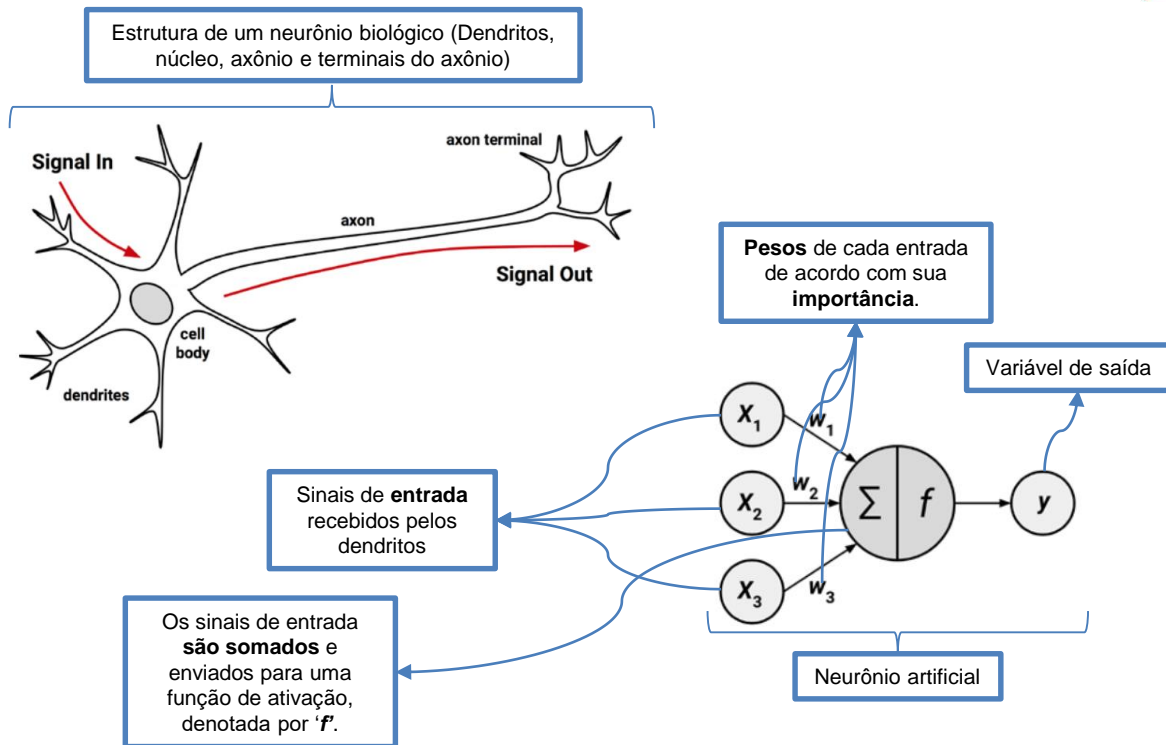


Aplicação de Técnicas de Aprendizagem de Máquina utilizando R

Prof. Mário de Noronha Neto

O material utilizado neste curso foi elaborado pelos professores Mario de Noronha Neto (IFSC) e Richard Demo Souza (UFSC)

Redes Neurais

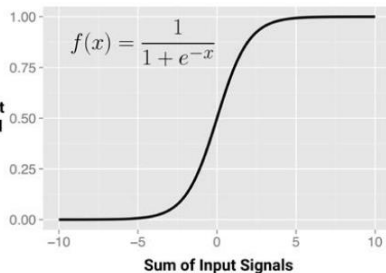


Redes Neurais - Funções de ativação

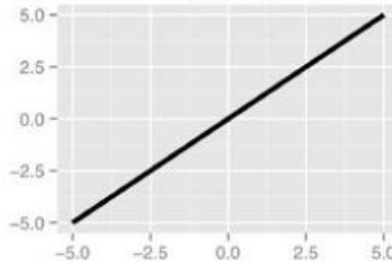
Função de ativação: Uma função que tem a finalidade de processar a informação de entrada da rede antes de encaminhá-la adiante.

$$y(x) = f \left(\sum_{i=1}^n w_i x_i \right)$$

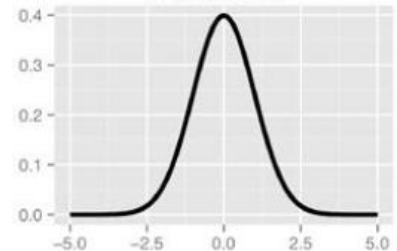
Sigmoid



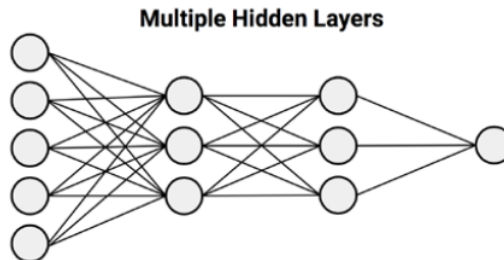
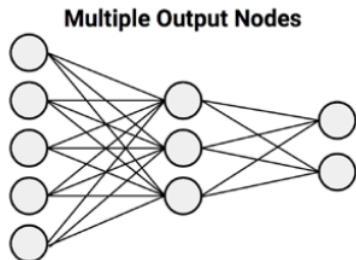
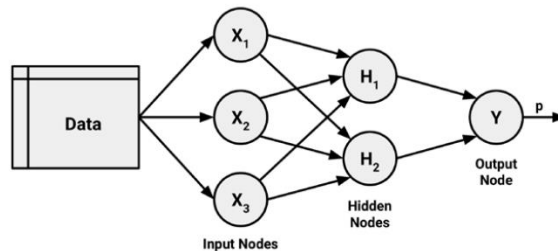
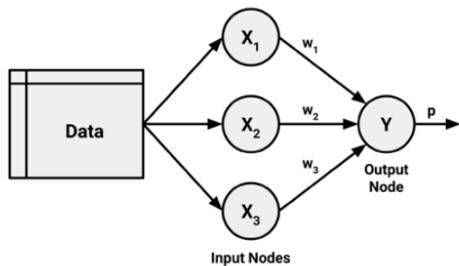
Linear



Gaussiana



Redes Neurais - Topologias



Exemplo: Modelando a resistência de concreto.



Passo 1: Coleta de dados

Dataset utilizado:

O *dataset* utilizado neste exemplo pode ser encontrado no repositório de ML da UCI (<http://archive.ics.uci.edu/ml>). Este conjunto possui 1030 exemplos de concretos com oito características descrevendo os componentes utilizado em sua mistura, tais como cimento, areia e água, todos em kg por metro cúbico. A combinação dessas características está relacionada à resistência do concreto (variável *strength*)

Passo 2: Explorando e preparando os dados

```
concrete <- read.csv("concrete.csv")  
str(concrete)
```

```
'data.frame':  1030 obs. of  9 variables:  
 $ cement      : num  141 169 250 266 155 ...  
 $ slag        : num  212 42.2 0 114 183.4 ...  
 $ ash         : num  0 124.3 95.7 0 0 ...  
 $ water       : num  204 158 187 228 193 ...  
 $ superplastic: num  0 10.8 5.5 0 9.1 0 0 6.4 0 9 ...  
 $ coarseagg   : num  972 1081 957 932 1047 ...  
 $ fineagg     : num  748 796 861 670 697 ...  
 $ age         : int   28 14 28 28 28 90 7 56 28 28 ...  
 $ strength    : num  29.9 23.5 29.2 45.9 18.3 ...
```

Passo 2: Explorando e preparando os dados

Utilizando a função *summary* e *hist.*, podemos observar que as características possuem escalas bem distintas e que distribuições distantes de uma distribuição normal. Desta forma, a normalização dos dados para uma escale entre 0 e 1 pode ser apropriada antes de realizar o treinamento. **Lembrando que qualquer transformação feita antes do processo de treinamento deve ser desfeita futuramente para obter os dados na unidade original de medida.**

```
> normalize <- function(x) {  
  return((x - min(x)) / (max(x) - min(x)))  
}  
  
> concrete_norm <- as.data.frame(lapply(concrete, normalize))  
  
> summary(concrete_norm$strength)  
  
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   
 0.0000 0.2664  0.4001  0.4172  0.5457 1.0000   
  
> summary(concrete$strength)  
  
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   
 2.33  23.71  34.44  35.82  46.14  82.60
```


Passo 2: Explorando e preparando os dados

Como o *dataset* extraído do arquivo .csv já está organizado de forma aleatória, podemos separar as sequências de teste e treinamento como apresentado a seguir. Utilizaremos 75% dos dados para o treinamento e 25% para testes.

```
> concrete_train <- concrete_norm[1:773, ]  
> concrete_test  <- concrete_norm[774:1030, ]
```

Passo 3: Treinando o modelo

Neural network syntax

using the `neuralnet()` function in the `neuralnet` package

Building the model:

```
m <- neuralnet(target ~ predictors, data = mydata,  
               hidden = 1)
```

- `target` is the outcome in the `mydata` data frame to be modeled
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` specifies the data frame in which the `target` and `predictors` variables can be found
- `hidden` specifies the number of neurons in the hidden layer (by default, 1)

The function will return a neural network object that can be used to make predictions.

Making predictions:

```
p <- compute(m, test)
```

- `m` is a model trained by the `neuralnet()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier

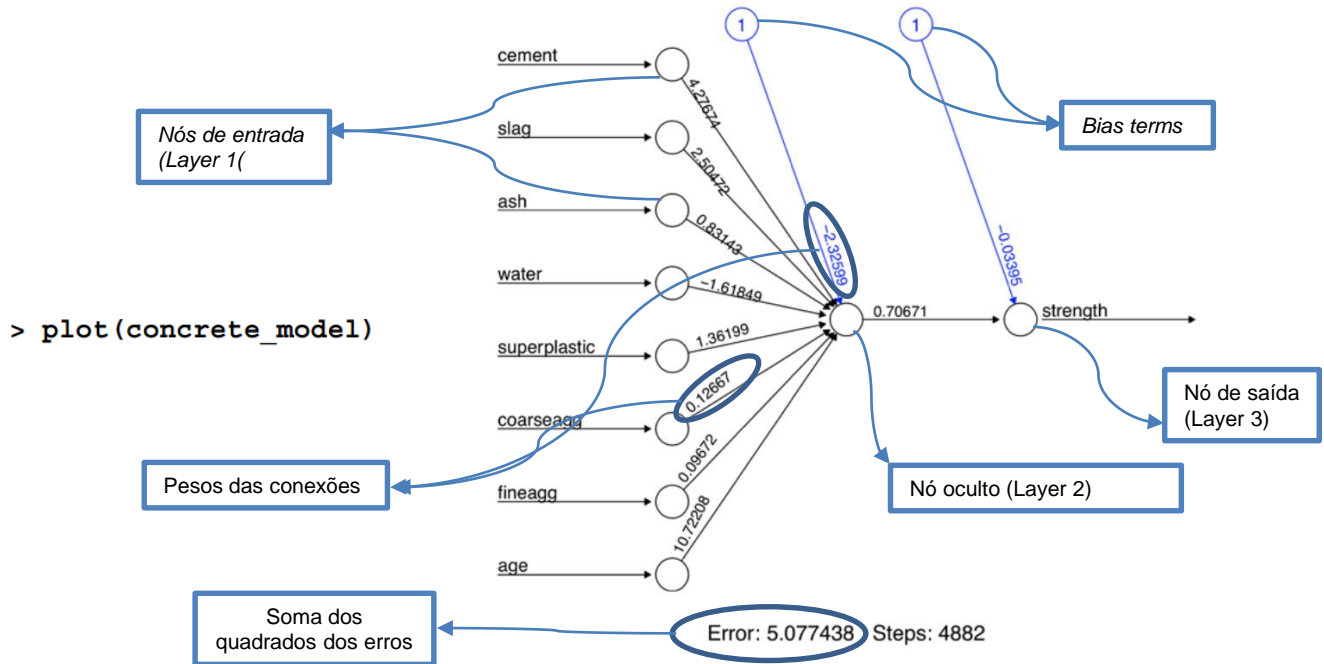
The function will return a list with two components: `$neurons`, which stores the neurons for each layer in the network, and `$net.result`, which stores the model's predicted values.

Example:

```
concrete_model <- neuralnet(strength ~ cement + slag  
                           + ash, data = concrete)  
model_results <- compute(concrete_model,  
                         concrete_data)  
strength_predictions <- model_results$net.result
```

Passo 3: Treinando o modelo

```
> concrete_model <- neuralnet(strength ~ cement + slag  
+ ash + water + superplastic + coarseagg + fineagg + age,  
data = concrete_train)
```



Passo 4: Avaliando o modelo

```
> model_results <- compute(concrete_model, concrete_test[1:8])
```

```
> predicted_strength <- model_results$net.result
```

```
> cor(predicted_strength, concrete_test$strength)
```

[,1]

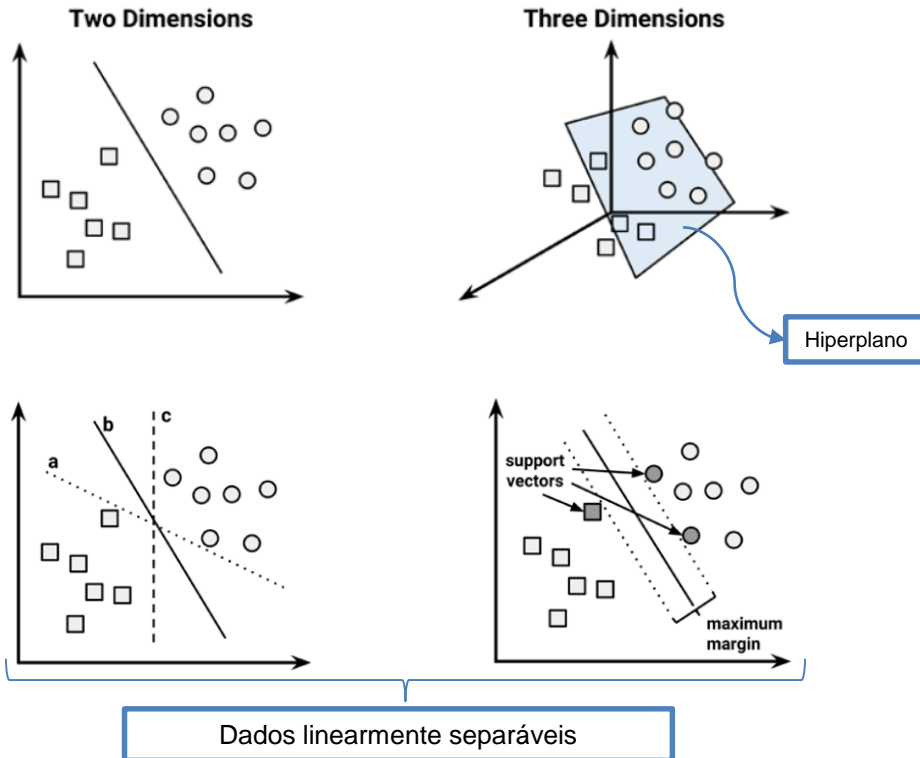
[1,] 0.806465576

Retorna duas componentes \$neurons, a qual armazena os valores do neurônio em cada camada da rede, e \$net.result, o qual armazena os valores previstos.

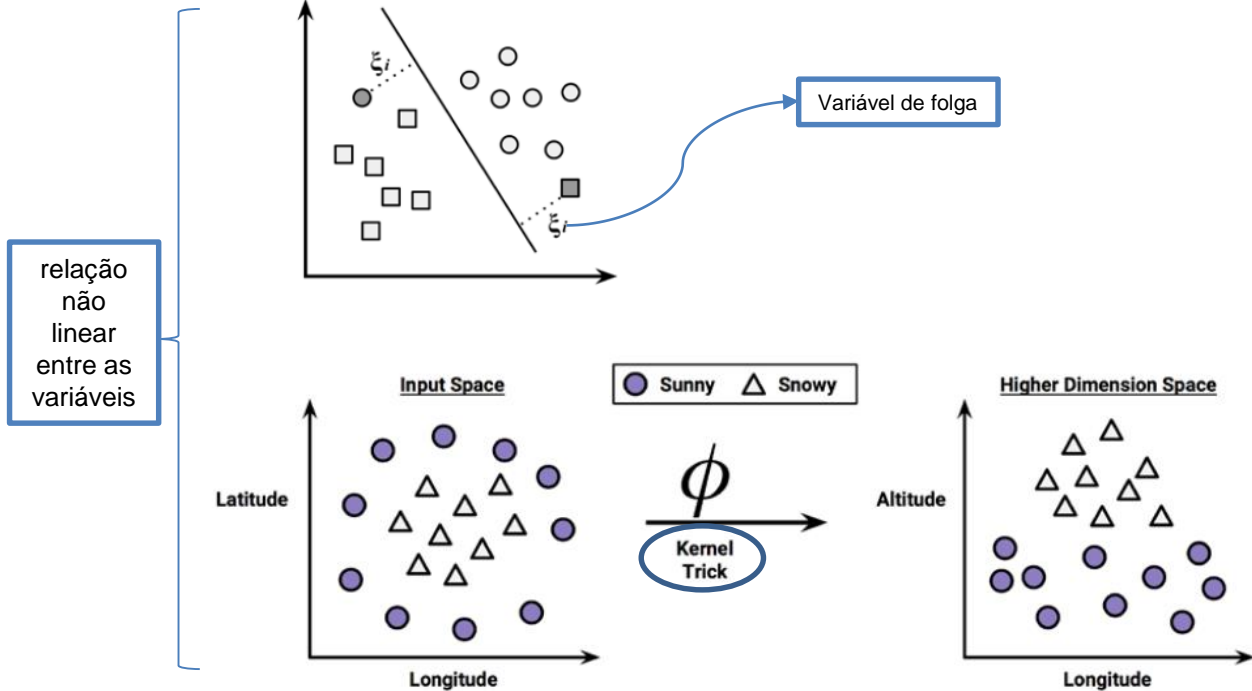
Como este exemplo é um problema de predição numérica, podemos utilizar a função `cor()` para avaliar o desempenho da NN.

Este valor indica que existe uma correlação relativamente forte entre as variáveis. Uma NN com topologia mais complexa é capaz de aprender conceitos mais difíceis.
Tente aumentar o número de camadas e nós ocultos e avalie o resultado.

Support Vector Machines - SVM



Support Vector Machines - SVM



Exemplo: OCR com SVM

Passo 1: Coleta de dados

Dataset utilizado:

O *dataset* utilizado neste exemplo pode ser encontrado no repositório de ML da UCI (<http://archive.ics.uci.edu/ml>). Este conjunto possui 20000 das 26 letras do alfabeto inglês, contendo 20 variações de formas e distorções, conforme exemplo a seguir. Neste exercício, vamos considerar que já desenvolvemos o algoritmo para particionar o documento em regiões retangulares, cada uma consistindo em um único caractere (glifo). Quando os glifos são escaneados pelo computador, eles são convertidos em pixels e 16 características são armazenadas, tais como as dimensões horizontais e verticais do glifo, a proporção de pixels pretos e a posição horizontal e vertical média dos pixels.



Passo 2: Explorando e preparando os dados

```
> letters <- read.csv("letterdata.csv")  
  
> str(letters)  
  
'data.frame':  20000 obs. of  17 variables:  
 $ letter: Factor w/ 26 levels "A","B","C","D",...  
 $ xbox  : int  2 5 4 7 2 4 4 1 2 11 ...  
 $ ybox  : int  8 12 11 11 1 11 2 1 2 15 ...  
 $ width : int  3 3 6 6 3 5 5 3 4 13 ...  
 $ height: int  5 7 8 6 1 8 4 2 4 9 ...  
 $ onpix  : int  1 2 6 3 1 3 4 1 2 7 ...  
 $ xbar   : int  8 10 10 5 8 8 8 8 10 13 ...  
 $ ybar   : int  13 5 6 9 6 8 7 2 6 2 ...  
 $ x2bar  : int  0 5 2 4 6 6 6 2 2 6 ...  
 $ y2bar  : int  6 4 6 6 6 9 6 2 6 2 ...  
 $ xybar  : int  6 13 10 4 6 5 7 8 12 12 ...  
 $ x2ybar : int  10 3 3 4 5 6 6 2 4 1 ...  
 $ xy2bar : int  8 9 7 10 9 6 6 8 8 9 ...  
 $ xedge  : int  0 2 3 6 1 0 2 1 1 8 ...  
 $ xedgey : int  8 8 7 10 7 8 8 6 6 1 ...  
 $ yedge  : int  0 4 3 2 5 9 7 2 1 1 ...  
 $ yedgey : int  8 10 9 8 10 7 10 7 7 8 ...
```


Passo 2: Explorando e preparando os dados

O SVM requer que todas as características de entrada sejam numéricas. Com neste caso todas as características de entrada são inteiros, não há necessidade de converter fatores em números, entretanto suas escalas estão um pouco diferentes. Esta etapa não será feita aqui, pois o pacote de SVM utilizado faz isto automaticamente! Neste caso, nos resta fazer a divisão dos dados em sequências de treinamento e teste e podemos pular direto para a etapa de treinamento do modelo

```
> letters_train <- letters[1:16000, ]  
> letters_test  <- letters[16001:20000, ]
```

Passo 3: Treinamento do modelo

Support vector machine syntax

using the `ksvm()` function in the `kernlab` package

Building the model:

```
m <- ksvm(target ~ predictors, data = mydata,  
          kernel = "rbfdot", C = 1)
```

- **target** is the outcome in the **mydata** data frame to be modeled
- **predictors** is an R formula specifying the features in the **mydata** data frame to use for prediction
- **data** specifies the data frame in which the **target** and **predictors** variables can be found
- **kernel** specifies a nonlinear mapping such as "**rbfdot**" (radial basis), "**polydot**" (polynomial), "**tanhdot**" (hyperbolic tangent sigmoid), or "**vanilladot**" (linear)
- **C** is a number that specifies the cost of violating the constraints, i.e., how big of a penalty there is for the "soft margin." Larger values will result in narrower margins

The function will return a SVM object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "response")
```

- **m** is a model trained by the `ksvm()` function
- **test** is a data frame containing test data with the same features as the training data used to build the classifier
- **type** specifies whether the predictions should be "**response**" (the predicted class) or "**probabilities**" (the predicted probability, one column per class level).

The function will return a vector (or matrix) of predicted classes (or probabilities) depending on the value of the type parameter.

Example:

```
letter_classifier <- ksvm(letter ~ ., data =  
  letters_train, kernel = "vanilladot")  
letter_prediction <- predict(letter_classifier,  
  letters_test)
```

Linear

Passo 4: Avaliando o desempenho

```
> letter_predictions <- predict(letter_classifier, letters_test)
```

```
> table(letter_predictions, letters_test$letter)
```

letter_predictions	A	B	C	D	E
A	144	0	0	0	0
B	0	121	0	5	2
C	0	0	120	0	4
D	2	2	0	156	0
E	0	0	5	0	127

Exemplo: 5 vezes em que a letra E foi confundida com a letra D