Fundamentos Matemáticos e Computacionais de Machine Learning

Especialização em Machine Learning e Big Data

UEL

Profa. Dra. Juliana Felix

jufelix16@uel.br

Bibliotecas

Scikit-Learn

Scikit Learn

- Scikit-learn (Sklearn) é uma biblioteca útil e robusta para aprendizado de máquina em Python.
- É a biblioteca mais famosa também.

Scikit Learn

- Esta biblioteca fornece uma seleção de ferramentas eficientes para aprendizado de máquina e modelagem estatística, incluindo:
  - classificação;
  - regressão;
  - agrupamento e redução de dimensionalidade.
- Esta biblioteca, que é amplamente escrita em Python, é construída sobre NumPy, SciPy e Matplotlib.

Scikit Learn

Chamava-se originalmente scikits.learn e foi inicialmente desenvolvido por David Cournapeau como um projeto Google Summer of Code em 2007.

Scikit Learn

Em 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort e Vincent Michel, do FIRCA (French Institute for Research in Ciência da Computação e Automação), levaram este projeto a outro nível:

- O primeiro lançamento público (v0.1 beta) ocorreu em 1º de fevereiro de 2010.

Scikit Learn

- A versão estável mais atual do Sci-kit learn é a 1.2.2 de março de 2023
  - Version 1.2.2 — scikit-learn 1.2.2 documentation
- A próxima versão está em desenvolvimento e é a 1.3
  - Version 1.3.0 — scikit-learn 1.3.dev0 documentation

Scikit Learn

- Os principais requisitos para o Scikit-Learn são
  - Python (>=3.8)
  - NumPy (>= 1.14.0)
  - Scipy (>= 1.1.0)
  - Joblib (>= 1.1.1)
  - Matplotlib (>= 3.1.3)
  - Pandas (>= 1.0.5)
- Mais detalhes em
  - Installing scikit-learn

Scikit Learn

Instalação

Para a instalação do Scikit Learn você pode usar:

- PIP

```
pip install -U scikit-learn
```

- Conda (recomendável caso esteja usando distribuições como o Anaconda).

```
conda install scikit-learn
```

Principais características

- Muitas bibliotecas concentram-se em carregamento, manipulação e resumo de dados, já a biblioteca Scikit-learn se concentra na modelagem dos dados.
- Alguns dos grupos de modelos de Machine Learning mais populares são fornecidos pelo Sklearn.

Principais características

- **Algoritmos de Aprendizagem Supervisionada**
  - Quase todos os algoritmos populares de aprendizagem supervisionada fazem parte do scikit-learn, como:
    - Regressão Linear
    - Máquina de Vetor de Suporte (SVM)
    - Árvore de Decisão
    - etc.

Principais características

- **Algoritmos de aprendizado não supervisionado**
  - Também possui todos os algoritmos populares de aprendizado não supervisionado desde
    - Clustering
    - Análise fatorial
    - PCA (Análise de Componentes Principais) até
    - Redes neurais não supervisionadas

Principais características

- **Clustering**
  - usado para agrupar dados não rotulados.
- **Validação Cruzada**
  - usado para verificar a precisão de modelos supervisionados em dados não vistos.

Principais características

- **Redução de dimensionalidade**
  - usado para reduzir o número de atributos em dados que podem ser usados posteriormente para resumo, visualização e seleção de recursos.
- **Métodos de conjunto**
  - usado para combinar as previsões de vários modelos supervisionados.

Principais características

- **Extração de recursos**
  - usado para extrair os recursos dos dados para definir os atributos em dados de imagem e texto.
- **Seleção de recursos**
  - É usado para identificar atributos úteis para criar modelos supervisionados.

Principais características

- **Open Source**
  - É uma biblioteca de código aberto e também utilizável comercialmente sob a licença BSD[1].

[1] Licenças BSD e GPL – Wikipédia, a enciclopédia livre.

Datasets no Scikit-Learn

- Uma coleção de dados é chamada de conjunto de dados.
- Datasets em Scikit-Learn possui dois componentes:
  - **Features (Características)**
  - **Response (Resposta)**

Datasets no Scikit-Learn

- **Features (Características)**
  - As variáveis dos dados são chamadas de suas características.
  - Elas também são conhecidas como
    - Preditores,
    - Entradas, ou
    - Atributos.

Datasets no Scikit-Learn

- **Response (Resposta)**
  - É a variável de saída que depende basicamente das variáveis de característica.
  - Ela também é conhecida como
    - Destino,
    - Rótulo, ou
    - Saída.

Datasets no Scikit-Learn

- Features (características) podem ser divididas em
  - **Matriz de características**
    - É o conjunto de características, caso haja mais de uma.
  - **Rótulos de recursos**
    - É a lista de todos os nomes dos recursos.

Datasets no Scikit-Learn

- Response (resposta) pode ser dividida em
  - **Vetor de resposta**
    - É usado para representar a coluna de resposta.
    - Geralmente, temos apenas uma coluna de resposta.
  - **Target Names (Nomes Alvo)**
    - Representam os possíveis valores tomados por um vetor de resposta.

Datasets no Scikit-Learn

- O Scikit-learn tem poucos conjuntos de dados de exemplo, como
  - Regressão: preços das casas de Boston
  - Classificação: íris e dígitos

Dataset Exemplo para Classificação
Iris Dataset

- O primeiro conjunto de dados que você verá em um tutorial introdutório sobre aprendizado de máquina é o "conjunto de dados Iris".
- Este é o "hello world!" do machine learning.

Dataset Exemplo para Classificação
Iris Dataset

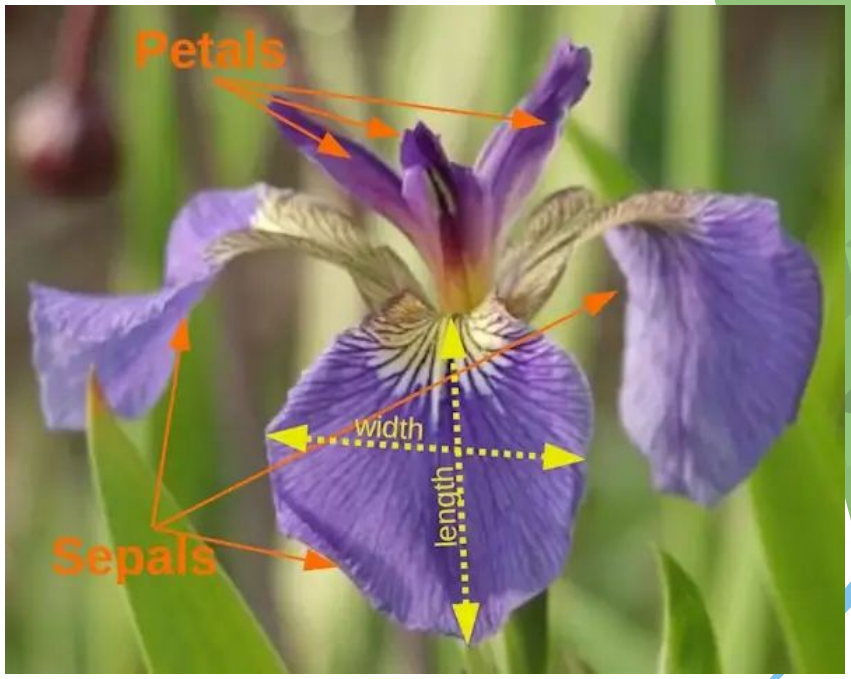
- O Dataset Iris contém as medições de 150 flores de íris de 3 espécies diferentes (50 de cada espécie):
  - Iris-Setosa,
  - Iris-Versicolor, e
  - Iris-Virginica.

Dataset Exemplo para Classificação
Iris Dataset

- O conjunto de dados da íris é frequentemente usado por sua simplicidade.
- Esse conjunto de dados está contido no scikit-learn.

Dataset Exemplo para Classificação
Iris Dataset

- Features no conjunto de dados Iris:
  - comprimento da sépala em cm
  - largura da sépala em cm
  - comprimento da pétala em cm
  - largura da pétala em cm
- Classes alvo para prever:
  - Iris Setosa
  - Iris Versicolor
  - Iris Virgínia



Dataset Exemplo para Classificação
Iris Dataset

- O scikit-learn incorpora uma cópia do arquivo CSV da íris junto com uma função auxiliar para carregá-lo em arrays numpy.

```
from sklearn.datasets import load_iris

iris = load_iris()
```

- O resultado é um objeto do tipo Bunch:

```
type(iris)
```

- Saída:

```
sklearn.utils.Bunch
```

Dataset Exemplo para Classificação
Iris Dataset

- Você pode ver o que está disponível para este tipo de dados usando o método keys():

```
iris.keys()
```

- Saída:

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR',
'feature_names', 'filename'])
```

Dataset Exemplo para Classificação
Iris Dataset

- Um objeto Bunch é semelhante a um dicionário, mas adicionalmente permite acessar as chaves em um estilo de atributo:

```
print(iris["target_names"])

print(iris.target_names)
```

- Saída:

```
['setosa' 'versicolor' 'virginica']

['setosa' 'versicolor' 'virginica']
```

Dataset Exemplo para Classificação
Iris Dataset

- As características de cada flor de amostra são armazenadas no atributo de dados do conjunto de dados:

```
n_samples, n_features = iris.data.shape

print('Number of samples:', n_samples)

print('Number of features:', n_features)

print(iris.data[0])
```

- Saída:

```
Number of samples: 150

Number of features: 4

[5.1 3.5 1.4 0.2]
```

Dataset Exemplo para Classificação
Iris Dataset

- As características de cada flor são armazenadas no atributo de dados do conjunto de dados.

```
iris.data[[12, 26, 89, 114]]
```

- Saída:

```
array([[4.8, 3. , 1.4, 0.1],
       [5. , 3.4, 1.6, 0.4],
       [5.5, 2.5, 4. , 1.3],
       [5.8, 2.8, 5.1, 2.4]])
```

Dataset Exemplo para Classificação
Iris Dataset

- As informações sobre a classe de cada amostra do nosso conjunto de dados Iris são armazenadas no atributo target do conjunto de dados:

```
print(iris.target)
```

- Saída:

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Dataset Exemplo para Classificação
Iris Dataset

- `bincount` de NumPy conta o número de ocorrências de cada valor em uma matriz de inteiros não negativos.

Dataset Exemplo para Classificação
Iris Dataset

- Podemos usar isso para verificar a distribuição das classes no conjunto de dados:

```
import numpy as np

np.bincount(iris.target)
```

  - Saída:

```
array([50, 50, 50])
```

- Podemos ver que as classes estão distribuídas uniformemente - são 50 flores de cada espécie, ou seja,
  - classe 0: Iris-Setosa
  - classe 1: Íris-Versicolor
  - classe 2: Iris-Virginica

Dataset Exemplo para Classificação
Iris Dataset

- Esses nomes de classe são armazenados no último atributo, ou seja, target_names:

  ```
  print(iris.target_names)
  ```

- Saída:

  ```
  ['setosa' 'versicolor' 'virginica']
  ```

Dataset Exemplo para Classificação
Iris Dataset

- Além da forma dos dados, também podemos verificar a forma dos rótulos, ou seja, o target.shape

```
print(iris.data.shape)

print(iris.target.shape)
```

- Saída:

```
(150, 4)

(150,)
```

Dataset Exemplo para Classificação
Iris Dataset

- Cada amostra de flor é uma linha na matriz de dados e as colunas (características) representam as medidas da flor em centímetros.
- Por exemplo, podemos representar este conjunto de dados Iris, consistindo de 150 amostras e 4 características, uma matriz bidimensional ou matriz $R^{150 \times 4}$ no seguinte formato:

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(150)} & x_2^{(150)} & x_3^{(150)} & x_4^{(150)} \end{bmatrix}.$$

O sobrescrito denota a i-ésima linha e o subscrito denota o j-ésimo recurso, respectivamente.

Dataset Exemplo para Classificação
Iris Dataset

- Geralmente, temos n linhas e k colunas:

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \dots & x_k^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \dots & x_k^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \dots & x_k^{(n)} \end{bmatrix}.$$

Dataset Exemplo para Classificação
Iris Dataset

- Geralmente, temos n linhas e k colunas:

$$\mathbf{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \dots & x_k^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \dots & x_k^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \dots & x_k^{(n)} \end{bmatrix}.$$

```
print(iris.data.shape)

print(iris.target.shape)
```

- Saída:

```
(150, 4)

(150,)
```

Visualizando o íris dataset

- Os dados (features) são quadridimensionais, mas podemos visualizar uma ou duas das dimensões de cada vez usando um simples histograma ou gráfico de dispersão.

```
from sklearn.datasets import load_iris
iris = load_iris()
print(iris.data[iris.target==1][:5])
print(iris.data[iris.target==1, 0][:5])
```

- Saída:

```
[[7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]]
 [7.  6.4 6.9 5.5 6.5]
```

Histograma de features

```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
x_index = 3
colors = ['blue', 'red', 'green']
for label, color in zip(range(len(iris.target_names)), colors):
     ax.hist(iris.data[iris.target==label, x_index],
          label=iris.target_names[label],
          color=color)

ax.set_xlabel(iris.feature_names[x_index])
ax.legend(loc='upper right')
plt.show()
```

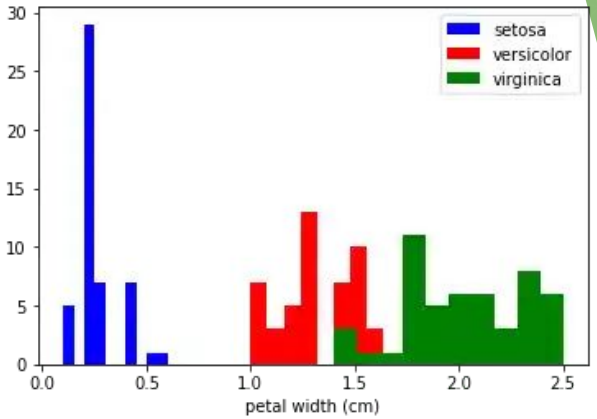


Gráfico de dispersão com dois recursos

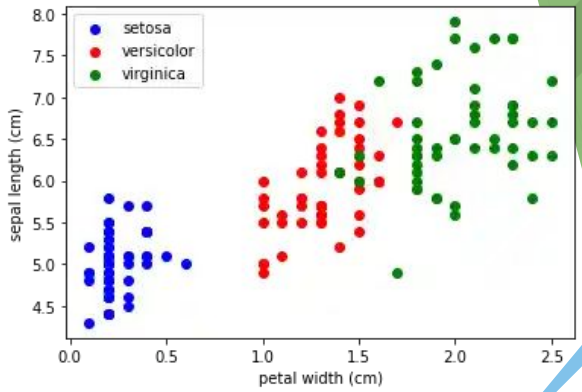
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()

x_index = 3
y_index = 0

colors = ['blue', 'red', 'green']

for label, color in zip(range(len(iris.target_names)), colors):
      ax.scatter(iris.data[iris.target==label, x_index],
                 iris.data[iris.target==label, y_index],
                 label=iris.target_names[label],
                 c=color)

ax.set_xlabel(iris.feature_names[x_index])
ax.set_ylabel(iris.feature_names[y_index])
ax.legend(loc='upper left')
plt.show()
```

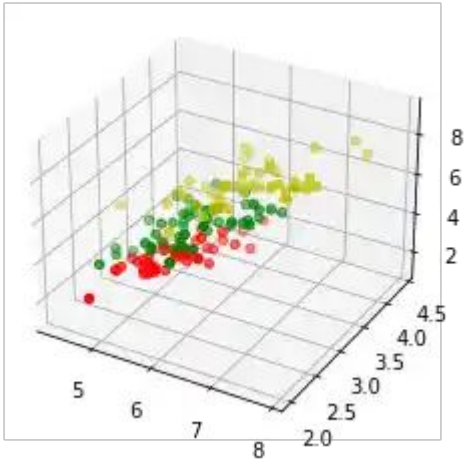


Visualização 3-D

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from mpl_toolkits.mplot3d import Axes3D
iris = load_iris()
X = []
for iclass in range(3):
      X.append([[], [], []])
      for i in range(len(iris.data)):
      if iris.target[i] == iclass:
            X[iclass][0].append(iris.data[i][0])
            X[iclass][1].append(iris.data[i][1])
            X[iclass][2].append(sum(iris.data[i][2:]))

colours = ("r", "g", "y")
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

for iclass in range(3):
      ax.scatter(X[iclass][0], X[iclass][1], X[iclass][2], c=colours[iclass])
plt.show()
```



Classificando com o SKLearn

- Para **classificar** com o Scikit learn há duas funções básicas:
  - Fit - ajusta os parâmetros do modelo
    - Para a regressão logística, significa ajustar os Thetas
  - Predict - realiza a entrada no modelo e coleta a saída
    - Para a regressão logística, significa efetuar a classificação
- Há também funções úteis para verificar o comportamento do modelo ajustado
  - Score - Classifica, compara com a saída esperada e calcula a taxa de acerto

Classificando com o SKLearn

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
import numpy as np

X, y = load_iris(return_X_y=True)                          # Carrega o Iris dataset

clf = LogisticRegression(random_state=0, max_iter=1000)    # Carrega o algoritmo de regressão logística

clf.fit(X, y)                                              # Ajusta os parâmetros do modelo

r = clf.predict(X)                                         # Realiza a classificação

print(np.column_stack((r,y)))

sc = clf.score(X, y)                                       # Calcula a taxa de acerto
print(sc)
```

Regressão Linear com o SKLearn

- Para exemplificar a Regressão linear vamos utilizar o Dataset California Housing

Número de Amostras	20640
Número de Atributos	8 atributos preditores numéricos e 1 atributo alvo
Descrição dos atributos	• MedInc - Renda mediana do grupo de quarteirões • HouseAge - Idade mediana da casa no grupo de quarteirões • AveRooms - número médio de cômodos por domicílio • AveBedrms número médio de quartos por domicílio • Population - População do grupo de quarteirões • AveOccup - número médio de membros da família • Latitude - latitude do grupo de quarteirões • Longitude - longitude do grupo de quarteirões

- A variável-alvo é o valor médio das casas nos distritos da Califórnia, expresso em centenas de milhares de dólares (US$ 100.000).

Regressão Linear com o SKLearn

- Carregando o dataset California Housing

```
from sklearn.datasets import fetch_california_housing

housing_data = fetch_california_housing()
descr = housing_data['DESCR']
feature_names = housing_data['feature_names']
data = housing_data['data']
target = housing_data['target']

print(descr)
print(feature_names)
```

Regressão Linear com o SKLearn

- Para **ajustar funções** com o Scikit learn há duas funções básicas:
  - Fit - ajusta os parâmetros do modelo
    - Para a regressão linear, significa ajustar os Thetas
  - Predict - realiza a entrada no modelo e coleta a saída
    - Para a regressão linear, significa efetuar a estimativa da saída da função
- Há também funções úteis para verificar o comportamento do modelo ajustado
  - Score - Apresenta as entradas, compara com a saída esperada e calcula a taxa de acerto

Regressão Linear com o SKLearn

```
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
import numpy as np

housing_data = fetch_california_housing()          # Carrega o dataset california housing
data = housing_data['data']                        # Extrai variáveis de entrada
target = housing_data['target']                    # Extrai a variável de saída

linreg = LinearRegression()                        # Carrega o algoritmo de regressão linear
linreg.fit(data, target)                           # Ajusta os parâmetros do modelo

r = linreg.predict(data)                           # Realiza a estimativa com a função ajustada

print(np.column_stack((r, target, r-target)))

sc = linreg.score(data, target)                    # Calcula a taxa de acerto
print(sc)
```

Bibliotecas

Statsmodels

Statsmodels

- *Statsmodels* é um módulo Python que fornece classes e funções para a estimativa de muitos modelos estatísticos diferentes, bem como para a realização de testes estatísticos e exploração de dados estatísticos.
- Podemos explorar os dados, estimar vários modelos estatísticos e até mesmo testar os modelos estatisticamente usando o pacote do Python chamado *statsmodels*.

Statsmodels

- Statsmodels é um pacote de programação Python e pertence à pilha de módulos que trata do domínio científico e tem sua implementação em tecnologias futuras, incluindo análise de dados, estatística e ciência de dados.
- Pode ser considerado o pacote complementar ao módulo de estatísticas denominado SciPy.

Statsmodels

- Essa biblioteca ou pacote é criado em cima dos pacotes SciPy e NumPy e também faz a manipulação de dados usando pandas e possui a interface patsy para a fórmula que se assemelha ao R-like.
- O matplotlib é a biblioteca da qual as funções gráficas são usadas.
- Muitos outros pacotes Python consideram este a base para criar bibliotecas de estatísticas.

Statsmodels

- Scipy.stats era o módulo do pacote scipy e foi escrito inicialmente por Jonathan Taylor, mas depois foi removido e um pacote completamente novo foi criado.
- Muitas melhorias, testes rigorosos e correções foram feitas no Google Summer of Code 2009 e, finalmente, o pacote com os statsmodels foi lançado.

Statsmodels

- Mesmo hoje em dia, muitos dos modelos estatísticos, ferramentas para plotagem e novos modelos estão surgindo e introduzidos no mercado com desenvolvimento contínuo pela equipe do statsmodel.

Statsmodels

- Podemos trabalhar com estatísticas de uma forma que nenhuma outra plataforma nos permite pois o próprio statsmodel é feito tendo em mente o propósito das estatísticas hardcore.
- Tem mais inclinação para R e é uma ótima ferramenta para analisar dados estatísticos.
- A maioria dos desenvolvedores que programam em R podem fazer uso disso e podem facilmente mudar para Python usando este pacote.

Statsmodels

- Requisitos para instalação
  - Python >= 3.8
  - NumPy >= 1.18
  - SciPy >= 1.4
  - Pandas >= 1.0
  - Patsy >= 0.5.2

Statsmodels

- Dependências opcionais
  - O cvxopt é necessário para o ajuste regularizado de alguns modelos.
  - Matplotlib >= 3 é necessário para plotar funções e executar muitos dos exemplos.
  - Se instalado, o X-12-ARIMA ou X-13ARIMA-SEATS pode ser usado para análise de séries temporais.
  - pytest é necessário para executar o conjunto de testes.
  - IPython >= 6.0 é necessário para construir os documentos localmente ou para usar os notebooks.
  - joblib >= 1.0 pode ser usado para acelerar a estimativa distribuída para determinados modelos.

Statsmodel

Instalação

- Para a instalação do statsmodel você pode usar:
- PIP

```
pip install statsmodel
```

- Conda (recomendável caso esteja usando distribuições como o Anaconda).

```
conda install statsmodel
```

Usando StatsModels

- Depois que o statsmodel estiver instalado, você poderá usar o pacote statsmodel dentro do seu programa Python simplesmente importando o pacote.

```
import statsmodel
```

Dataset de exemplo

- Alcool e Tabaco

Este dataset visa avaliar se as pessoas que usam produtos de tabaco são mais propensas a consumir álcool.

Link: Tobacco_and_alcohol | DASL

Dataset de exemplo

- Nesta base estão os dados sobre os gastos das famílias (em libras) obtidos pelo governo britânico em 11 regiões da Grã-Bretanha.
- Assim, visava-se analisar se:
  - Os gastos com tabaco e álcool parecem estar relacionados?
  - Que perguntas você tem sobre esses dados?
  - Que conclusões você pode tirar?

Regressão Linear com Statsmodels

```
import statsmodels.api as sm
import numpy as np

data_str = '''Region;Alcohol;Tobacco
     North;6.47;4.03
     Yorkshire;6.13;3.76
     Northeast;6.19;3.77
     East Midlands;4.89;3.34
     West Midlands;5.63;3.47
     East Anglia;4.52;2.92
     Southeast;5.89;3.20
     Southwest;4.79;2.71
     Wales;5.27;3.53
     Scotland;6.08;4.51
     Northern Ireland;4.02;4.56'''

d = data_str.split('\n')
d = [ i.split(';') for i in d ]
reg = []
dados = []
```

```
for r in d:
     reg.append(r[0])
     dados.append(r[1:])

reg = np.array(reg[1:])
dados = np.array(dados[1:],dtype=np.float32)

X = dados[:,0]
Y = dados[:,1]

X = sm.add_constant(X)
model = sm.OLS(Y, X)
results = model.fit()

print(results.summary())
print(results.params)
print(np.column_stack((results.fittedvalues, Y)))
```

Regressão Linear com Statsmodels

```
import statsmodels.api as sm
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)

X = sm.add_constant(X)

model = sm.OLS(y, X).fit()

print(model.summary())

print(results.params)

print(np.column_stack((results.fittedvalues, Y)))
```

 # Sklearn foi usado apenas para
 # carregar a base Iris
 # Carregando a base Iris

 # Adiciona o Theta_0

 # Ajusta o modelo

 # Imprime o relatório

 # Imprime os thetas

 # Imprime dados preditos e
 # dados esperados lado a lado

Regressão Logística com Statsmodels

```
# importing libraries
import statsmodels.api as sm
import pandas as pd

# loading the training dataset
df = pd.read_csv('logit_train1.csv', index_col = 0)

# defining the dependent and independent variables
Xtrain = df[['gmat', 'gpa', 'work_experience']]
ytrain = df[['admitted']]

# building the model and fitting the data
log_reg = sm.Logit(ytrain, Xtrain).fit()

# printing the summary table
print(log_reg.summary())

# loading the testing dataset
df = pd.read_csv('logit_test1.csv', index_col = 0)

# defining the dependent and independent variables
Xtest = df[['gmat', 'gpa', 'work_experience']]
ytest = df['admitted']
```

```
# performing predictions on the test dataset
yhat = log_reg.predict(Xtest)
prediction = list(map(round, yhat))

# comparing original and predicted values of y
print('Actual values', list(ytest.values))
print('Predictions :', prediction)

from sklearn.metrics import (confusion_matrix,
                                    accuracy_score)

# confusion matrix
cm = confusion_matrix(ytest, prediction)
print ("Confusion Matrix : \n", cm)

# accuracy score of the model
print('Test accuracy = ', accuracy_score(ytest,
prediction))
```

Link: https://www.geeksforgeeks.org/logistic-regression-using-statsmodels/

Considerações Finais

- Esta aula cobriu uma introdução ao Scikit-Learn e ao Statsmodels
- O Scikit-learn é largamente mais usado que o Statsmodels, trazendo muito mais funcionalidades, e sendo uma biblioteca mais madura

Considerações Finais

- O Statsmodels é uma biblioteca com bastante funcionalidades para fins estatíticos
- Ambas bibliotecas são bastante ricas e trazem muitas funcionalidades semelhantes