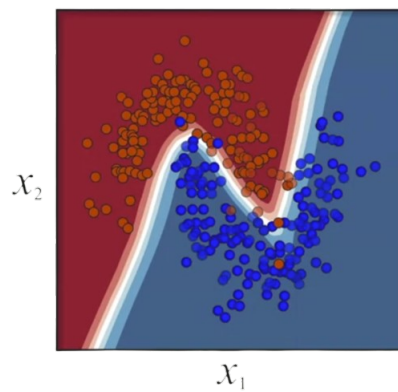


Figura 10: Situação em que é ruim usar Logistic Regression.

Logistic Regression somente é efetivo quando um classificador linear pode ser facilmente distinguido entre classes 1 e 0.

O **Multilayer Perception** é uma extensão do **Logistic Regression** que pode ser usado para esses problemas mais complexos.

2 Conceitos Python Importantes

Listing 1: NumPy

```
import numpy as np

x = np.array([2, 4, 6]) # create a rank 1 array
A = np.array([[1, 3, 5], [2, 4, 6]]) # create a rank 2 array
B = np.array([[1, 2, 3], [4, 5, 6]])

# Indexing/Slicing examples
print(A[0, :]) # index the first "row" and all columns
print(A[1, 2]) # index the second row, third column entry
print(A[:, 1]) # index entire second column

# Arithmetic Examples
C = A * 2 # multiplies every elemnt of A by two
D = A * B # elementwise multiplication rather than matrix multiplication
E = np.transpose(B)
F = np.matmul(A, E) # performs matrix multiplication
# -- could also use np.dot()
G = np.matmul(A, x) # performs matrix-vector multiplication
# -- again could also use np.dot()

# Broadcasting Examples
H = A * x # "broadcasts" x for element-wise
# multiplication with the rows of A
print(H)
J = B + x # broadcasts for addition, again across rows
print(J)

X = np.array([[3, 9, 4], [10, 2, 7], [5, 11, 8]])
all_max = np.max(X)
# gets the maximum value of matrix X
column_max = np.max(X, axis=0)
# gets the maximum in each column -- returns a rank-1 array [10, 11, 8]
row_max = np.max(X, axis=1)
# gets the maximum in each row -- returns a rank-1 array [9, 10, 11]

# In addition to max, can similarly do min. Numpy also has argmax
# to return indices of maximal values
column_argmax = np.argmax(X, axis=0) # note that the "index" here is
# actually the row the maximum occurs for each column

total_sum = np.sum(X)
column_sum = np.sum(X, axis=0)
row_sum = np.sum(X, axis=1)

X = np.arange(16) # makes a rank-1 array of integers from 0 to 15
X_square = np.reshape(X, (4, 4)) # reshape X into a 4 x 4 matrix
X_rank_3 = np.reshape(X, (2, 2, 4))
# reshape X to be 2 x 2 x 4 --a rank-3 array
```

consider as two rank-2 arrays with 2 rows and 4 columns

Listing 2: Matplotlib.PyPlot

```
import numpy as np
import matplotlib.pyplot as plt

# We'll start with a parabola
# Compute the parabola's x and y coordinates
x = np.arange(-5, 5, 0.1)
y = np.square(x)

# Use matplotlib for the plot
plt.plot(x, y, 'b') # specify the color blue for the line
plt.xlabel('X-Axis-Values')
plt.ylabel('Y-Axis-Values')
plt.title('First-Plot:-A-Parabola')
plt.show() # required to actually display the plot
```

Another Matplotlib function you'll encounter is `*imshow*` which is used to display images. Recall that an image may be considered as an array, with array elements indicating image pixel values. As a simple example, here is the identity matrix:

```
import numpy as np
import matplotlib.pyplot as plt

X = np.identity(10)
identity_matrix_image = plt.imshow(X, cmap="Greys_r")
plt.show()

# Now plot a random matrix, with a different colormap
A = np.random.randn(10, 10)
random_matrix_image = plt.imshow(A)
plt.show()
```

3 Multilayer Perceptron

É uma extensão direta da **Logistic Regression**. Ao invés de fazer ela 1 vez, faremos ela k vezes.

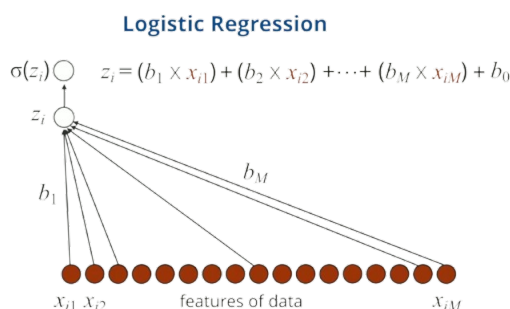


Figura 11: Logistic Regression.

Então, mapearemos nosso vetor X para um vetor de k dimensões chamado de **K Latent Processes/Features**, é chamado de latente porque são representações de coisas que não podemos, a princípio, observar. Depois, esses k recursos são enviados por um modelo de **Logistic Regression** para, no final, gerar uma probabilidade binária para a classificação dos dados.

Os x_{iM} dados que possuímos primeiramente serão mapeados para z_{i1} até z_{ik} processos latentes, para então cada um ser transformado em probabilidades (um vetor de k dimensões) que, por fim, produz um resultado: a probabilidade e que os dados correspondam à classe $y = 1$.

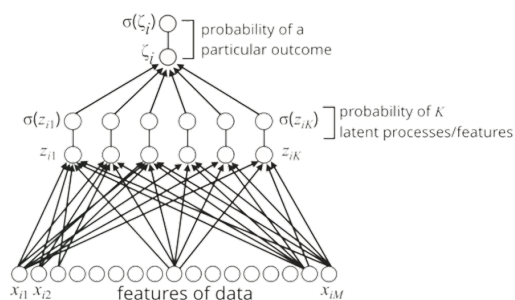


Figura 12: Logistic Regression.

Deve ser visto como uma Logistic Regression sendo aplicada em K Latent Features, ao invés de ser aplicada diretamente nos M componentes dos dados iniciais.

Porque usar isso ao invés da Logistic Regression?



Figura 13: Um filtro para diferentes casos.

Em alguns casos, usar apenas 1 filtro, como é feito na LR, não é o ideal. No exemplo acima, o filtro para o 4 não vai conseguir corresponder a todas as variações nas imagens. As vezes, podemos considerar situações em que temos mais de um filtro para um mesmo número:

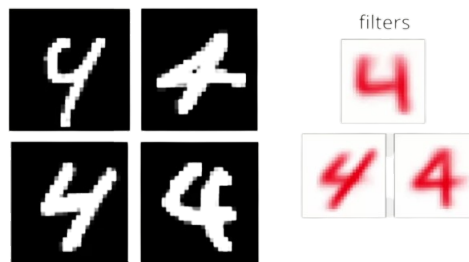


Figura 14: Múltiplos filtros para vários casos.

Voltando para o **Multilayer Perceptron**:

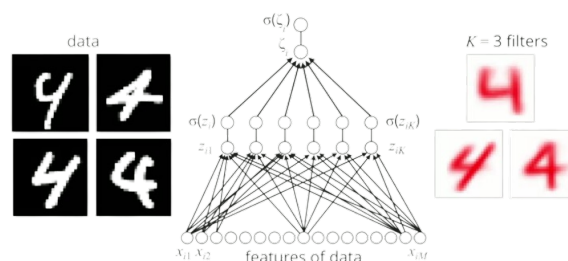


Figura 15: Multilayer Perceptron.

3.1 Modelo Matemático para o Multilayer Perceptron

A matemática está no coração dos modelos neurais que serão estudados. Primeiramente, é importante lembrar que o modelo estudado é uma extensão do Logistic Regression.

Nele, os dados x_{i1} até x_{iM} são mapeados para uma variável z_i por meio do produto interno $x_i \odot b$. Isso gerava um único filtro que era aplicado a todos os dados.

A ideia do **Multilayer Perceptron** é que estamos considerando k filtros e fazemos o produto interno:

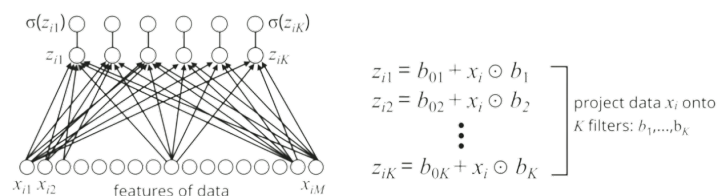


Figura 16: Como é a Multilayer Perceptron.

Então teremos z_{i1} até z_{iK} . Esses recursos serão enviados por meio de uma Logistic Function, que é um mapeamento de um número real para um número entre 0 e 1, e fazemos isso para conseguir as probabilidades dos k produtos latentes. Por fim, todas essas k probabilidades latentes são enviadas por meio de um modelo de Logistic Regression para conseguir um modelo único de filtro c .

Assim, ao invés de conseguir o filtro b , passamos a ter o filtro c dos dados na seguinte fórmula:

$$\zeta_i = c_0 + \sigma(z_i) \odot c$$

Ao invés de usarmos a Logistic Regression para mapear os dados brutos para a probabilidade de um binário, passamos primeiramente por uma etapa intermediária para calcular k recursos latentes que são enviados para um modelo do tipo Logistic Regression para nos dar a probabilidade de um resultado binário.

Essa adição de etapa nos permite considerar limites de decisão não lineares no espaço de recurso, como, por exemplo, o 4 escrito de diversas maneiras diferentes. Os resultados produzidos por ela se mostram mais efetivos.

3.2 Deep Learning

Primeiramente, vamos fazer o mesmo do anterior só que 2 vezes:

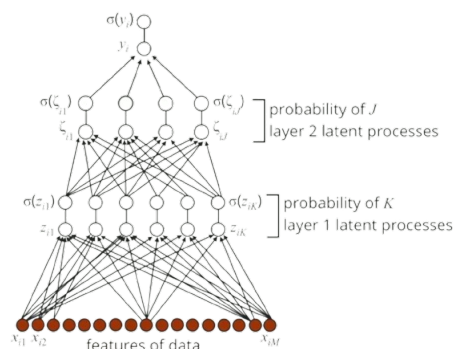


Figura 17: 2 camadas de Multilayer Perceptron.

Basicamente, estamos fazendo o **Multilayer Perceptron** 2 vezes. No final de tudo, ainda conseguimos uma probabilidade. O modelo é mais complicado que o anterior. Dessa forma, conseguimos entender o conceito *Deep* de **Deep Learning**.

Deep Learning é uma forma de **Machine Learning** na qual o modelo possui múltiplas camadas de processos latentes.

Sempre, no topo, há um classificador de **Logistic Regression**, o que nos dá uma probabilidade da saída binária $Y = 1$ ou $Y = 0$.

As camadas intermediárias correspondem a k recursos latentes na camada 1 e j recursos latentes na camada 2. É possível ir a modelos com ainda mais camadas.

Esse clássico modelo chamado de **Multilayer Perception**, também chamado de **Neural Network**, é basicamente constituído por uma sequência de aplicações repetidas da **Logistic Regression**.

Cada componente do **Multilayer Perception** pode ser visto como uma aplicação repetida da **Logistic Regression**.

Porque usaríamos um modelo mais complicado?

Na **Logistic Regression**, poderíamos resolver problemas lineares, como na figura 9. Usando o **Multilayer Perception**, descobrimos que podemos aprender limites de decisão muito mais sofisticados, como na figura 10 (muitas vezes os dados estão distribuídos dessa forma).

Agora, um exemplo para entender sua vantagem: a análise de documentos.

Problema:

Um documento é um conjunto de palavras e x_i representa um conjunto de features para o documento i . Temos um vocabulário composto de V palavras e, para caracterizar um documento, simplesmente vamos contar o número de vezes que cada palavra aparece nele.

Dado o documento e as palavras que estão nele, queremos fazer uma previsão sobre se uma pessoa vai gostar ou não do documento. Like: $y_i = 1$, dislike: $y_i = 0$.

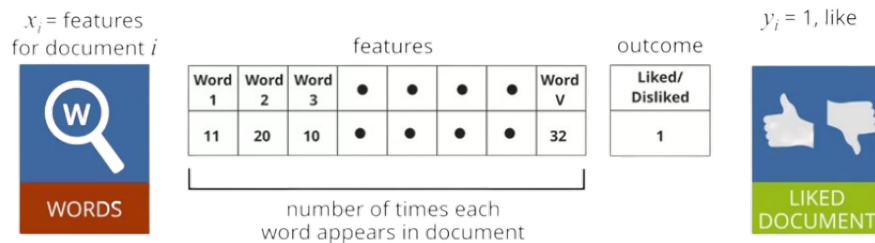


Figura 18: Exemplo documento.

Nós vamos receber dados que serão usados para ensinar o modelo, que seria o mesmo que: aprender os parâmetros do modelo.

Resolução:

Para resolver esse problema, iremos utilizar a **Multilayer Perception**.

Nesse caso dos documentos, os parâmetros b_1, b_2, \dots, b_k serão chamados de tópicos dos documentos. Podemos entender b_1 como palavras características de um determinado tópico, por exemplo, esportes. Portanto, dentro do vocabulário v , provavelmente há um subconjunto de palavras que têm alta probabilidade de aparecer no assunto esportes. Assim, esperamos que o filtro b_1 tenha valores altos para palavras que tenham características de esportes.

Em outro exemplo, b_2 pode estar relacionado com o tópico de história, com valores altos de palavras relacionadas a história e baixos caso contrário.

Uma maneira de pensar nisso é: recebemos um vetor de características x_i que representa a contagem de palavras no documento. Nós vamos calcular o produto interno de x_i com k filtros ou templates. Eles podem ser representados como k processos latentes. No contexto de documentos, podemos pensar neles como tópicos. Assim, o resultado na primeira camada diz: qual é a probabilidade desse tópico de ser sobre esportes? História? Política?

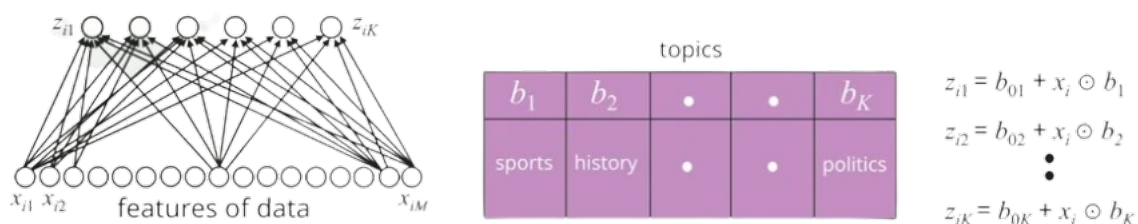


Figura 19: Análise da primeira camada.

Agora, na segunda camada, faremos a mesma coisa que na primeira: pegaremos a saída da primeira camada z_i , que vão passar por uma **Sigmoid Function** para transformá-lo em uma probabilidade e vamos fazer o produto interno com os filtros c_1, c_2, \dots, c_j da segunda camada.

Esses filtros podem enfatizar alguns recursos da camada 1, representando meta-tópicos: uma combinação entre os tópicos da primeira camada. c_1 pode ter alto valor para os tópicos combinados de esportes e história. Se a saída de c_1 é grande, então o documento provavelmente é

uma combinação entre os tópicos esportes e história. c_2 pode ser a combinação entre política e esportes, ou seja, como a política está relacionada aos esportes.

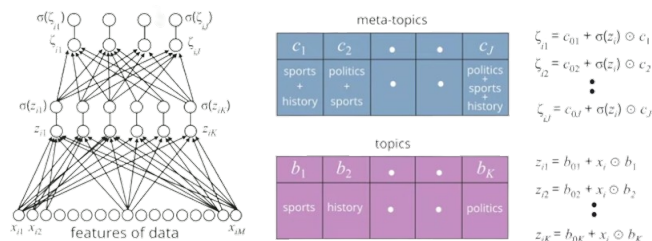


Figura 20: Análise da segunda camada.

Na primeira camada, os filtros estão procurando por tópicos; na segunda, estão buscando meta-topics: combinação desses tópicos.

Agora, na **Logistic Regression** do topo, fazemos uma pergunta: dados os meta-tópicos que parecem ser uma característica aparente dos recursos latentes na camada 2, qual é a probabilidade de a pessoa de interesse gostar ou não do documento?

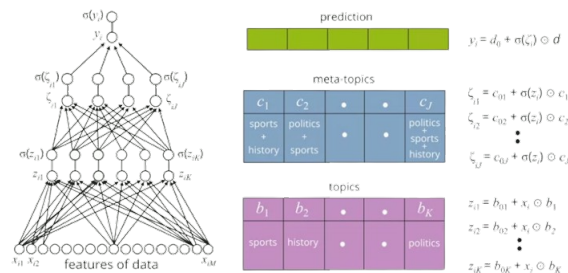


Figura 21: Análise completa com a predição.

Se uma pessoa gosta dos tópicos de esportes e história, nosso filtro d procuraria a situação em que os metatópicos associados à história dos esportes é alta.

3.3 Porque Deep Learning é tão poderoso?

O que ele faz que não é feito facilmente por outros modelos? Usando o exemplo dos documentos: na maioria das vezes, não queremos saber se uma pessoa vai gostar de certo documento, mas sim quantas pessoas podem gostar dele.

As 2 primeiras camadas do modelo, caracterizam o documento e não a pessoa. Consequentemente, se quisermos fazer modelos para várias pessoas, essas 2 primeiras camadas podem ser **reutilizadas** em modelos de várias pessoas. A última camada depende da pessoa.

Portanto, podemos considerar que: as camadas mais pertos dos dados possuem relação com eles e as camadas mais acima possuem relação com a pessoa. Dessa forma, posso transferir os dados de uma pessoa para a próxima quando eu já aprendi os parâmetros da parte de baixo do modelo, então só uso dados da pessoa em interesse na parte superior. Assim, temos um uso muito mais eficiente os dados.

A grande vantagem do **Deep Learning** é que podemos usar vários dados gerais de várias pessoas para construir as bases inferiores do modelo e, para analisar uma pessoa em específico, usamos o seu conjunto de parâmetros relativamente pequeno no topo.

Com a **Logistic Regression**, não podemos fazer a aprendizagem por transferência (**Transfer Learning**) da mesma forma.

3.4 Seleção de modelo

Há vários modelos diferentes para escolher e a melhor forma de escolher o certo é usar a experiência.

Agora, temos 2 modelos à disposição: **Logistic Regression** e **Multilayer Perceptron**.

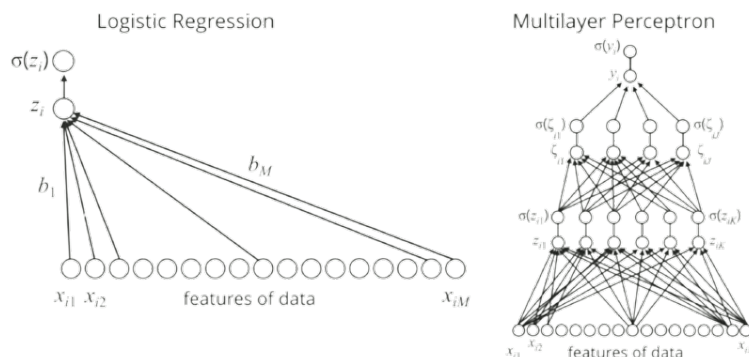


Figura 22: Um filtro para diferentes casos.

Em Machine Learning, essa escolha é chamada de **Bias Variance Trade-Off**.

O primeiro modelo, é bem simples com pouco parâmetros e o segundo é totalmente o oposto. Por isso, em cada treinamento do primeiro modelo, os resultados não terão uma grande variância, dado que é simples. No segundo, há uma maior mudança, dado a sua complexidade.

Portanto, escolhemos modelos mais simples quando não queremos muita variância. Porém essa simplicidade pode trazer uma pior performance.

3.5 História Multilayer Perceptron/Neural Networks

Multilayer Perceptron foi inventado por volta de 1960. Seu uso era limitado, já que o volume de dados era pequeno e o computador não era tão potente. Em 1986, a técnica **Back Propagation** tornou a tarefa de aprender os parâmetros muito mais eficiente. Além disso, tínhamos um poder computacional bem maior. Em 1989, um modelo chamado **Convolutional Neural Network** foi desenvolvido, uma chave para análise de imagens.

No início da década de 90, os modelos foram usados em dados reais. Porém, não estavam tendo uma boa performance, pois não tinham dados suficientes. Em 1995, surgiu a tecnologia **Long Short-Term Memory**, que era muito importante para analisar dados que variam com o tempo. O entusiasmo aumentou.

As tecnologias que estão propulsionando o tópico hoje surgiram há relativamente bastante tempo, antes de 95, não mudaram muito.

Novamente, no início do século, a tecnologia foi para campo novamente. E a história se repetiu: não teve um desempenho tão bom.

Entre 2005 e 2010, nos principais fóruns de Machine Learning, as pessoas não falavam de Multilayer Perceptron ou Neural Networks. Estava muito mal falado.

Assim em 2010, já que isso se tornou algo bem negativo pelos seus insucessos, o nome da tecnologia foi mudado para **Deep Learning**.

Em 2013, houve uma esperança: CNN + GPU + ImageNet. Computação paralela forte, muitos dados de imagens para trabalhar. Então, a CNN foi combinada com a computação moderna, as milhares de imagens disponíveis facilmente e trouxe um avanço fundamental para seu crescimento.

Em 2015, veio o AlphaGo. Ganhar *Go* contra humanos.

Durante essa história, houve vários momentos de empolgação e decepção. Nós precisamos entender quando o Deep Learning é efetivo e quando precisamos usar modelos mais simples.

"All things being equal, the simplest solution tends to be the best one." - William of Ockham.

4 Convolutional Neural Networks

Nessa seção, focaremos em imagens. A técnica de CNN possui um desempenho em análise de imagens maior que a dos humanos. Os dados analisados serão as imagens.

IMG1 AQUI

Cada imagem compõe exemplos de High-Level Motifs que são descolados para outros locais dentro da imagem. Nenhuma imagem é composta por todos os Motifs, mas cada imagem é caracterizada por um subconjunto deles e a localização deles muda em cada imagem.

As imagens na vida real possuem coisas que repetem: bordas, cantos, texturas e formatos. Isso é uma característica de todas as imagens naturais. Queremos explorar isso no contexto da CNN.

a imagem geral é composta por um subconjuntos de Motifs que repetem entre as imagens.

IMG2 AQUI

Como podemos construir um modelo que captura essa hierarquia?

4.1 Convolutional Example

IMG3 AQUI

Nos queremos começar da parte de baixo da hierarquia e procurar por um dos elementos atômicos. Para isso, vamos mover esse triângulo em cada local da imagem. Depois, vamos calcular uma correlação que mede o quão semelhante uma região local da imagem é do nosso elemento atômico. Isso chamaremos de filtro.

IMG4 AQUI

Assim, ao passar esse triângulo em todos os pontos possíveis da imagem, geramos o que é chamado de **feature map**, que contém a saída do processo. O que está em vermelho mostra uma alta correlação e em azul baixa. O processo em que passamos o triângulo em toda a imagem possui um nome: convolução (convolution). Ela se manifesta ao passar o filtro (o elemento atômico) em cada local na imagem.

Nos temos múltiplos elementos atômicos, por isso iremos passar cada um desses filtros fundamentais por meio de um Two Dimensional Convolutional Process. Nós vamos conseguir um feature map para cada elemento atômico. Um mapa que reflete o grau de correspondência do elemento atômico na imagem. Portanto, conseguimos uma pilha de feature maps.

IMG5 AQUI

Como podemos identificar os submotifs? Vamos repetir o processo, no qual vamos convolver (convolve) filtros para cada localização bidimensional no feature map. Os filtros agora são várias pilhas de elementos atômicos.

IMG6 AQUI

Se um sub-motif está presente, é esperado que os respectivos feature maps associados e seus elementos atômicos tivessem uma grande amplitude próxima um do outro no espaço.

Agora, repetimos o processo: vamos fazer a convolution dos filtros da segunda camada com os feature maps da primeira. Isso vai produzir um novo conjunto de feature maps. Eles nos dirão onde cada um dos sub-motifs está na imagem. Então, repetimos esse processo novamente.

IMG7 AQUI

Por fim, conseguimos construir um feature map para a terceira camada.

Agora, finalmente, queremos tomar uma decisão de classificação. Ela será baseada no feature map do topo do modelo. O que é realmente importante notar sobre essa Deep Architecture é que ela se manifesta numa aplicação repetida do mesmo processo.

No início, pegamos uma imagem e a convolvemos ela com um conjunto de blocos de construção fundamentais. Depois, conseguimos um feature map, o grau de como cada um dos elementos se manifesta na imagem. Isso nos resulta em uma pilha de feature maps.

Depois, os filtros da segunda camada procuram elementos da primeira camada que estão, simultaneamente, próximos uns dos outros. Assim, novamente convolvemos o feature map da primeira camada com os filtros da segunda, e conseguimos outra pilha de feature maps.

Por fim, pegamos os filtros dos Motifs e convolvemos com o feature map da segunda camada, o que nos gera o ultimo feature map e, com base nele, decidiremos a classificação da imagem.

No mundo real, os formatos não são tão simples quanto nesse exemplo. A pergunta é: como projetamos modelos que aprendam como construir um modelo desses para imagens reais?

4.2 Modelo Matemático

Tudo o que foi aprendido no exemplo ilustrado é essencialmente transferido diretamente.

Para aprender um modelo, iremos analisar milhões de imagens. O feature map reflete a intensidade com que esse elemento se manifesta na imagem. Agora, vamos fazer a mesma coisa; porém, não conhecemos os elementos atômicos. Vamos representar eles por parâmetros que vamos aprender.

IMG8 AQUI

Em um primeiro momento, vamos assumir que sabemos os valores ϕ_1 a ϕ_k . Se soubermos isso, podemos construir um feature map.

$$M_n = f(I_n; \phi_1, \dots, \phi_k)$$

M_n representa o feature map na camada 1 para a n -ésima imagem.

Então, repetimos o processo:

$$L_n = f(M_n; \psi_1, \dots, \psi_k)$$

Os valores ψ_1 a ψ_k representam uma pequena pilha de imagens que querem corresponder à pilha de feature map da camada abaixo. Se eu soubesse quem são eles, eu poderia fazer a convolução. O que vamos fazer é: aprender esses parâmetros.

IMG9 AQUI

Podemos repetir novamente essa ideia. No topo, teremos:

$$G_n = f(L_n; \omega_1, \dots, \omega_k)$$

IMG10 AQUI

Dessa forma, podemos criar um classificador que fornecerá um rótulo para essa imagem:

$$\ell(G_n; W)$$

O rótulo é uma função das features da parte superior e o parâmetros do classificador.

A diferença entre o exemplo que foi usado anteriormente é que, ao invés de assumir as formas atômicas básicas, temos os parâmetros. O processo de filtragem convolucional é exatamente o mesmo.

O classificador poderia assumir uma forma de um Multilayer Perceptron ou um Logistic Regression. Em qualquer um dos casos, ele é caracterizado por alguns parâmetros W .

4.3 Como o modelo aprende?

IMG11 AQUI

A maneira com que aprendemos é supor que possuímos um acesso a um grande conjunto de dados rotulados. Uma das principais coisas que fez essa tecnologia tão poderosa foi o acesso a imensas quantidades de dados. Os dados são representados pelo par I_n como a n -ésima imagem e y_n é o rótulo da imagem.

Para simplicidade, vamos supor que os labels são binárias, mas não precisariam ser.

No terceiro ponto, o que vamos fazer é constituir o que chamamos de uma função de energia, representado por E , que vai computar o que chamamos de perda entre o rótulo verdadeiro y_n e o do modelo, ℓ_n .

No topo, nós conseguimos prever o ℓ_n e, como estamos lidando com dados em que sabemos o rótulo, podemos calcular uma perda. Gostaríamos de fazer essa perda o menor possível.

Queremos tentar aprender ou identificar ou calcular os parâmetros ϕ , ψ , ω e W que minimizam essa perda. A perda é a diferença entre a precisão dos rótulos previstos e os verdadeiros.

Isso é o que queremos dizer com Machine Learning. À esquerda, está a máquina. A parte do aprendizado é pegar os dados e tentar aprender os parâmetros, de modo que a diferença entre ℓ e o rótulo y_n seja pequena.

Porque isso é difícil?

Um dos principais aspectos disso é que o número de parâmetros que temos que estimar é enorme, muito grande. Portanto, tentar estimá-los é difícil.

IMG12 AQUI

Olhando para esse gráfico, com apenas 2 parâmetros, é claro perceber que é muito difícil se mover nesse espaço para otimizar os parâmetros, pois caímos em vários máximos locais. Foi preciso de várias décadas de estudo e pesquisa para descobrir métodos de fazer isso de forma eficaz.

Vantagens da Hierarquia de Features Se apenas os recursos do topo da arquitetura são usados para fazer a decisão de classificação, porque não podemos simplesmente criar um modelo com base nos motivos da camada superior? Qual é a vantagem dessa arquitetura profunda?

Se quisermos construir baseado apenas no topo, eles seriam aprendidos independentemente, ou seja, não saberíamos que esses filtros compartilham certas partes de sua estrutura.

A ideia é que: ao compartilhar as estruturas os diferentes elementos dos filtros nas várias camadas, manifestada pelo fato de que todos eles são compostos de versões alteradas de blocos de construção mais fundamentais, podemos usar nossos dados de maneira mais eficiente, já que o conhecimento de um motivo pode melhorar o conhecimento de outro motivo por meio da subestrutura compartilhada que eles possuem.

Aprendizado é manifestado ao coletar grandes quantidades de dados rotulados.

In general terms, the basic steps to do learning are:

Obtain a large set of labeled data

Determine the loss function, which computes loss between true label and model label

Determine parameters that minimize the sum over loss