LIT

SaintPaul

Modelos Supervisionados de Machine Learning

Prof. Dr. Daniel Bergmann

Modelos Supervisionados de Machine Learning

1 Introdução

Olá, sou o Professor Daniel Bergmann do curso de Machine Learning e Data Science com Python do LIT. Neste tópico, aprenderemos as principais modelos supervisionados de Machine Learning e Data Science. É recomendável que os participantes e aluno(a)s já tenham assistido a unidade sobre Fundamentos de Python e a outra sobre Python para Data Science.

Espero que seja uma ótima experiência de aprendizado e que possamos vencer todos os desafios juntos. Neste módulo, vamos dar nossos primeiros passos para a modelagem dos modelos supervisionados.

Quando criança aprendemos as cores através de uma pessoa (pai, professor, mãe etc) nos dizendo que determinado objeto tem uma certa cor. Fomos supervisionados por uma pessoa para nos dizer que verde tem aquela aparência, por exemplo. Sucessivamente vamos associando e decorando as cores como nos foram evidenciadas neste experimento (banco de dados).

Assim, são os modelos supervisionados em Machine Learning: necessita-se de uma definição, a priori, de quem é a variável dependente (y). Y pode ser o parâmetro sobre vendas, status do cliente, inadimplência etc. Mas, como uma máquina aprende?

Vamos entender como isso funciona com um exemplo mais simples. Imagine que você é um médico e está interessado em classificar seus pacientes como “saudáveis” ou “doentes”. Para cada paciente, você possui um conjunto de dados (informações) como: altura, peso, pressão arterial, nível de colesterol, percentual de gordura, entre outros.

Você já possui um histórico muito grande de pacientes com seus respectivos dados e classificações como saudáveis ou doentes, e quer utilizar um algoritmo de machine learning para aprender com esses dados de maneira que, quando você receber os dados de um novo paciente, o algoritmo possa dizer se ele está saudável ou doente.

Então como isso vai funcionar? Você irá alimentar o algoritmo de machine learning com os dados históricos já classificados (entenda que estamos interessados aqui na classificação entre saudável ou doente). A partir desses dados, o algoritmo de machine learning irá aprender quais condições são necessárias para o paciente ser classificado como saudável e quais condições classificam o paciente como doente.

Essa etapa o algoritmo irá fazer sozinho, você não precisa fazer nada além de alimentar os dados no programa e colocar a função para rodar (essas funções de algoritmos de machine learning já estão prontas e você pode solicitá-las com comandos simples de programação no Python).

Repare que agora, como saída desse processo, temos um modelo já treinado. Em outras palavras, temos um modelo que já aprendeu a relação dos dados (quais características classificam um paciente como doente ou saudável).

Propriedade Intelectual Saint Paul Educacional LTDA

Então já podemos utilizar esse modelo para realizar previsões sobre novos dados. Podemos alimentar o modelo treinado com as informações de novos pacientes e o modelo irá prever se cada um desses pacientes é saudável ou doente. Mas como saberemos se essas previsões que o algoritmo irá fazer estão corretas ou não? Qual seria o grau de confiabilidade deste modelo?

Imagine que, em vez de utilizar todos os nossos dados históricos para treinar o modelo, utilizamos apenas 70% desses dados e separamos os outros 30% para servirem como teste depois (esses 20% são dados o modelo não irá receber para treinar). A lógica é a seguinte: se o modelo conseguir fazer previsões corretas sobre os dados de teste, significa que o modelo está funcionando bem.

Esses dados de teste também são dados históricos, ou seja, já estão classificados, então basta comparar as previsões do modelo com as respectivas classificações para ver se ele acertou ou errou nas suas previsões.

Sugerimos a leitura do material da apostila com os vídeos de aula que apresentam os mesmos problemas demonstrados aqui neste material.

2 Modelo de Regressão

Em estatística ou econometria, o modelo de regressão linear é uma equação para se estimar os valores de uma variável dependente (y) com base nos atributos (x) por meio do método dos mínimos quadrados ordinários. Vamos tratar de um exemplo para verificar como podemos rodar um modelo de regressão em Python.

Um agente imobiliário deseja prever, com acurácia, os preços das casas das regiões nos EUA e, dessa forma, contrata um cientista de dados (você). Seria ótimo se você pudesse de alguma forma criar um modelo a fim de permitir que sejam inseridas algumas características de uma casa e retornar uma estimativa de quanto a casa possa ser vendida (preço justo). Sua decisão é implementar um modelo de Regressão linear a fim de resolver o problema.

O banco de dados possui as seguintes variáveis:

- 'Avg. Area Income': Média da renda dos residentes onde a casa está localizada.
- 'Avg. Area House Age': Média de idade das casas da mesma cidade.
- 'Avg. Area Number of Rooms': Número médio de quartos para casas na mesma cidade.
- 'Avg. Area Number of Bedrooms': Número médio de quartos para casas na mesma cidade
- 'Area Population': A população da cidade onde a casa está localizada.
- 'Price': Preço de venda da casa.
- 'Address': Endereço da casa;

Primeiro devemos inserir os pacotes de Python necessários para rodar o modelo de regressão:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn import datasets, linear_model
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from scipy import stats
%matplotlib inline
```

Devemos importar o banco de dados no GitHub (repositório de pacotes e programas) de propriedade do Prof. Daniel Bergmann conforme os códigos abaixo:

```
import pandas as pd
fonte = "https://github.com/daniel-usp/LIT/blob/master/USA_Housing.csv?raw=true"
USAhousing = pd.read_csv(fonte) #Banco de Dados do modelo de regressão
```

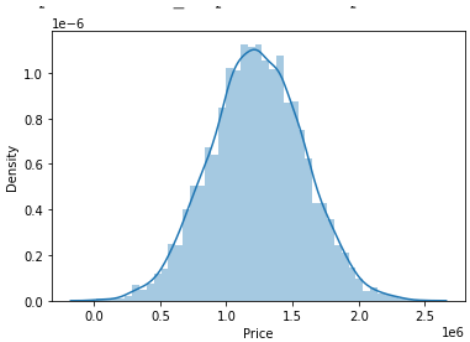
Podemos calcular as estatísticas descritivas conforme o código a seguir:

```
USAhousing.describe()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

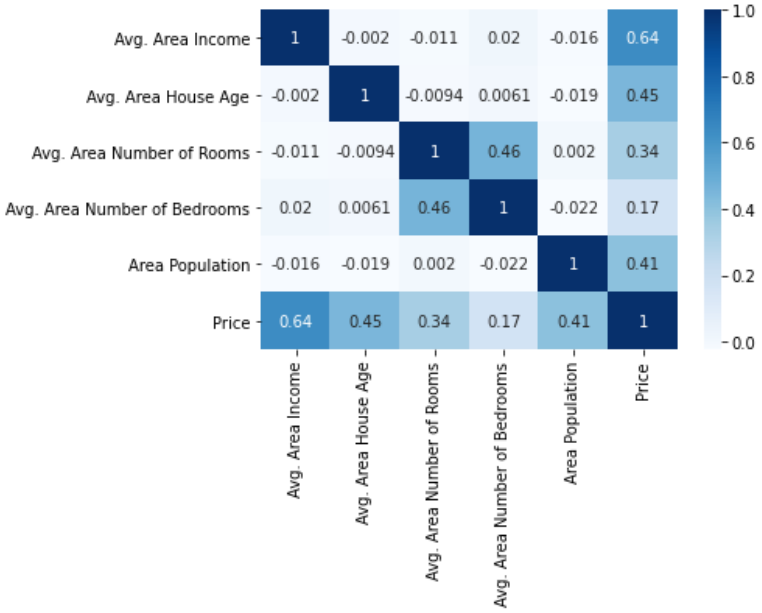
Podemos visualizar o histograma da distribuição dos Preços (‘Price’) por meio do pacote Seaborn.

```
sns.distplot(USAhousing['Price'])
```



Verificamos que os preços das casas nos EUA tendem a seguir uma distribuição normal de probabilidades. Podemos também elaborar uma matriz de correlação entre as variáveis quantitativas do banco de dados por meio do seguinte comando:

```
sns.heatmap(USAhousing.corr(), cmap = "Blues", annot = True)
```



Percebemos que a variável 'Price' possui maior correlação com o atributo 'Avg. Area Income' (média da renda dos residentes onde a casa está localizada. Provavelmente, o atributo 'Avg. Number of Bedrooms' (Número médio de quartos para casas na mesma cidade) não terá relacionamento significativo com o Preço devido a correlação baixa de 17%, como demonstrado na última linha do mapa de calor de correlações acima.

Um dos grandes problemas dos modelos de regressão é o problema de multicolinearidade, ou seja, trata-se da correlação forte (acima de 70%) entre os atributos do modelo supervisionado. Na sua presença, o cientista de dados pode excluir uma delas a fim de obter resultados mais satisfatórios em termos de previsão.

Suponhamos dois atributos (X1, X2) com alto grau de correlação dentro de um modelo com diversos outros atributos. Uma alternativa seria excluir, por exemplo, X2 e avaliar o impacto de X1 e outros atributos no coeficiente de explicação R2 (o quanto os atributos do modelo explicam as variações da variável dependente y). Posteriormente, excluímos X1 e verificamos o impacto de X2 e outros atributos no coeficiente R2. Aquela alternativa que oferecer maior coeficiente de explicação é o atributo que deve ficar no modelo final de regressão.

1.1. Separação em amostras Treino e Teste e o problema de Overfitting

Poderíamos utilizar a totalidade dos dados históricos, criando assim um modelo de machine learning pronto para receber novos dados e realizar suas previsões, porém desta forma não saberíamos o real desempenho deste modelo. O algoritmo poderia aprender perfeitamente a relação existentes nos dados apresentados e com isso criar um modelo que sofre de Overfitting (Superajustamento) e só descobriríamos esse problema após as previsões desastrosas geradas por este modelo.

Desta forma, para medir o desempenho real do modelo criado, é necessário que realizemos testes com ele, utilizando dados diferentes dos que foram apresentados em sua criação.

Com esta finalidade, após a realização do pré-processamento, iremos **separar a totalidade dos dados históricos existentes em dois grupos**, sendo o primeiro responsável pelo **aprendizado** do modelo, e o segundo por realizar os **testes**.

Dados de Treino

Conforme podemos imaginar, dados de treino são os dados que serão apresentados ao algoritmo de machine learning para criação do modelo. Estes dados costumam representar cerca de 70% ou 80% da totalidade dos dados.

Dados de Teste

Serão apresentados ao modelo após a sua criação, simulando previsões reais que o modelo realizará, permitindo assim que o desempenho real seja verificado. Estes dados costumam representar cerca de 30% ou 40% da totalidade dos dados.

É importante observar que a separação dos dados em treino e teste é uma etapa essencial, e que caso ela seja realizada de maneira improducente, poderá resultar em problemas no modelo.

Imagine que você tenha dados sobre dez mil carros, com algumas características e o valor deles. Você decide que irá utilizar cinco mil amostras para treinar seu modelo, e para isso seleciona as cinco mil primeiras linhas, deixando as últimas cinco mil para teste. Porém estas linhas estavam organizadas em ordem alfabética com base nos nomes dos carros. Percebem o problema? Alguns modelos de carros específicos estarão presentes apenas nos dados de treino, e outros apenas nos dados de teste.

Esta situação levará a um modelo ineficiente, que não aprendeu com todos os tipos de dados que poderia também não ser testado da maneira correta. Afinal, nos dados de teste haverão apenas modelos de carros que começam, digamos, com as letras X, Y, Z (final do alfabeto).

A **solução para este problema** está na **aleatoriedade**. Selecionando os dados de maneira aleatória não haverá padrão algum no momento da divisão dos dados, e cada observação terá a mesma probabilidade de ser selecionada. Assim, iremos usar o pacote sckilearn do Python para dividir nossa amostra em dados de treino e de teste de forma aleatória conforme os comandos abaixo.

Agora vamos dividir os dados em um conjunto de treinamento e um conjunto de testes. Vamos criar o modelo usando o conjunto de treinamento e depois usar o conjunto de testes para avaliar o modelo.

```
from sklearn.model_selection import train_test_split
```

O método `train_test_split` fará o trabalho da escolha aleatória dos dados de treino e de teste. Vamos então codificar para nossa amostra de carros dos EUA.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=101)
X_train
```

No caso acima, teremos uma amostra de teste com 40% das observações no total da amostra original e outra amostra treino com 60% de observações da mesma. Os rótulos **`X_train, y_train`** representam os atributos e a variável dependente da amostra treino e os rótulos **`X_test, y_test`** designam os atributos e a variável dependente da amostra teste.

Para treinar o modelo importaremos o método da Regressão da seguinte forma:

```
from sklearn.linear_model import LinearRegression
```

Criaremos a instância `lm` (nome arbitrário) para designar o método `LinearRegression()` do seguinte modo:

```
>> lm = LinearRegression()
```

Qualquer objeto do modelo de Regressão (parâmetros estimados, erros, dados de previsão) serão chamados por meio da instância `lm`. Para realizar a estimação dos modelos supervisionados sempre utilizaremos a função `fit()` dentro do método `lm`.

```
>> lm.fit(X_train,y_train)
```

Neste caso, todos os parâmetros estimados da regressão são armazenados no objeto `lm`. Podemos gerar o Relatório de Regressão a partir do seguinte conjunto de códigos (explicado na vídeo-aula sobre o modelo de Regressão):

```
X2 = sm.add_constant(X_train)
est = sm.OLS(y_train, X2) #Estinmação dos parâmetros
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.918
Model:                            OLS   Adj. R-squared:                  0.918
Method:                 Least Squares   F-statistic:                     8391.
Date:                Sun, 31 Jan 2021   Prob (F-statistic):               0.00
Time:                        22:37:27   Log-Likelihood:                -38807.
No. Observations:                3000   AIC:                         7.762e+04
Df Residuals:                    2995   BIC:                         7.765e+04
Df Model:                           4
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                        -2.64e+06   2.22e+04   -119.052      0.000   -2.68e+06    -2.6e+06
Avg. Area Income              21.5283      0.174    124.024      0.000      21.188      21.869
Avg. Area House Age          1.65e+05   1883.369     87.584      0.000    1.61e+05    1.69e+05
Avg. Area Number of Rooms    1.237e+05   1843.072     67.092      0.000     1.2e+05    1.27e+05
Area Population               15.1437      0.184     82.375      0.000      14.783      15.504
==============================================================================
Omnibus:                        5.112   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.078   Jarque-Bera (JB):                4.443
Skew:                           0.017   Prob(JB):                        0.108
Kurtosis:                       2.814   Cond. No.                     9.46e+05
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 9.46e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Podemos verificar que todos os atributos evidenciados acima possuem relacionamento significativo com a variável preço, pois os Valores-P (P>|t|) são inferiores a 5%[1]. Assim, preservaremos estas variáveis na nossa base treino de imóveis dos EUA.

O ajuste do modelo está muito bom, pois o R ao quadrado foi de 91,8%, ou seja, todos os atributos evidenciados acima explicam cerca de 91,8% as variações dos preços dos imóveis dos EUA na amostra de treino.

```
>> coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coeficientes'])
coeff_df
```

	Coeficientes
Avg. Area Income	21.528348
Avg. Area House Age	164953.468397
Avg. Area Number of Rooms	123655.616725
Area Population	15.143711

Interpretando os coeficientes:

[1] Explicação do Valor-P está no curso do LIT sobre Métodos Quantitativos Aplicados a Negócios na Unidade de Aprendizagem sobre Regressão. No vídeo sobre 'Conceitos de Regressão' o Prof. Daniel explica sobre os temas de Valor-P e R ao quadrado.

- Mantendo todas as outras variáveis constantes, um aumento de 1 unidade em **Avg. Area Income** está associado a um aumento de $ 21,52.
- Mantendo todas as outras variáveis constantes, um aumento de 1 unidade em **Avg. Area House Age** está associada a um aumento de $ 164883.28.
- Mantendo todas as outras variáveis constantes, um aumento de 1 unidade em **Avg. Area Number of Bedrooms** está associada a um aumento de $ 122368.67.
- Mantendo todas as outras variáveis constantes, um aumento de 1 unidade em **Avg. Area Number of Bedrooms** está associada a um aumento de $ 2233.8.
- Mantendo todas as outras variáveis constantes, um aumento de 1 unidade em **Area Population** está associado a um aumento de $ 15.15.

Agora, iremos avaliar a capacidade de previsão com os dados da amostra teste de acordo com os seguintes códigos:

```
>> predictions = lm.predict(X_test)
```

No código acima, faremos a previsão dos valores de y com base na amostra `X_test` a fim de comparar com os resultados reais existentes em `y_test.`

Aqui estão três métricas de avaliação comuns para problemas onde a variável y é contínua:

Erro absoluto médio (MAE) é a média do valor absoluto dos erros:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Onde y_i são os valores reais da amostra teste `y_test` e $\hat{y}_i$ representam as previsões obtidas na regressão.

Erro médio quadrático (MSE) é a média dos erros quadrados:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Raiz do erro quadrático médio (RMSE) é a raiz quadrada da média dos erros quadrados:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Comparando estas métricas:

- MAE é o mais fácil de entender, porque é o erro médio clássico da estatística.

- MSE é mais popular que o MAE, porque a MSE "puniria" erros maiores, o que tende a ser útil no mundo real.
- RMSE é ainda mais popular do que MSE, porque o RMSE é interpretável nas unidades "y".

Todas estas são **funções de perda**, porque queremos minimizá-las. Logo, quanto menor forem estes valores em comparação a outros modelos utilizados melhor será o ajuste da técnica supervisionada em questão. Devemos importar o método `metrics` do pacote `sklearn`.

```
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

Os resultados foram: MAE: 82248.33, MSE: 10459278321.47 e RMSE: 102270.61. Devemos comparar estes valores com os resultados das métricas de outro modelo supervisionado (por exemplo, Random Forest ao qual veremos em seção adiante nesta apostila). Se os resultados deste outro modelo possuírem métricas superiores ao mostrado acima, concluiremos que o modelo de regressão possui um ajuste superior aos dados de previsão.

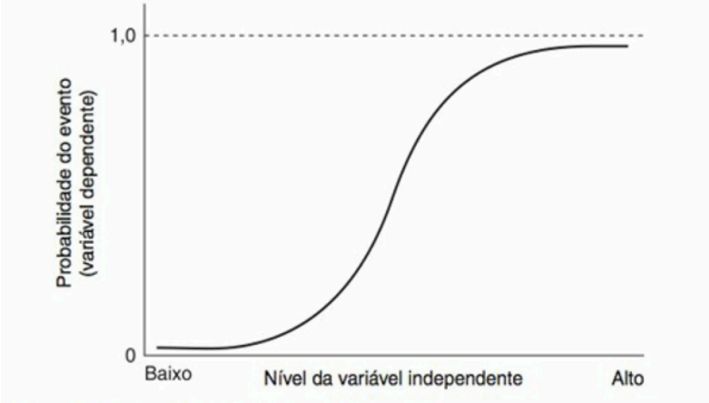
3 Modelo de Regressão Logística

A técnica de regressão logística foi desenvolvida por volta da década de 60, em resposta ao desfio de realizar predições ou explicar a ocorrência de determinados fenômenos quando a variável dependente fosse de natureza binária.

A regressão logística é uma técnica estatística utilizada para descrever o comportamento entre uma variável dependente binária e variáveis independentes. Ou seja, destina-se a investigar o efeito das variáveis pelas quais os indivíduos, objetos ou sujeitos estão expostos sobre a probabilidade de ocorrência de determinado evento de interesse. Por exemplo, suponha que uma seguradora esteja interessada em investigar qual a probabilidade de uma pessoa falecer dado que é ou não fumante. Neste sentido, o evento de interesse seria a morte (variável dependente (y) = 1), enquanto a não ocorrência poderia ser denotada por 0. Adicionalmente, podem ser introduzidas no modelo algumas variáveis de controle que podem estar, de alguma forma, relacionadas ao evento de interesse, como, por exemplo, a idade, o sexo, a prática de esportes etc.

A regressão logística foi desenvolvida para lidar especificamente com essas questões. Não obstante, sua relação única entre variáveis dependente e independentes exige uma abordagem um tanto diferente para estimar a variável estatística, avaliar adequação de ajuste e interpretar os coeficientes, quando comparada com regressão múltipla. Para tal propósito, a regressão logística se utiliza de uma função especial, conhecida como função logística, para sanar os problemas destacados acima e investigar a probabilidade de ocorrência da variável dependente de interesse.

A função logística (ou modelo logit), $f(Z) = \frac{1}{1+e^{-Z}}$, assume valores entre 0 e 1 para qualquer Z no intervalo dos números reais. Assim, a popularidade da técnica advém não apenas da possibilidade de prever a ocorrência de eventos de interesse, mas também da capacidade de apresentar probabilidade de sua ocorrência. Segue a Figura a seguir que evidencia o formato da função logística.



Como mostrado anteriormente, o modelo logit usa a forma específica da curva logística, que é em forma de S para ficar no intervalo de 0 a 1, medindo desta forma o nível de probabilidade da variável de interesse. Para estimar um modelo de regressão logística, essa curva de valores previstos é ajustada aos dados reais, exatamente como é realizado com os modelos de regressão. No entanto, como os valores reais dos dados das variáveis dependentes podem ser somente 0 ou 1, o processo é de algum modo diferente[2].

Neste modelo de ciência de dados estaremos trabalhando com um conjunto de dados referente a uma propaganda de um e-commerce. Estes dados indicam se um usuário (possível cliente) clicou ou não nesta propaganda. Vamos tentar criar um modelo que preveja se clicará ou não em um anúncio baseado nos recursos rastreados desse usuário.

Este conjunto de dados contém os seguintes recursos:

- 'Daily Time Spent on Site': tempo diário no site em minutos.
- 'Age': idade do consumidor.
- 'Area Income': Média da renda do consumidor na região.
- 'Daily Internet Usage': Média em minutos por dia que o consumidor está na internet.
- 'Linha do tópico do anúncio': Título do anúncio.
- 'City': Cidade do consumidor.
- 'Male': Se o consumidor é do gênero masculino.
- 'Country': País do consumidor.
- 'Timestamp': hora em que o consumidor clicou no anúncio ou janela fechada.

[2] Detalhes dos conceitos do modelo de Regressão Logística podem ser encontrados no curso de Análise Multivariada para tomada de decisões ministrado pelo Prof. Daniel Bergmann no LIT.

- 'Clicked on Ad": 0 ou 1 indicam se clicou ou não no anúncio.

Primeiramente, devemos importar os pacotes necessários no Python para rodar o modelo de Regressão Logística.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Caso o banco de dados seja nomeado por **ad_data** podemos calcular a estatística descritiva da seguinte forma:

```
ad_data.describe()
```

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	65.000200	36.009000	55000.000080	180.000100	0.481000	0.50000
std	15.853615	8.785562	13414.634022	43.902339	0.499889	0.50025
min	32.600000	19.000000	13996.500000	104.780000	0.000000	0.00000
25%	51.360000	29.000000	47031.802500	138.830000	0.000000	0.00000
50%	68.215000	35.000000	57012.300000	183.130000	0.000000	0.50000
75%	78.547500	42.000000	65470.635000	218.792500	1.000000	1.00000
max	91.430000	61.000000	79484.800000	269.960000	1.000000	1.00000

Agora vamos dividir os dados em um conjunto de treinamento e um conjunto de testes. Vamos criar o modelo usando o conjunto de treinamento e depois usar o conjunto de testes para avaliar o modelo.

```
from sklearn.model_selection import train_test_split
```

O método **train_test_split** fará o trabalho da escolha aleatória dos dados de treino e de teste. Vamos então codificar para nossa amostra sobre os dados dos usuários no e-commerce.

```
X = ad_data[['Daily Time Spent on Site', 'Age', 'Area Income','Daily Internet
Usage', 'Male']]
y = ad_data['Clicked on Ad']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=101)
```

No caso acima, teremos uma amostra de teste com 40% das observações no total da amostra original e outra amostra treino com 60% de observações da mesma. Os rótulos **X_train**, **y_train**

representam os atributos e a variável dependente da amostra treino e os rótulos **`X_test, y_test`** designam os atributos e a variável dependente da amostra teste.

```
from sklearn.linear_model import LogisticRegression
```

Criaremos a instância `logmodel` (nome arbitrário) para designar o método `LogisticRegression()` do seguinte modo:

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

Para realizar a estimação dos modelos supervisionados sempre utilizaremos a função `fit()` dentro do método `logmodel.`

As previsões da amostra teste serão realizadas pelo método do **`sklearn`** conhecido como **`predict()`** conforme se vê logo a seguir.

```
>> predictions = logmodel.predict(X_test)
```

1.2. Matriz de Confusão

É uma tabela que mostra as frequências de classificação para cada classe do modelo. Imagine um exemplo que um cientista de dados gostaria de prever se um email pode ser classificado como spam.

		Classe Prevista	
		Não-SPAM	SPAM
Classe Real	Não-SPAM	Verdadeiro Negativo	Falso Positivo
	SPAM	Falso Negativo	Verdadeiro Positivo

- Verdadeiro positivo (VP): ocorre quando no conjunto real, a classe que estamos buscando foi prevista corretamente. Por exemplo quando o email é spam e o modelo previu corretamente que ele é spam.

- Falso Positivo (FP): ocorre quando no conjunto real, a classe que estamos buscando foi prevista incorretamente. Exemplo: o email não é spam, mas o modelo afirmou que é spam.

- Verdadeiro negativo (VN): ocorre quando no conjunto real, a classe que não estamos buscando foi prevista corretamente. Exemplo: o email não é spam e o modelo concluiu que ele realmente não é spam.

- Falso negativo (FN): ocorre quando no conjunto real, a classe que não estamos buscando prever foi prevista incorretamente. Por exemplo, o email é spam e o modelo previu que ele não é spam.

A medida de acurácia tradicional (total de acerto) é igual a VP + VN / Total. Ou seja, mede a quantidade de acertos do modelo em relação ao total de classes. Esta medida é a mais clássica, mas no nosso caso nossa amostra é desbalanceada, ou seja, temos mais emails sem spam do que aqueles que possuem spam. Assim, a acurácia sozinha não consegue trazer uma boa precisão para o modelo. Suponha que de uma amostra de 100 emails de uma base de dados, 99 possuem spam. Logo, não precisamos de modelo de previsão bastar assumir não-spam (mais frequente) para ter um ótimo nível de acurácia. Logo, isto não indica que o algoritmo implementado (apostar no mais frequente) possui um nível de acerto satisfatório.

Agora trataremos de algumas situações para lançar mão das medidas de precisão e revocação (recall). Exemplificando, podemos citar:

- Imaginemos um algoritmo que classifique se um filme pode ser indicado para crianças. Nesse caso, não desejamos que o algoritmo classifique um filme para crianças quando na verdade ele não deveria ser sugestionado. Assim a precisão é conceito de maior relevância, pois pretendemos ter um maior número de verdadeiros positivos em relação aos falsos positivos. A precisão é determinada por: VP / VP + FP

		Classe Prevista	
		Filme Adulto	Filme Infantil
Classe Real	Filme Adulto	Verdadeiro Negativo	Falso Positivo (crianças assistem filme adulto)
	Filme Infantil	Falso Negativo	Verdadeiro Positivo

- O recall (revocação) é obtdo por: VP / VP + FN e pode ser usada em uma situação em que os Falsos Negativos são considerados mais prejudiciais do que os Falsos Positivos. Por exemplo, o modelo deve de qualquer maneira encontrar todos os pacientes doentes, mesmo que classifique alguns saudáveis como doentes (situação de FP) no processo. Ou seja, o modelo deve ter alta revocação, pois classificar pacientes doentes como saudáveis pode ser uma tragédia.
-

		Classe Prevista	
		Não Doente	Doente
Classe Real	Não Doente	Verdadeiro Negativo	Falso Positivo
	Doente	Falso Negativo	Verdadeiro Positivo

- Caso queiramos um equilíbrio entre as medidas de precisão e revocação, podemos utilizar a pontuação F1 que nada mais é do que uma média harmônica entre as medidas acima. Assim, quando F1 possuir um valor pequeno, sugere-se que a precisão ou o recall está baixo.

No nosso exemplo, determinaremos a matriz de confusão pelos seguintes comandos:

```
from sklearn.metrics import classification_report

>> print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.87      0.96      0.91       162
          1       0.96      0.86      0.91       168

avg / total       0.91      0.91      0.91       330
```

Neste caso, o modelo conseguiu classificar satisfatoriamente os dados devido a apresentar um alto f1-score de 91%.

4 Modelo de Florestas Aleatórias (*Random Forest*)

Este nome explica muito bem o funcionamento do algoritmo de Machine Learning, que irá criar muitas árvores de decisão, de maneira aleatória, formando o que podemos enxergar como uma floresta, onde cada árvore será utilizada na escolha do resultado final.

Para entender o algoritmo de Random Forest, precisamos primeiramente conhecer os Métodos Ensemble, dos quais ele faz parte. Estes métodos são construídos da mesma forma que algoritmos mais básicos, como regressão linear, árvore de decisão ou KNN, por exemplo, mas possuem uma característica principal que os diferenciam, a combinação de diferentes modelos para se obter um único resultado. Essa característica torna esses algoritmos mais robustos e complexos, levando a um maior custo computacional que costuma ser acompanhado de melhores resultados na amostra teste.

Normalmente na criação de um modelo, escolhemos o algoritmo que apresenta o melhor desempenho para os dados em questão. Podemos testar diferentes configurações deste algoritmo escolhido, gerando assim diferentes modelos, mas no fim do processo de Machine Learning, escolhemos apenas um. Com

um método ensemble serão criados vários modelos diferentes a partir de um algoritmo, mas não escolheremos apenas um para utilização final, e sim todos.

Com esta metodologia poderemos ter um resultado para cada modelo criado. Se criarmos 100 modelos, teremos 100 resultados, que serão agregados em apenas um. Em problemas de regressão poderá ser utilizada a média dos valores para obtenção do resultado final, e em problemas de classificação o resultado que mais se repete será o escolhido. Há casos nos quais o resultado de um modelo será utilizado na criação do próximo, criando uma dependência entre os modelos, e levando a um único resultado final, gerado a partir de vários resultados intermediários.

Muitos métodos ensemble dependem do conceito de árvore de decisão, sendo de grande valia o conhecimento deste conceito no aprendizado dos métodos. Inclusive, quem já conhece árvores de decisão aprenderá os métodos ensemble com muita facilidade e rapidez.

Diferentemente do que acontece na criação de uma árvore de decisão simples, ao utilizar o Random Forest, o primeiro passo executado pelo algoritmo será selecionar aleatoriamente algumas amostras dos dados de treino, e não a sua totalidade. Nesta etapa é utilizado o *bootstrapping*, que é um método de reamostragem onde as amostras selecionadas podem ser repetidas na seleção. Com esta primeira seleção de amostras será construída a primeira árvore de decisão. Trata-se de um procedimento puramente estatístico.

No Random Forest a definição desta variável não acontece com base em todas as variáveis disponíveis. O algoritmo irá escolher de maneira aleatória duas ou mais variáveis, e então realizar os cálculos com base nas amostras selecionadas, para definir qual dessas variáveis será utilizada no primeiro nó. Para escolha da variável do próximo nó, novamente serão escolhidas duas (ou mais) variáveis, excluindo as já selecionadas anteriormente, e o processo de escolha se repetirá. Desta forma a árvore será construída até o último nó. A quantidade de variáveis a serem escolhidas pode ser definida na criação do modelo.

Na construção da próxima árvore, os dois processos anteriores se repetirão, levando a criação de uma nova árvore. Provavelmente essa árvore será diferente da primeira, pois tanto na seleção das amostras, quanto na seleção das variáveis, o processo acontece de maneira aleatória.

Neste sentido, podemos elaborar quantas árvores desejarmos, sendo que quanto mais árvores forem criadas, melhor podem ser os resultados obtidos, onde uma nova árvore não conseguirá levar a uma melhora significativa no desempenho do modelo. É importante lembrarmos que quanto mais árvores forem criadas, maior será o tempo de criação do modelo.

Com o modelo de machine learning devidamente elaborado, podemos apresentar novos dados e obter o resultado da previsão. Cada árvore criada irá apresentar o seu resultado, sendo que em problemas de regressão será realizada a média dos valores previstos, e esta média informada como resultado final, e em problemas de classificação o resultado que mais vezes foi apresentado será o escolhido.

No nosso caso, um analista deseja projetar um modelo de Random Forest para modelar o comportamento dos clientes que desistem de um pacote de assinatura de TV a Cabo referente a uma grande empresa no Brasil. Para tanto, tomou-se a variável dependente (cancel) que informa se houve a desistência no período de 6 meses após o contrato (valor igual a um) ou se não houve a desistência no período de 6 meses após o contrato (valor igual a zero). As variáveis independentes (atributos) são:

- idade: idade dos clientes em anos
- linhas: quantidade de linhas que o cliente id possui em seu nome
- temp_cli: tempo do cliente em meses com o plano de assinatura da TV a Cabo
- renda: renda mensal média do cliente durante o plano de assinatura
- fatura: fatura média mensal do cliente durante o plano de assinatura da TV a Cabo
- temp_rsd: tempo de residência em anos do cliente
- netflix: se possui Netflix (1)

Nosso DataFrame **df** pode ser visualizado como mostra a Figura a seguir:

id	idade	linhas	temp_cli	renda	fatura	temp_rsd	netflix	cancel
1	51	4	26	5320.0	543	7.3	1	0
2	36	2	16	5620.0	482	4.5	1	0
3	35	1	15	4860.0	593	4.8	0	0
5	40	1	22	6590.0	1184	6.2	1	0
6	52	1	30	6370.0	634	2.2	0	0
9	38	4	16	6120.0	146	5.6	1	0
10	27	1	18	5600.0	1273	4.8	0	0
11	45	3	29	10080.0	717	4.1	1	0
12	35	1	12	4720.0	446	6.9	0	0
13	30	1	21	5840.0	184	7.2	1	0

Realizando a divisão de treino e teste conforme os comandos já aprendidos em seções anteriores, temos:

```
from sklearn.model_selection import train_test_split
X = df.drop('cancel', axis = 1)
y = df['cancel']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30,
random_state = 101)
```

No caso acima, teremos uma amostra de teste com 30% das observações no total da amostra original e outra amostra treino com 70% de observações da mesma. Os rótulos **X_train, y_train** representam os atributos e a variável dependente da amostra treino e os rótulos **X_test, y_test** designam os atributos e a variável dependente da amostra teste.

Para rodar o Random Forest no Python procederemos com as instâncias do `sklearn` igualmente ao que foi feito com os modelos de regressão.

```
from sklearn.ensemble import RandomForestClassifier
```

```
rfc = RandomForestClassifier(random_state=100, criterion = 'gini', max_depth=10,
n_estimators=50, n_jobs = -1)
```

Para ajustar o modelo de Random Forest utilizaremos o método `fit()`

```
rfc.fit(X_train, y_train)
```

Também podemos medir a importância de cada um dos atributos utilizados. Com isso podermos ver quais são os principais atributos que conseguem discriminar satisfatoriamente o status do cliente.

```
features = X_test.columns
features_importance = zip(rfc.feature_importances_,features)
for importance, feature in sorted(features_importance, reverse = True):
  print("%s: %f%%" % (feature, importance*100))
```

```
fatura: 29.878961%
renda: 17.647012%
temp_cli: 17.342463%
temp_rsd: 13.899938%
idade: 13.674215%
linhas: 5.415295%
netflix: 2.142116%
```

Percebemos que os três principais atributos são: fatura, renda e tempo dos clientes para poder segregar o status de cancelamento do plano de TV por assinatura.

Em relação ao poder de previsão na amostra teste, elaboraremos a matriz de confusão com os comandos abaixo:

```
previsoes = rfc.predict(X_test)

from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,previsoes))
```

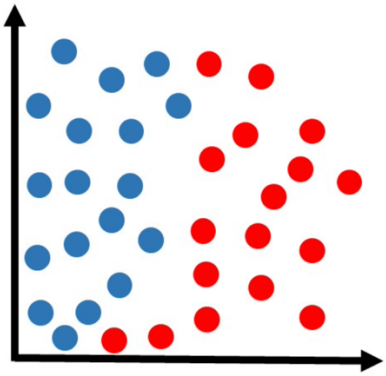
```
              precision    recall  f1-score   support

           0       0.82      0.91      0.86       461
           1       0.53      0.35      0.42       139

    accuracy                           0.78       600
   macro avg       0.67      0.63      0.64       600
weighted avg       0.75      0.78      0.76       600
```

A acurácia é de 78%, ou seja, o total de acertos de classificações se mostra adequado quando consideramos o desempenho dentro da amostra de teste. O f1-escore também se mostra elevado, evidenciando um equilíbrio entre a precisão e a revocação do modelo de Random Forest.

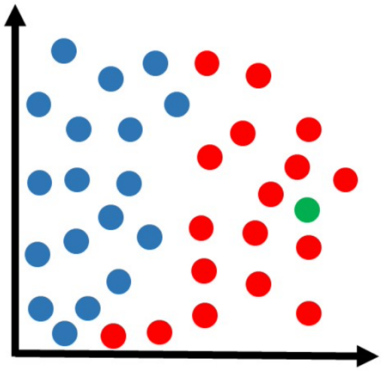
5 Modelo KNN (K-Nearest Neighbors)

O KNN (K-nearest neighbors, ou “K-vizinhos mais próximos”) é muito utilizado em problemas de classificação nas áreas de negócios, marketing e finanças

Para entender como o KNN funciona, primeiro considere que temos um conjunto de dados dividido em duas classes: azul (bons clientes) e vermelho (maus clientes), conforme a figura abaixo.

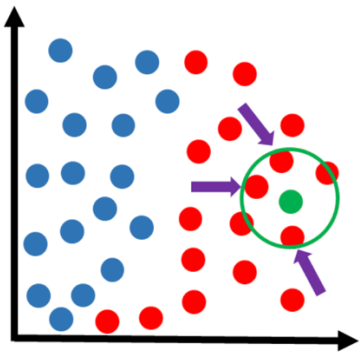


Agora recebemos uma amostra que ainda não está classificada, e gostaríamos de definir se ela pertence à classe azul ou à classe vermelha. Digamos que essa nova amostra (cor verde na figura abaixo) esteja localizada nessa região:



Intuitivamente, podemos observar que faz mais sentido classificar essa amostra como pertencendo à classe vermelha. Mas o algoritmo não possui “intuição”, ele precisa de um cálculo matemático para poder definir a solução.

Observamos que a classe dos vizinhos mais próximos, em uma votação onde a maioria vence. Por exemplo, vamos supor que estamos analisando os 3 vizinhos mais próximos.



Na figura acima, podemos ver que os 3 vizinhos mais próximos pertencem à classe vermelha. Então como houve 3 votos a zero para a classe vermelha, essa amostra fica sendo classificada como pertencente ao grupo dos vermelhos!

Suponha que tenhamos um banco de dados com indicadores financeiros (eles foram criptografados neste exemplo com códigos) e uma outra variável meta que indica 1, se a empresa teve um desempenho acima da mediana dos retornos de seu setor e 0, se a empresa não teve um desempenho acima da mediana dos retornos de seu setor.

Os comandos para rodar o modelo KNN são:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.30)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1) #Realizando o KNN para 1 vizinho
próximo
knn.fit(X_train,y_train) #Ajuste do modelo com a amostra treino

prediction = knn.predict(X_test) #Previsões com a amostra teste

from sklearn.metrics import classification_report,confusion_matrix
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.43      0.47      0.45       135
           1       0.53      0.49      0.51       165

    accuracy                           0.48       300
   macro avg       0.48      0.48      0.48       300
weighted avg       0.48      0.48      0.48       300
```

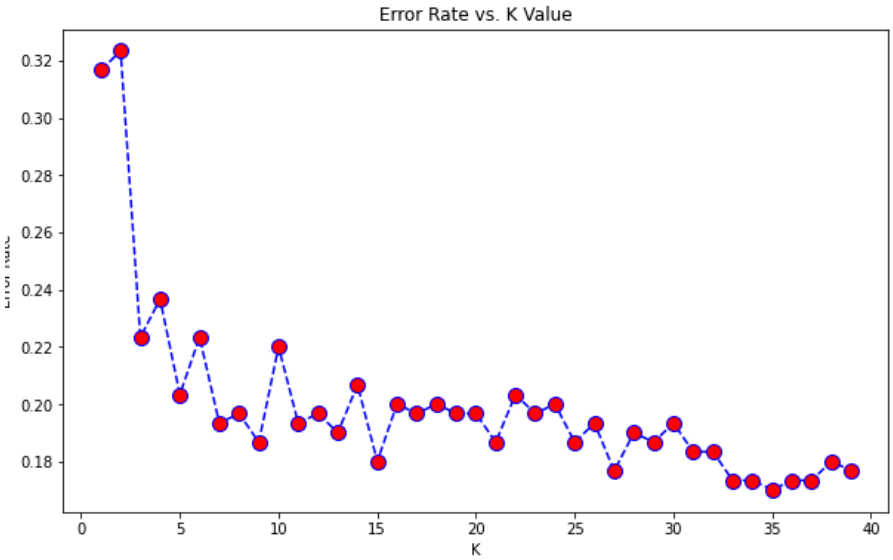
Ou seja, para um vizinho mais próximo a acurácia foi somente de 48%, implicando em escolher outros vizinhos para otimizar o ajuste do modelo do KNN nos dados de teste.

Vamos criar uma estrutura condicional com for que treine vários modelos KNN para diferentes valores de k e, em seguida, mantenha um registro do `error_rate` para cada um desses modelos.

```
error_rate = []

for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))
```

A estrutura acima armazenará em uma lista `error_rate` a média da quantidade de ocorrências que as classificações foram incorretas. Quando maior for esta taxa, pior é o ajuste do modelo. Assim, podemos plotar o seguinte gráfico:

```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```



Verificamos que para K= 35 alcançamos a menor taxa de erro para o modelo KNN. Logo, para este parâmetro (K = 35), teremos a seguinte matriz de confusão:

	precision	recall	f1-score	support
0	0.80	0.84	0.82	135
1	0.86	0.82	0.84	165
accuracy			0.83	300
macro avg	0.83	0.83	0.83	300
weighted avg	0.83	0.83	0.83	300

A acurácia passou para 83%, mostrando que para K = 35 conseguimos atingir um nível satisfatório de discriminação das classes do nosso problema de desempenho financeiro das empresas.