**Machine Learning e Data Science com Python**

**Python:**

* Instalar o Python
* Jupyter notebook pelo gerenciador de pacote pip:
  + *pip3 install jupyter*
* Chamar esse app através do comando e criar um novo notebook
  + jupyter-notebook
* Copia do python em pasta diferente: pip3 install virtualenv (bibliotecas separadamente em cada copia)
* Python possui tipagem dinamica e forte. Forte significa que não faz conversão automatica
* Booleanos: True e False
* Operador a//b realiza a divisão truncada
* Operadores lógicos: and, or, not
* Função para pegar o tamanho de string: len(a)
* Exponenciação: a\*\*b
* String format exemplo: print('Maria tem %d anos e %d amigos' % (idade, idade))
* Função para inputar dados (sempre vem String): input('mensage: ')
* if condicao :

bloco

elif condicao:

bloco

else:

bloco

* Python não tem chaves, nem ; ele trabalha puramente com identação
* for variavel in elemento :

bloco

* Criar lista basta : lista = [ ] , podendo colocar tipos diversos. É mutável.
* Função reverso da lista: lista.reverse()
* lista.copy() = lista[:] (irão apontar para lugares distintos de memoria)
* lista.pop(indice) = remove por posição , se não passar nada, vai tirar o último
* lista.remove(elemento) = remove por elemento (obrigatorio passar um elemento)
* for i in range(inicio,fim,passo): ele percorre todos dentro do intervalo proposto
* No print, se colocar um segundo parametro end = ‘ ’ vai indicar o separador entre um print e outro dentro de um for por exemplo (print (variavel , end= ‘+’) indica que deve ser colocar um + no fim de cada elemento)
* Uma tupla é uma lista Imutável. tupla =(1,2,3)
* Para criar um conjunto: conj = set() ou conj = {1,2,3} . Não tem indexação
* União de conjuntos: conj1.union(conj2)
* Interseção de conjuntos: conj1.intersection(conj2)
* Diferença de conjuntos: conj1.difference(conj2)
* Dicionario de dados: d = {‘variavel’: valor, ‘variavel2’:valor2} , utiliza chave como índice. Quando a chave ja existe, o valor é atualizado, se colocar uma nova, ele irá criar.
* Para testar se um item existe num dicionario basta : ‘variavel’ in dicionario
* Criar função : def nome(parametros):
* Criar funcao com numero ilimitado de parametros : def funcao(\*args): , é possível não passar nenhum argumento
* Para passar um valor default para algum parametro da funcao basta fazer :

def paramDefault(nome='desconhecido'):

* É possível passar uma função como argumento
* Para criar funções anonimas : lambda in: ....
* Quando tem uma função com um numero de parametros, por exemplo (a,b,c) , se for passado para essa funcao um argumento do tipo (\*[1,2,3]) o compilador ira desempacotar e funcionará corretamente.
* Para importar módulos basta fazer import nomedomodulo
* Para importar modulos de outro pacote basta colocar o subdiretorio referenciado
* Para renomear um import basta fazer um import nome as nome2
* **Import estatico de metodos: from nomedomodulo import nomedafuncao**
* Se houver um import estático de funcoes com mesmo nome, a funcao escolhida sera a de ultimo import
* Abrir/ criar arquivo = open(nomedoarquivo, tipodeoperacao). Na operacao pode passar ‘r’ ,‘w’, ‘a’. Se o arquivo não existir ele cria. Se já existir e for passado o ‘w’ ele irá SOBRESCREVER! Para aproveitar o que já tem no arquivo , passar o parametro ‘a’. Pode colocar ‘r+’ ou ‘w+’ para combinar e reaproveitar
* Para criar uma classe : class nomedaclasse
* Construtor de classe: def \_\_init\_\_(self):
* Todas as funçoes de uma classe tem como primeiro parametro ela mesma (self)
* Para instanciar classe basta chama-la, nao precisa do new
* Para criar herança, basta passar como parametro na classe: class nome(pai):
* filter(lambda, lista) retorna os valores de acordo com o filtro passado
* map(lambda,lista) irá realizar alguma operacao em cada elemento da lista e retorna esses novos valores
* Para mudar o locale:
  + import locale
  + locale.setlocale(locale.LC\_ALL, ‘pt\_BR’)

**Machine Learning:**

* Teste de Turing: Jogo de imitação
* Conjunto de treinamento (experiencia) , cada objeto é uma amostra ou padrão
* O atributo de saída é chamado de meta ou alvo: seus valores sao estimados atraves dos dados de entradas ou preditivos
* Dados imperfeitos são chamados de ruido, dados ausentes.
* Conjunto de atributos de entrada = vetor de caracteristicas
* Escala de atributos = Nominal ou Ordinal (qualitativo) , Intervalar ou Racional (quantitativo)
* Aprendizado Indutivo = Supervisionado (Preditivo) ou Não Supervisionado (Descritivo)
* Supervisionado = possui um supervisor externo, já existe um resultado certo pra cada exemplo. O algoritmo vai treinar
  + Classificação : rotulos discretos (bom ou ruim; sim ou nao...). Atribuir rótulo para nova amostra
  + Regressão: rotulos continuos ( peso, altura..)
* Não supervisionado = não se conhece a saída, exploram as regularidades nos dados
  + Agrupamento: dadois agrupados de acordo com sua similaridade (clusters)
  + Sumarizaçao: descricao compacta para dados
  + Associação: padroes frequentes de associacoes entre atributos
* Semi-supervisionado = utiliza dados rotulados (normalmente uma pequena quantidade) e não rotulados para o treinamento (maior quantidade). Os não rotulados são mais baratos e mais faceis de ser obtidos
* Por Reforço: o algoritmo descobre por tentativa e erro as ações que geram maiores recompensas. Possui 3 componentes:
  + Agente = tomador de decisões
  + Ambiente = Tudo com o qual o agente interage
  + Ações = o que o agente pode fazer
* Técnicas: Redes Neurais, árvores de decisões , KNN, K-Means, Redes Bayesianas, Máquina de Vetor de Suporte (SVMs), Regressão Linear..
* Site com Dataset = <https://archive.ics.uci.edu/ml/datasets.html>
* Iris = Dataset famoso e didatico <https://archive.ics.uci.edu/ml/datasets/iris> -> <https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>
* Regressao linear estatistica: Variavel explicada Y , Variavel explicativa X, Valores reais B -> Y = B \* X
  + Regressao linear multipla: OLS (minimos quadrados)
  + Redes neurais
  + Knn
  + Svm
* Método dos mínimos quadrados (OLS): otimização que busca o melhor ajuste para minimizar a soma dos quadrados dos erros encontrados
* KNN : supervisionado, simples, classificar atraves de exemplos de treinamento que estão mais próximos no espaço de características. Em alguns casos apresenta ótimos resultados. Porém pode ser um processo computacionalmente complexo. Tem 3 passos a seguir:
  + Obter um conjunto de exemplos de treinamento
  + Definir uma métrica para calcular a distância entre os exemplos de treinamento
  + Definir o valor de K = numero de vizinhos mais proximos (numero impar para desempatar). A quantidade default é , onde n é o numero de amostras de treinamento.
    - Se K é muito pequeno, a classificaçao fica sensível a pontos de ruído
    - Se K é muito grande, pode incluir elementos de outras classes
  + Calcular a distancia entre o exemplo desconhecido e os outros elementos do grupo de treinamento
    - Pode ser pela distancia euclidiana:

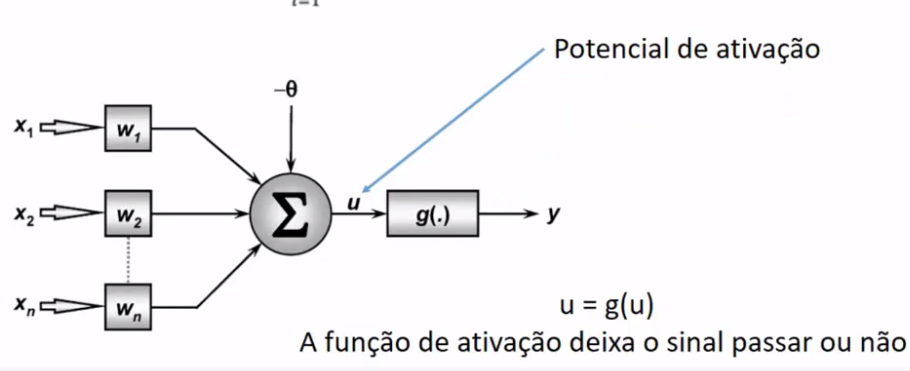
* Distancia Manhattan ou distancia retangular :

* + Identificar quem sao os K vizinhos mais proximos
  + Utilizar o rotulo da classe dos mais proximos para determina o rotulo de classe do desconhecido (votacao majoritaria)
  + KNN também pode ser utilizado para REGRESSÃO (saída contínua)
    - Ao invés de selecionar por votação majoritária, faz-se a média aritmética
    - Para medir o erro quadratico médio, pode utilizar a função mean\_squared\_error do módulo sklearn.metrics. Por exemplo: mean\_squared\_error(esperado,obtido)
* Redes Neurais Artificiais: modelos computacionais inspirados no sistema nervoso central para aprendizado de máquina. Podem coordenar diversos graus de liberdade durante a execução de tarefas complicadas sem que tenham que desenvolver um modelo matemático específico. Vários tipos, entre eles Rede Perceptron e Rede Adalaide
  + Sinais de entrada (x1,x2...) possuem pesos sinápticos (w1,w2...). A entrada pro neurônio é o produto do sinal pelo seu peso.
  + Função agregadora recebe os sinais e realiza a soma dos produtos
  + O neurônio deixa passar ou inibe um determinado sinal ou pode até alterar o valor da saída conforme entrada.
  + Existe um Limiar de Ativação (ou Bias) -Ɵ que é uma constrante que vai indicar um limiar para o sinal passar ou não.
  + g é a função de ativação: depende do problema.
    - Função degrau (se u >= 0 -> 1 senão -> 0)
    - Função degrau bipolar (se u > 0 -> 1, se u = 0 -> 0 , senão -> -1)
    - Função Logística: o valor de β vai depender do problema, são necessários testes.

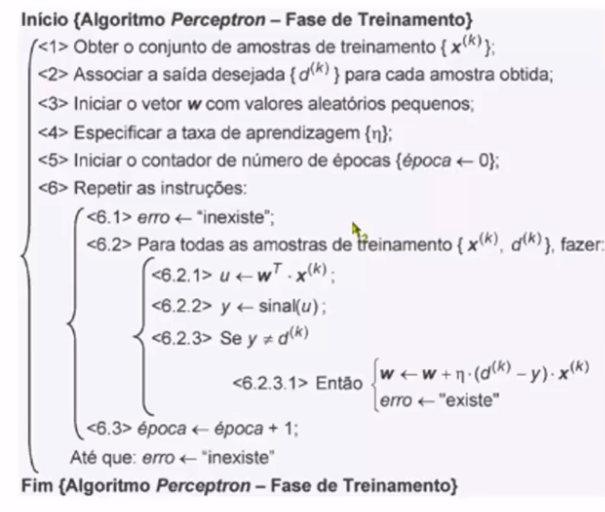
* + - Função Tangente Hiberbólica:

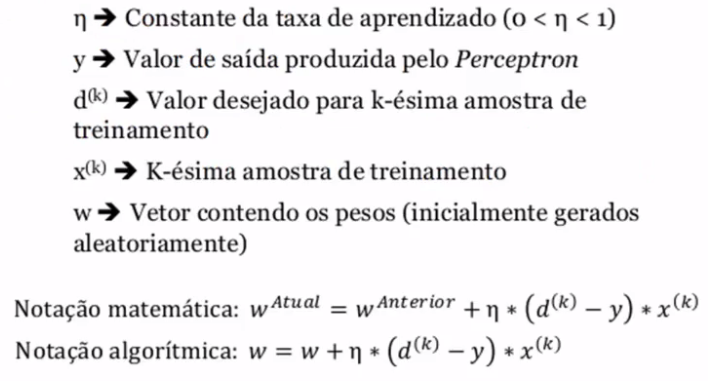
* + - Função Gaussiana: c é o centro da Função Gaussiana e σ é o desvio padrão

* + u = potencial de ativação:

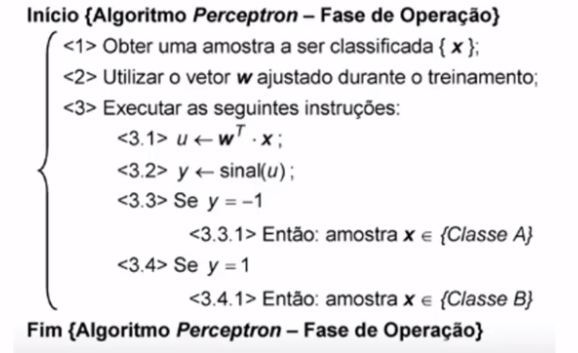


* Rede Perceptron: simples, apenas 1 neurônio. Para essa rede, uma classificação deve ser linearmente separável, onde irá obter a reta que devide as classes. Normalmente Ɵ = -1.
  + A cada iteração vai tentando achar a reta que separa as classes corretamente
  + Processo de treinamento:
    - O Objetivo é o ajuste dos pesos. Na primeira iteração os pesos são distribuídos aleatoriamente. Normalmente entre 0 e 1



Se η é muito pequeno, demora a convergir, se for muito alto pode cair em lugares muito distantes. Ela depende do problema, são necessários experimentos.

* + Processo de operação:



* Biblioteca para realizar operações com redes neurais: PyBrain
  + Primeiro é configurado as dimensões dos vetores de entrada e também do vetor objetivo: SupervisedDataSet(2,1)
  + Adiciona os itens ao treinamento: a partir da função addSample
  + Para criar a rede: buildNetwork(dataset.indim, 4 , dataset.outdim, bias = True) , dataset.indim é a dimensao do vetor de entrada , 4 é o numero de neuronios, dataset.outdim é a dimensao de saida e bias = True é pra usar o bias
  + Para criar o treino: BackpropTrainer(network, dataset, learningrate = 0.01, momentum=0.99) , onde o momentum é para acelerar o aprendizado
  + Para testar: trainer.testOnData(test\_data)

**Biblioteca Numpy:**

* NumPy = biblioteca para computação cientifica , é baseado em C (boa performance), gratuito
* Criar matriz usando o numpy: numpy.array ([]) qualquer array n-dimensional . É melhor utilizar os arrays dessa lib, porque tem performance muito melhor.
* Para acessar o array correspondente à linha: matriz[1,:] (2ª linha)
* Para acessar o array correspondente à coluna: matriz[:,0] (1ª coluna)
* Para fazer a transposta da matriz: matriz.transpose()
* Somar todos os elementos dentro da matriz: matriz.sum()
* Retornar o índice do maior e o menor elemento da matriz: matriz.argmax() e matriz.argmin()
* Retornar a dimensão da matriz: matriz.ndim
* Copiar array por valor: a = numpy.array() -> a.copy()
* Transformar uma lista em uma matriz = numpy.reshape(data,(5,4)) - transforma em uma matriz 5x4
* Retornar o item de uma posicao = matriz.item(index)
* Biblioteca para plotar graficos = pip3 install **matplotlib**
* Para plotar alguma informação basta utilizar o método plt.plot(info) e em seguida plt.show()
* Parametros do plot = plt.plot(info , c = 'Red', ls='--', marker='s')
* Função para repetir itens dentro do array = np.tile(elementos, numero de vezes)
* Criar matriz de 1s = np.zeros((linhas,colunas)) // matriz de 0s = np.ones((linhas,colunas))
* Criar matriz identidade = np.eye(ordem)
* Filtrar elementos dentro de um array = a[condição] (exemplo: a=[1,2,3,4,5] -> a[a>3] = [4,5]
* Indexação booleana=Para retornar um array de true e false segundo uma condição: idx = (condicao no array)
* Para ler txt = np.loadtxt(nomedoarquivo, (opcional) pularlinha, unpack=True) - o unpack eh pra carregar cada coluna em uma variavel.
* Tambem pode usar a funcao np.genfromtxt(nomedoarquivo, skip\_header=1, filling\_values=-1) - o filling\_values preenche em caso de MISSING
* Para misturar um array = np.random.shuffle(array)
* Exemplo de criar uma matriz com numeros complexos: np.array([1,10+2j,20+5j], **dtype =complex**) - parte imaginaria representada pelo j
* Funcao np.linspace = gera numeros uniformemente espaçados num intervalo especifico.
  + Np.linspace(inicio,fim,amostras)
* Função que retorna os valores unicos num array = np.unique(array)
* Para ler de um CSV basta usar a mesma função genfromtxt porém com o argumento delimiter=”;” para especificar qual o delimitador
* KNN no Numpy:
  + Função train\_test\_split da lib sklearn , irá facilitar a divisão dos grupos de treinamento e de teste. Por exemplo : train\_test\_split(x,y, test\_size=0.3, random\_state=42)
  + Função KNeighborsClassifier da sklearn é o KNN. Para configurar:
    - knn = KNeighborsClassifier(n\_neighbors=17, p=2) , onde p é o tipo do calculo da distancia, no caso p=2 corresponde a distancia euclidiana.
    - Utiliza a função knn.fit(x\_treino, y\_treino) para treinar o algoritmo
    - Por fim, para obter um conjunto de saída: result = knn.predict(x\_teste)
    - Para ver a porcentagem de acertos: knn.score(x\_teste, y\_teste)

**Biblioteca PyDataset:**

* Essa biblioteca possui datasets pre carregados para uso (dados de acesso publico)
* Para ver quais datasets essa lib possui: pset.data()
* Para retornar um dataset especifico titanic = pset.data('titanic')
* Para retornar os primeiros n elementos = titanic.head(n)
* Para retornar os ultimos n elementos = titanic.tail(n)

**Biblioteca db.py :**

* Biblioteca para interação com banco de dados relacional (postgree, mysql, sql server, oracle)
* É necessario ter instalado a class do driver de conexao com o banco
* Retornar todos os elementos da tabela database.tables.nometabela.all()
* Para realizar uma query: database.query('select.... ')

**Biblioteca Pandas:**

* Biblioteca para análise de dados, facilidade de trabalhar com dados tabulados (Linguagem R)
* 2 tipos de dados: Serie (1 dimensao) e Dataframe (2 dimensoes)
* Criar série: pd.Series([1,2,3])
* Comando s.describe() = faz uma descricao da série (estatisticas basicas)
* s.mean() = Mediana = ordenar de forma crescente e se a quantidade for impar, pega o valor que está no meio, se for par pega os dois que estao no meio soma e divide por 2
* Para criar um dataframe (tabela) = pd.DataFrame([[],[],[]])
* df.shape retorna as dimensoes do dataframe
* Indicar qual o nome de cada coluna = pd.DataFrame([[],[],[]], columns=[‘nome1’,’nome2’..])
* Para retornar uma linha especifica = df.iloc[numerocoluna]
* Para criar indices associados ao dataframe : df.index = pd.Index([...])
* Transformar coluna em indice: df.index = df[nomedacoluna]
* Para ler e escrever em excel, é necessário instalar as packages **xlrd** e **xlwt**
* Para ler csv = pd.read\_csv()
* Exemplo de filtro de um dataset = print(copa.loc[copa['Quartos'] ==6]) seleciona todos as linhas cuja coluna Quartos seja = 6
* Dados categóricos = mais performático. Para transformar uma coluna em uma coluna categorica por exemplo: titanic['class'].**astype('category')**
* Para deletar todos os itens que não tiverem valor dentro do dataset = **df.dropna()** esse comando nao modifica o dataset existente, ele cria uma copia do mesmo. Irá deletar as linhas em que tiverem pelo menos uma coluna vazia. Para deletar somente se todas as colunas tiverem vazias= df.dropna(how=’all’) . Se quiser deletar coluna basta passar como parametro axis =1. Passando o threash como parametro = dropna(threash=N) , é para deletar em caso de no minimo N informacoes vazias
* Para substituir dentro de uma coluna todos os elementos vazios por um valor =

df[coluna].fillna(valor, inplace=True) #preenche com valor em caso de vazio

* Comando para plotar usando o matlib: plot(kind='TIPO') no caso de histograma usar ‘bar’ , para o grafico de pizza usar o ‘pie’
  + O metodo hist() da lib matlibplot também realiza o plot de histograma
* Plotar graficos de dispersao utilizar a funcão **scatter**() do matlibplot

**Biblioteca seaborn:**

* Abstração do matplotlib
* Quando vai instalar o seaborn, uma das dependencias é o scipy. No windows essa ultima não instala corretamente, sendo necessário instalar através de um wheel. <http://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy>
* Uma das dependencias é o numpy+mkl <http://www.lfd.uci.edu/~gohlke/pythonlibs/#numpy>
* Tem diversos estilos pre definidos para plotar graficos

**Biblioteca mplleaflet:**

* Converte o matlibplot em uma pagina web com mapa geografico