**Resumo – INTRODUÇÃO À CIÊNCIA DE DADOS**

|  |  |  |
| --- | --- | --- |
| **Aprendizado Supervisionado** | **Regressão** | 1. Regressão Linear Simples 2. Regressão Linear Múltipla 3. Regressão Ridge (Regularização) 4. Regressão Lasso ( Regularização) 5. Regressão Polinomial 6. Árvores de Decisão 7. Floresta Aleatória |
| **Classificação** | 1. Regressão Polinomial 2. KNN 3. Naive Bayes 4. Árvores de Decisão 5. Floresta Aleatória |
| **Aprendizado Não Supervisionado** | **Clusterização** | 1. K-Means 2. Agrupamento Hierárquico |
| **Outros Modelos** |  | 1. Principal Componet Analisis (PCA) 2. Suport Vector Machine (SVM) |

**O que o cientista de dados faz?**

Usa um conhecimento interdisciplinar (Matemática, Estatística, Programação, Aprendizado de Máquina (Machine Learning-ML) e conhecimento de negócios) para auxiliar na resolução de um problema de negócio.

Para responder aos problemas de negócio, geralmente um modelo de ML é implementado e para isso precisamos entender sobre os dados, o problema de negócio a ser resolvido, a capacidade computacional disponível, o nível de precisão a ser atingido, o tempo de resolução, precisamos escolher um ou mais algoritmos de ML disponíveis.

Para utilizar o aprendizado de Máquina são necessários três elementos:

1. Que um padrão existe
2. Não existe um único modelo matemático que represente esse padão (se existir um único que satisfaz, não há sentido em usar o aprendizado de máquina)
3. Disponibilidade de dados

**Modelos de Machine Learning**

As etapas para a construção de um modelo de Machine Learning é:

1. Problema de Negócio.
2. Carregar a base de dados e visualizar as primeiras linhas
3. Análise Exploratória e Limpeza
4. Pré-Processamento
5. Divisão do conjunto de dados em dados de treinamento e teste
6. Criação e Avaliação do Modelo de Machine Learning
7. Comparação de Modelos
8. Problema de Negócio

* Definição e entendimento do problema de negócio
* Informações sobre os atributos (nomes/ o que significa cada linha/coluna)

1. Carregar a base de dados e visualizar as primeiras linhas:

**Carregando os dados no Pandas:**

*import Pandas as pd*

*dados = pd.read\_csv(‘caminho/nomeArquivo.csv’)*

*dados.head( )-* visualizo as 5 primeiras linhas do data Data Frame.

Outras formas de carregar o arquivo:

1. *dados = pd.read\_csv(‘caminho/nomeArquivo.csv’, header = (0))*
2. *dados = pd.read\_csv(‘caminho/nomeArquivo.csv’, names = [‘nomecoluna1’, ‘nomecoluna2’,...,’nomecolunan’])*
3. *arquivo\_dados = ‘caminho/nomeArquivo.csv’*
4. *colunas = (‘nomecoluna1’, ‘nomecoluna2’,...,’nomecolunan’)*
5. *dados = pd.read\_csv(arquivo\_dados, names = colunas)*
6. *dados = pd.read\_csv('dados.csv', delimiter=';', encoding='UTF-8', usecols[‘nomecoluna1’, ‘nomecoluna2’,...,’nomecolunan’])*

Outras formas de visualizar linhas/ colunas do arquivo:

*dados.iloc[ : n, : ] –* visualizar as n primeiras linhas do Data Frame

*dados.tail( ) –* visualizo as 5 últimas linhas do data Data Frame.

*dados.iloc[-n: , : ] –* visualizo as n últimas linhas do data Data Frame.

Outra possibilidade de carregar os dados: usando Numpy:

*import numpy as np*

*arquivo\_dados = ‘caminho/nomeArquivo.csv’*

*arquivo\_dados = open(arquivo, 'rb')*

*dados = np.loadtxt(arquivo\_dados, delimiter = ",")*

1. Análise Exploratória e Limpeza:

Análise inicial dos dados e entender se os dados disponíveis são adequados e/ou suficientes para resolvermos o problema.

**Quais e quantas são as colunas/linhas observadas? (Dimensões do Data Frame)**

*dados.shape*

*dados.shape[0]*

*dados.shape[1]*

**O que significa os rótulos dos dados? Preciso alterar o índice ou o cabeçalho?**

*dados = pd.read\_csv('dados.csv', names=['nomecolx, 'nomecoly'], header=0)*

*dados.rename(columns={' nomecolx ': ' novo\_nomecolx '})*

*dados.rename(columns={' nomecolx ': ' novo\_nomecolx }, inplace = True)*

**Tem variáveis sem importância (tipo Identificação, por exemplo)?**

Analiso os dados e posso remover.

*dados = dados.drop(“nome\_coluna”, axis = 1)* – remove uma coluna específica

**Qual é o delimitador das colunas?**

- Abro o arquivo csv. Geralmente é a vírgula para arquivos csv.

- Posso usar o delimitador ao abrir o arquivo.

**Quais são os tipos de dados de cada atributo?**

Qualitativos, quantitativos, numérico, string, ... O Python importou corretamente? Os dados são de diferentes tipos?

*dados.dtypes*

**Tem valores faltantes nas linhas/colunas? (vazio, ?, Nan,...)**

*dados.isnull( )*

*dados.isna( )*

*dados.isnull( ).values.any( )*

*dados.isna().sum(axis=1).sum(axis=0)* – quantos dados faltantes tem em todo o DataFrame

*dados.dropna( )*

*dados.dropna(how=’all’ )* – remove linhas (padrão) onde todos os dados estejam faltando

*dados.dropna(how=’any’ , axis=0 ou 1)* – remove linhas ou colunas com algum dado faltante

*dados.dropna(tresh = n)* – remove linhas com n ou mais valores faltantes

*dados.fillna(valor)* – substitui todos os dados faltantes pelo valor (ou ‘string’) passado como parâmetro

*dados.fillna({índice ou rótulo da coluna:valor associado}, ...)* – substitui os valores faltantes nas colunas especificadas na chave do dicionário pelo valor especificado no valor do dicionário.

*dados[“nome\_coluna”] = dados[“nome\_coluna”].fillna(mode.iloc[0])*

Quantos valores nulos (Nan) tenho por coluna eu varro todas as linhas (axis = 0) ou por linha eu varro todas as colunas (axis = 1):

*dados.isna( ).sum(axis=1)*

*dados.isna( ).sum(axis=0)*

A linha/coluna com o maior número de valores nulos:

*dados.isna( ).sum(axis=1).idxmax( )*

*dados.isna( ).sum(axis=0).idxmax( )*

O valor máximo da linha/coluna com o maior número de valores nulos:

*dados.isna( ).sum(axis=1).max( )*

*dados.isna( ).sum(axis=0).max( )*

E se eu quiser para uma dada informação (uma coluna específica de um Data Frame, qual tem o menor número de informação? (Tem mais valores Nan nas linhas)

*dados.iloc[dados.isna( ).sum(axis=1).idmax( ), : ][“nome\_coluna”]*

*dados.isna( ).sum(axis=1).max( )*

**Tem dados duplicados?**

*dados.duplicated( )*

*dados.drop\_duplicates( )* – deleta os dados duplicados

*dados.duplicated( ) .sum( )* – total de linhas com dados duplicados

*dados[“nome\_coluna”].duplicated( ).sum( ) –* total de linhas de duplicadas de uma coluna específica

**Os dados estão balanceados?**

*dados.groupby('coluna\_classe').size()*

*dados['coluna\_classe'].value\_counts().data.shape[0]* – freqüência das classes

Se tiver: Na divisão dos dados (treino e teste) usar stratified Cross Validation ou técnicas de balanceamento, reamostragem ou eliminação.

**Estatística para compreensão do comportamento dos dados:**

Resumo estatístico dos dados:

*dados.describe( )*

*dados[“nome\_coluna”].describe( )*

Análise de tendência Central:

- média

- mediana

- moda

Dispersão:

- desvio padrão

- variância

- valor máximo e mínimo

- amplitudes

-IQR

Análise Gráfica:

- histograma – para visualizar a distribuição dos dados (univariado)

*dados.hist()*

*plt.show()*

- gráfico de densidade univariado

*dados.plot(kind = 'density', subplots = True, layout = (3,3), sharex = False)*

*plt.show()*

- Box-plot: ajuda a ter uma ideia da dispersão dos dados e os possíveis outliers.

*dados.plot(kind = 'box', subplots = True, layout = (3,3), sharex = False, sharey = False)*

*plt.show()*

**Como é o relacionamento entre as variáveis? Elas são correlacionadas? Os dados são redundantes?**

*dados.corr(method = 'pearson')*

ou:

*dados.corr(method = 'spearman')*

**As variáveis (atributos) disponíveis são representativas?**

Matriz de Correlação com nomes das variáveis

*correlacoes = dados.corr()*

Gráfico (simples)

*fig = plt.figure()*

*ax = fig.add\_subplot(111)*

*cax = ax.matshow(correlacoes, vmin = -1, vmax = 1)*

*fig.colorbar(cax)*

*plt.show()*

Gráfico mais sofisticado:

import numpy as np

fig = plt.figure()

ax = fig.add\_subplot(111)

cax = ax.matshow(*correlacoes*, vmin = -1, vmax = 1)

fig.colorbar(cax)

ticks = np.arange(0, 9, 1)

ax.set\_xticks(ticks)

ax.set\_yticks(ticks)

ax.set\_xticklabels(colunas)

ax.set\_yticklabels(colunas)

plt.show()

Scatterplot para todas as variáveis para visualizar o relacionamento de todas as variaveis:

*from pandas.plotting import scatter\_matrix*

*scatter\_matrix(dados)*

*plt.show()*

USANDO SEABORN PARA FAZER VISUALIZAÇÕES GRÁFICAS:

*dados.describe*

*import seaborn as sns*

*sns.pairplot(dados)*

*sns.boxplot(data = dados, orient = "v")*

*sns.clustermap(dados)*

*from scipy import stats*

*sns.distplot(dados.pedi, fit = stats.norm);*

1. Pré-processamento

Analisar os pré-requisitos de cada algoritmo e realizar as transformações nos dados para melhorar a performance dos modelos.

O algorimo aceita dados numéricos/ categóricos?

**Variável Dummy** – transformo uma string em uma variável numérica. Por exemplo: Sim ou Não convertidos em 0 e 1.

**One-Hot-Enconding-** técnica para representar uma variável categórica em numérica. Diversos valores possíveis para uma variável. Exemplo: cores: red-yellow-green

|  |  |  |  |
| --- | --- | --- | --- |
| Red | 1 | 0 | 0 |
| Yellow | 0 | 1 | 0 |
| Green | 0 | 0 | 1 |

O algorimo é sensível à valores extremos (outliers)?

O algoritmo é sensível à escala dos dados?

**Técnica 1: Normalização:**

1. Import dos módulos

*from pandas import read\_csv*

*from sklearn.preprocessing import MinMaxScaler*

1. Carregando os dados

*dados = pd.read\_csv(‘caminho/nome.csv’)*

*array = dados.values*

**C.** Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Gerando a nova escala (normalizando os dados)

*scaler = MinMaxScaler( )*

*rescaledX = scaler.fit\_transform(X)*

1. Dados transformados

*print("Dados Originais: ", dados.values)*

*print("\n Dados Normalizados: ", rescaledX( ))*

**Técnica 2: Padronização:** valores entre 0 e 1, média=0 e desvio padrão =1

1. Import dos módulos

*from pandas import read\_csv*

*from sklearn.preprocessing import StandardScaler*

**B.** Carregando os dados

*colunas = ['co0', 'co1', ' co2', ' co3',...]*

*dados = pd.read\_csv(‘caminho/nome.csv’, names = colunas)*

*array = dados.values*

**C.** Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

**D.** Gerando a nova escala (dados padronizados)

*scaler = StandardScaler().fit(X)*

*standardX = scaler.transform(X)*

**E.** Dados transformados

*print("Dados Originais: ", dados.values)*

*print("\n Dados Normalizados: ", standardX( ))*

Quais são as variáveis mais importantes? Tem muitas variáveis (Maldição da Dimensionalidade)?

**Feature Selection** – seleciona as melhores variáveis e descarta as piores, ajuda a reduzir o overfitting, ), aumenta a precisão do modelo e reduz o tempo de treinamento.

Exemplo: Técnica 1: Regressão Logística, Técnica 2: Métodos Esembles e Técnica 3:função SelectKBest() do Scikit Learn.

.

**Feature Extraction** - agrupa as variáveis em componentes. Cada componente é um grupo de variáveis/atributos com variância similar. O primeiro componente principal tem as variáveis com a maior variância (atributos mais relevantes) e os próximos componentes vão armazenando os atributos com menos relevância (menor variância). Usaremos o PCA para reduzir a dimensionalidade. O PCA precisa dos dados normalizados. Quanto maior a variância resultante, maior a quantidade de informação contida no componente.

**Feature Selection** - **Técnica 1 – Regressão Logística:**

1. Import dos módulos

*from pandas import read\_csv*

*from sklearn.feature\_selection import RFE*

*from sklearn.linear\_model import LogisticRegression*

**B.** Carregando os dados

*colunas = ['co0', 'co1', ' co2', ' co3',...]*

*dados = pd.read\_csv(‘caminho/nome.csv’, names = colunas)*

*array = dados.values*

**C.** Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Criação do modelo

*modelo = LogisticRegression()*

1. Eliminação Recursiva (RFE)

*rfe = RFE(modelo, n) – n é o número de variáveis/ features que gostaria no modelo*

*fit = rfe.fit(X, Y)*

1. Resultados

*print("Variáveis Preditoras:", dados.columns[colunas de X])*

*print("Variáveis Selecionadas: %s" % fit.support\_)*

*print("Ranking dos Atributos: %s" % fit.ranking\_)*

*print("Número de Melhores Atributos: %d" % fit.n\_features\_)*

**Feature Selection** - **Técnica 2 - Métodos Esemble para Seleção de Atributos:** Quanto maior o score, maior a importância do atributo.

1. Import dos Módulos

*from pandas import read\_csv*

*from sklearn.ensemble import ExtraTreesClassifier*

**B.** Carregando os dados

*colunas = ['co0', 'co1', ' co2', ' co3',...]*

*dados = pd.read\_csv(‘caminho/nome.csv’, names = colunas)*

*array = dados.values*

**C.** Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

**D.** Criação do Modelo - Feature Selection

*modelo = ExtraTreesClassifier()*

*modelo.fit(X, Y)*

**E**. Resultados

*print(dados.columns[do vetor X])*

*print(modelo.feature\_importances\_)*

**Técnica 3 –** Testes estatísticos podem ser usados para selecionar os atributos que possuem forte relacionamento com a variável que estamos tentando prever. O **Scikit-learn** fornece a função **SelectKBest()** que pode ser usada com diversos testes estatísticos, para selecionar os atributos.

**A.** Import dos módulos

*from pandas import read\_csv*

*from sklearn.feature\_selection import SelectKBest*

*from sklearn.feature\_selection import chi2*

**B.** Carregando os dados

*colunas = ['co0', 'co1', ' co2', ' co3',...]*

*dados = pd.read\_csv(‘caminho/nome.csv’, names = colunas)*

*array = dados.values*

**C.** Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

**D.** Função para seleção de variáveis

*best\_var = SelectKBest(score\_func = chi2, k = número de variáveis)*

*fit = best\_var.fit(X, Y)*

*features = fit.transform(X)*

**E.** Variáveis selecionadas

*print('Número original de features:', X.shape[1])*

*print('Número reduzido de features:', features.shape[1])*

*print('Features (Variáveis Selecionadas): ', features)*

Exemplo:

import numpy as np

from sklearn.feature\_selection import SelectKBest, f\_classif

predictors = ["Pclass", "Sex", "Age", "SibSp", "Parch", "Fare", "Embarked", "FamilySize", "Title", "FamilyId"]

# Perform feature selection

selector = SelectKBest(f\_classif, k=5)

selector.fit(titanic[predictors], titanic["Survived"])

# Get the raw p-values for each feature, and transform from p-values into scores

scores = -np.log10(selector.pvalues\_)

# Plot the scores. See how "Pclass", "Sex", "Title", and "Fare" are the best?

plt.bar(range(len(predictors)), scores)

plt.xticks(range(len(predictors)), predictors, rotation='vertical')

plt.show()

Por exemplo, o método do valor F da ANOVA é apropriado para entradas numéricas e dados categóricos, como vemos no conjunto de dados Pima.

Isso pode ser usado através da função f\_classif (). Selecionaremos os 4 melhores recursos usando esse método no exemplo abaixo.



**Feature Extraction – Principal Component Analysis (PCA)**

1. Import dos módulos

*from pandas import read\_csv*

*from sklearn.preprocessing import MinMaxScaler*

*from sklearn.decomposition import PCA*

**B.** Carregando os dados

*colunas = ['co0', 'co1', ' co2', ' co3',...]*

*dados = pd.read\_csv(‘caminho/nome.csv’, names = colunas)*

*array = dados.values*

**C.** Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

**D.** Normalizando os dados

*scaler = MinMaxScaler(feature\_range = (0, 1))*

*rescaledX = scaler.fit\_transform(X)*

**E.** Seleção de atributos

*pca = PCA(n\_components = escolho quantas componentes quero)*

*fit = pca.fit(rescaledX)*

**F.** Sumarizando os componentes

*print("Variância: %s" % fit.explained\_variance\_ratio\_)*

*print(fit.components\_)*

1. Divisão do conjunto de dados em dados de treinamento e teste

**Técnica 1: Hold-Out**

Divido o conjunto de dados em treino e teste em p e 1-p (exemplo: 80 e 20%, 70 e 30%)

*X\_treino, X\_teste, Y\_treino, Y\_teste = train\_test\_split(X, Y, test\_size = p, random\_state = seed)*

Quando for ajustar o modelo:

*modelo.fit(X\_treino, Y\_treino)*

Quando for avaliar a performace do modelo (score):

*result = modelo.score(X\_teste, Y\_teste)*

**Técnica 2: Cross-Validation**

Defino o número de folds:

*kfolds = 10*

Quando for ajustar o modelo:

*modelo = algortimoescolhido( )*

Quando for avaliar o modelo:

*resultado = cross\_val\_score(modelo, X, Y, cv = kfold)*

*print("Acurácia Final: %.3f%%" % (resultado.mean() \* 100.0))*

Quando for avaliar o modelo usando a curva ROC:

*resultado = cross\_val\_score(model, X, Y, cv = kfold, scoring = 'roc\_auc')*

1. Criação e Avaliação do Modelo de Machine Learning

**Regressão Linear**

Pré-requisitos: Distribuição Normal, variáveis do modelo são importantes e não correlacionadas

1. Import do módulo

*Import pandas as pd*

*from sklearn.linear\_model import LinearRegression*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.metrics import r2\_score*

*from sklearn.metrics import mean\_squared\_error*

*from sklearn.metrics import mean\_absolute\_error*

*from sklearn.metrics import mean\_squared\_error*

1. Carregando os dados

*dados = pd.read\_csv(‘caminho/nome.csv’)*

*array = dados.values*

1. Exploração dos dados + Pré-processamento
2. Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Divisão dos dados em treino e teste

*X\_treino, X\_teste, Y\_treino, Y\_teste = train\_test\_split(X, Y, test\_size = p)*

*X\_treino = dataset\_treino\_variaveis\_preditoras*

*Y\_treino = dataset\_treino\_variavel\_prevista*

*X\_teste = dataset\_teste\_variaveis\_preditoras*

1. Criação do modelo: algoritmo de Regressão Linear, ajuste, avaliação dos coeficientes (score) e predições

*modelo = linear\_model.LinearRegression()*

*modelo.fit(x\_treino, y\_treino)*

*modelo.score(x\_treino, y\_treino)*

*print('Coefficient:', modelo.coef\_)*

*print('Intercept:', modelo.intercept\_)*

*valores\_previstos = modelo.predict(x\_teste)*

1. Avaliação do modelo – usando várias métricas

*r2 = r2\_score(Y\_test, Y\_pred)*

*print("O R2 do modelo é:", r2)*

*mse = mean\_squared\_error(Y\_test, Y\_pred)*

*print("O MSE do modelo é:", mse)*

*mae = mean\_absolute\_error(Y\_test, Y\_pred)*

*print("O MAE do modelo é:", mae)*

**Regressão Logística**

1. Imports do módulo

*from sklearn.linear\_model import LogisticRegression*

1. Carregando os dados

*dados = pd.read\_csv(‘caminho/nome.csv’)*

*array = dados.values*

1. Exploração dos dados + Pré-processamento
2. Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Divisão dos dados em treino e teste

*X\_treino, X\_teste, Y\_treino, Y\_teste = train\_test\_split(X, Y, test\_size = p)*

*X\_treino = dataset\_treino\_variaveis\_preditoras*

*Y\_treino = dataset\_treino\_variavel\_prevista*

*X\_teste = dataset\_teste\_variaveis\_preditoras*

1. Criação do modelo: algoritmo de Regressão Linear, ajuste, avaliação dos coeficientes (score) e predições

*modelo = LogisticRegression()*

*modelo.fit(x\_treino, y\_treino)*

*modelo.score(x\_treino, y\_treino)*

*print('Coefficient:', modelo.coef\_)*

*print('Intercept:', modelo.intercept\_)*

*valores\_previstos = modelo.predict(x\_teste)*

1. Avaliação do modelo

**Regressão Ridge**

A Regressão Rigde é uma Regressão Linear onde a cost function possui um parâmetro de penalização (2ª norma – Euclidiana). Esse parâmetro “controla” os valores dos Betas para evitar o overfitting.

1. Import do módulo

*import pandas as pd*

*from sklearn.linear\_model import Ridge*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.metrics import mean\_squared\_error*

1. Carregando os dados

*dados = pd.read\_csv(‘caminho/nome.csv’)*

*array = dados.values*

1. Exploração dos dados + Pré-processamento
2. Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Divisão dos dados em treino e teste

*X\_treino, X\_teste, Y\_treino, Y\_teste = train\_test\_split(X, Y, test\_size = p)*

1. Criação do modelo: algoritmo, ajuste, avaliação dos coeficientes (score) e predições

*modelo = Ridge()*

*modelo.fit(X\_train, Y\_train)*

*Y\_pred = modelo.predict(X\_test)*

1. Avaliação do modelo

*mse = mean\_squared\_error(Y\_test, Y\_pred)*

*print("O MSE do modelo é:", mse)*

**Regressão Lasso**

A Regressão Lasso é uma Regressão Linear onde a cost function possui um parâmetro de penalização (1ª norma – Manhattan). Esse parâmetro zera os Betas para atributos irrelevantes, ou seja, ajuda a eliminar os atributos que não contribuem para a predição dos dados.

1. Import do módulo

*import pandas as pd*

*from sklearn.linear\_model import Lasso*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.metrics import mean\_squared\_error*

1. Carregando os dados

*dados = pd.read\_csv(‘caminho/nome.csv’)*

*array = dados.values*

1. Exploração dos dados + Pré-processamento
2. Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Divisão dos dados em treino e teste

*X\_treino, X\_teste, Y\_treino, Y\_teste = train\_test\_split(X, Y, test\_size = p)*

1. Criação do modelo: algoritmo, ajuste, avaliação dos coeficientes (score) e predições

*modelo = Lasso()*

*modelo.fit(X\_train, Y\_train)*

*Y\_pred = modelo.predict(X\_test)*

1. Avaliação do modelo

*mse = mean\_squared\_error(Y\_test, Y\_pred)*

*print("O MSE do modelo é:", mse)*

**Árvores de decisão**

1. Import do módulo

*Import pandas as pd*

*from sklearn import tree*

1. Carregando os dados

*dados = pd.read\_csv(‘caminho/nome.csv’)*

*array = dados.values*

1. Exploração dos dados + Pré-processamento
2. Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Divisão dos dados em treino e teste

*X\_treino, X\_teste, Y\_treino, Y\_teste = train\_test\_split(X, Y, test\_size = p)*

1. Criação do modelo: algoritmo, ajuste, avaliação dos coeficientes (score) e predições

modelo = tree.DecisionTreeRegressor() *–* Para Regressão

*modelo = tree.DecisionTreeClassifier­()* – Para Classificação

*modelo.fit(x\_treino, y\_treino)*

*modelo.score(x\_treino, y\_treino)*

*valores\_previstos = modelo.predict(x\_teste)*

1. Avaliação do modelo

*acc = round(accuracy\_score(y\_test, y\_pred), 2)*

*f1 = round(f1\_score(y\_test, y\_pred, average="macro"), 2)*

*precision = round(precision\_score(y\_test, y\_pred, average="macro"), 2)*

*recall = round(recall\_score(y\_test, y\_pred, average="macro"), 2)*

*print(f"F1 score: {f1}")*

*print(f"Precision: {precision}")*

*print(f"Accuracy: {acc}")*

*print(f"Recall: {recall}")*

**Classificação**

**KNN**

1. Import do módulo

*Import pandas as pd*

*from sklearn.neighbors import KNeighborsRegressor*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.metrics import mean\_squared\_error*

1. Carregando os dados

*dados = pd.read\_csv(‘caminho/nome.csv’)*

*array = dados.values*

1. Exploração dos dados + Pré-processamento
2. Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Divisão dos dados em treino e teste

*X\_treino, X\_teste, Y\_treino, Y\_teste = train\_test\_split(X, Y, test\_size = p)*

1. Criação do modelo: algoritmo, ajuste, avaliação dos coeficientes (score) e predições

*modelo = KNeighborsRegressor()*

*modelo.fit(X\_train, Y\_train)*

*Y\_pred = modelo.predict(X\_test)*

1. Avaliação do modelo

*mse = mean\_squared\_error(Y\_test, Y\_pred)*

*print("O MSE do modelo é:", mse)*

**Floresta Aleatória (Randon Forest)**

1. Import do módulo

Import pandas as pd

from sklearn.model\_selection import KFold

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import cross\_val\_score

1. Carregando os dados

*dados = pd.read\_csv(‘caminho/nome.csv’)*

*array = dados.values*

1. Exploração dos dados + Pré-processamento
2. Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Definindo o número de trees

*num\_trees = 100*

*max\_features = 3*

1. Criação do modelo

*modelo = RandomForestClassifier(n\_estimators = num\_trees, max\_features = max\_features)*

*model.fit(x\_train, y\_train)*

*model.score(x\_train, y\_train)*

*y\_pred = model.predict(x\_test)*

1. Avaliação do modelo

resultado = cross\_val\_score(modelo, X, Y, cv = kfold)

print("Acurácia: %.3f" % (resultado.mean() \* 100))

Outras Métricas de avaliação

*acc = round(accuracy\_score(y\_test, y\_pred), 1)*

*f1 = round(f1\_score(y\_test, y\_pred, average="macro"), 1)*

*precision = round(precision\_score(y\_test, y\_pred, average="macro"), 1)*

*recall = round(recall\_score(y\_test, y\_pred, average="macro"), 1)*

**Regressão**

1. Import do módulo

*Import pandas as pd*

1. Carregando os dados

*dados = pd.read\_csv(‘caminho/nome.csv’)*

*array = dados.values*

1. Exploração dos dados + Pré-processamento
2. Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Divisão dos dados em treino e teste

*X\_treino, X\_teste, Y\_treino, Y\_teste = train\_test\_split(X, Y, test\_size = p)*

1. Criação do modelo: algoritmo, ajuste, avaliação dos coeficientes (score) e predições
2. Avaliação do modelo

**Regressão**

1. Import do módulo

*Import pandas as pd*

1. Carregando os dados

*dados = pd.read\_csv(‘caminho/nome.csv’)*

*array = dados.values*

1. Exploração dos dados + Pré-processamento
2. Separando as entradas X e saídas Y

*X = array[:,0:n]*

*Y = array[:,n]*

1. Divisão dos dados em treino e teste

*X\_treino, X\_teste, Y\_treino, Y\_teste = train\_test\_split(X, Y, test\_size = p)*

1. Criação do modelo: algoritmo, ajuste, avaliação dos coeficientes (score) e predições
2. Avaliação do modelo
3. Comparação de Modelos