

## 2 Biblioteca Scikit-Learn

# Biblioteca Scikit-Learn

## Conceito

O scikit-learn é uma biblioteca criada para facilitar atividades de dados na área de *machine learning*. É uma biblioteca estruturada na linguagem python, *open source* e comercialmente utilizada com licença Berkeley Source Distribution (BSD) que são um tipo de licença de baixa restrição para *software* de código aberto que não impõe requisitos de redistribuição.

Esta biblioteca oferece uma gama de ferramentas para gerar análises preditivas de dados e faz uso dos benefícios atrelados à linguagem Python para o desenvolvimento de códigos simples, eficientes e reutilizáveis.

A construção desta biblioteca é fundamentada nos pacotes Numpy, Scipy e Matplotlib. Dentre os pacotes oferecidos pelo scikit-learn, podemos destacar tanto o Pandas como o Numpy. Estes pacotes são os pacotes mais utilizados para compor *pipelines* de dados em projetos de ciência de dados, justamente porque possuem muitas funcionalidades que permitem a preparação dos dados, exploração e análises preditivas de dados com maior agilidade e eficiência.

## Campos de Aplicação

Conforme citado acima, a biblioteca scikit-learn foi construída para atender a área de *machine learning*. Desta maneira, dado que existem diversos problemas de negócio com diferentes finalidades, a biblioteca scikit-learn possui uma vasta quantidade de módulos e estimadores que têm como objetivo atender a diferentes necessidades que podemos encontrar em projetos de dados.

Abaixo, compartilho alguns módulos e como os mesmos são utilizados dentro da disciplina de *machine learning*:

- **Pré-processamento:** Etapa responsável por capturar, preparar e explorar os dados. Em média, tende a utilizar 70% do tempo dentro de um projeto de ciência de dados, sendo a etapa mais custosa. O scikit-learn possui diversas funcionalidades que permitem a captação e exploração dos dados através de estatísticas, como também prepará-los com conversões e transformações necessárias.
- **Classificação:** Oferece modelos de classificação para o desenvolvimento de soluções que possam classificar classes ou variáveis dependentes com base em um conjunto de atributos, *features*, instâncias ou variáveis independentes.
- **Regressão:** Permite o desenvolvimento de modelos de regressão, onde são produzidos resultados compostos com valores numéricos e contínuos. Através destes modelos, é



[Script Python](#)[Dataset 1](#)[Dataset 2](#)

```
## Let's Code - Regressão Logística
```

```
#### Carregando as bibliotecas
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

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

```
# esta linha permite ver os gráficos sem precisar chamar a função "show()"
```

```
%matplotlib inline
```

```
#### Carregando dos dados
```

```
df = pd.read_csv('eleicao.csv', sep = ';')
```

```
plt.scatter(df.DESPESAS, df.SITUACAO)
```

```
df.describe()
```

```
#### Verificação da correlação entre variáveis
```

```
coerr = np.corrcoef(df.DESPESAS, df.SITUACAO)
```

```
coerr
```

```
#### Segregando dataset df entre variáveis dependentes e independentes
```

```
X = df.iloc[:, 2].values
```

```
X = X[:, np.newaxis]
```

```
y = df.iloc[:, 1].values
```

```
#### Criação do modelo
```

```
modelo = LogisticRegression()
```

```
modelo.fit(X, y)
```

```
modelo.coef_
```

```
modelo.intercept_
```

```
#### Apresentação dos resultados
```

```
plt.scatter(X, y)
```

```
# Geração de novos dados para gerar a função sigmoide
```

```
X_teste = np.linspace(10, 3000, 100)
```

```
# Implementação da função sigmoide
```

```
def model(x):
```

```
    return 1 / (1 + np.exp(-x))
```

```
# Geração de previsões (variável r) e visualização dos resultados
```

```
r = model(X_teste * modelo.coef_ + modelo.intercept_).ravel()
```

```
plt.plot(X_teste, r, color = 'red')
```

```
#### Verificando novos candidatos
```

```
# Carregamento da base de dados com os novos candidatos
df_novos_candidatos = pd.read_csv('novoscandidatos.csv', sep = ';')

# Mudança dos dados para formato de matriz
despesas = df_novos_candidatos.iloc[:, 1].values
despesas = despesas.reshape(-1, 1)

# Previsões e geração de nova base de dados com os valores originais e as
previsões
previsoes_teste = model_LR.predict(despesas)
df_novos_candidatos = np.column_stack((df_novos_candidatos, previsoes_teste))
df_novos_candidatos
```

## Árvores de Decisão

[Script Python](#)

[Dataset 1](#)

```
### Árvores de Decisão
```

```
# Carregando as bibliotecas
```

```
import pandas as pd
```

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

```
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.metrics import confusion_matrix, accuracy_score
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
# Carregando o dataset
```

```
df_credito = pd.read_csv('credit.csv')
```

```
df_credito.shape
```

```
# Apresentando as 5 primeiras linhas do dataset
```

```
df_credito.head()
```

```
# Segregando as variáveis previsoras e classe
```

```
previsores = df_credito.iloc[:,0:20].values
```

```
classe = df_credito.iloc[:,20].values
```

```
# Realizando a conversão de atributos categóricos para numéricos de acordo
com o respectivo índice
```

```
labelencoder = LabelEncoder()
```

```
previsores[:,0] = labelencoder.fit_transform(previsores[:,0])
```

```
previsores[:,2] = labelencoder.fit_transform(previsores[:,2])
```

```
previsores[:, 3] = labelencoder.fit_transform(previsores[:, 3])
```

```
previsores[:, 5] = labelencoder.fit_transform(previsores[:, 5])
```

```
previsores[:, 6] = labelencoder.fit_transform(previsores[:, 6])
```

```
previsores[:, 8] = labelencoder.fit_transform(previsores[:, 8])
previsores[:, 9] = labelencoder.fit_transform(previsores[:, 9])
previsores[:, 11] = labelencoder.fit_transform(previsores[:, 11])
previsores[:, 13] = labelencoder.fit_transform(previsores[:, 13])
previsores[:, 14] = labelencoder.fit_transform(previsores[:, 14])
previsores[:, 16] = labelencoder.fit_transform(previsores[:, 16])
previsores[:, 18] = labelencoder.fit_transform(previsores[:, 18])
previsores[:, 19] = labelencoder.fit_transform(previsores[:, 19])

# Segregando os dados entre treinamento e teste
X_treinamento, X_teste, y_treinamento, y_teste = train_test_split(previsores,
                                                                    classe,
                                                                    test_size =
0.3,

random_state = 0)

# Criação do modelo
arvore = DecisionTreeClassifier()

# Treinamento do modelo
arvore.fit(X_treinamento, y_treinamento)

# Obtendo as previsões
previsoes = arvore.predict(X_teste)
previsoes

# Confusion Matrix
confusao = confusion_matrix(y_teste, previsoes)
confusao

# Calculando a taxa de acerto
taxa_acerto = accuracy_score(y_teste, previsoes)
taxa_acerto

# Calculando a taxa de erro
taxa_erro = 1 - taxa_acerto
taxa_erro
```

## KNN (K-Nearest Neighbors)

[Script Python](#)

[Dataset 1](#)

```
## KNN - K Nearest Neighbor
```

```
### Importando as bibliotecas
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy as scp
import warnings
warnings.filterwarnings("ignore")

### Importando o dataset
df = pd.read_csv('mushrooms.csv', engine='python', sep=',')

### Explorando o dataset
df.head()

# Informações gerais sobre o dataset
df.info()

# Produção de estatísticas sobre os dados
df.describe()

### Verificação de registros nulos no dataset
def distribuicao (data):
    '''
    Esta função exibirá a quantidade de registros únicos para cada coluna
    existente no dataset

    dataframe -> Histogram
    '''
    # Calculando valores únicos para cada label: num unique labels
    num_unique_labels = data.apply(pd.Series.nunique)

    # plotando valores
    num_unique_labels.plot( kind='bar')

    # Nomeando os eixos
    plt.xlabel('Campos')
    plt.ylabel('Número de Registros únicos')
    plt.title('Distribuição de dados únicos do DataSet')

    # Exibindo gráfico
    plt.show()

distribuicao(df)

### Análise de distribuição dos dados da classe Y (Venenoso = p, Comestível = e)
e = pd.value_counts(df['class']) [0]
```

```
p = pd.value_counts(df['class']) [1]

tam = len(df)

print('Cogumelos Comestiveis: ',e)
print('Cogumelos Venenosos: ',p )

pie = pd.DataFrame([[ 'Comestivel',e],[ 'Venenoso',p]],columns=[ 'Tipo' ,
'Quantidade'])

def pie_chart(data,col1,col2,title):
    labels = {'Comestivel':0,'Venenoso':1}
    sizes = data[col2]
    colors = ['#e5ffcc', '#ffb266']

    plt.pie(sizes, labels=labels, colors=colors,
            autopct='%1.1f%%', shadow=True, startangle=140, labeldistance
=1.2)
    plt.title( title )

    plt.axis('equal')
    plt.show()

pie_chart(pie,'Tipo' , 'Quantidade','Distribuição Percentual Classes de
Cogumelos')

plt.bar(pie.Tipo,pie.Quantidade, color = ['#e5ffcc', '#ffb266'])
plt.title("Distribuição das Classes de Cogumelos")
plt.xlabel("Tipo de Cogumelo")
plt.ylabel('Quantidade de Registros')
plt.show()

### Split do conjunto de dados
# X = colunas de informação, variáveis independentes
X = df.drop('class', axis=1)

# y = Variável dependente, a qual será utilizada para classificar os dados
y = df['class']

# Verificando se X está com a coluna class
X.head()

### Aplicação da técnica One Hot Encoder
```



```
#### Transformar as labels em números.
#### O OneHotEncoder gera novas colunas com valor 0 ou 1, em que 1 representa
a ocorrência daquela característica e 0 a não ocorrência.

#Importando o encoder para transformar as labels em chaves numéricas
from sklearn.preprocessing import OneHotEncoder
Oht_enc = OneHotEncoder()
X = pd.DataFrame(Oht_enc.fit_transform(X).A)
X.shape

### Train Test Split

#### Nesta fase separamos o conjunto de dados em Treinamento e Teste,
definindo o percentual que utilizaremos para teste e para treino do modelo
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.3)

### Feature Scaling

#### Etapa importante que irá reduzir a escala numérica das colunas, para que
todas estejam dentro de uma mesma escala de valor, lembrando que na
matemática os números são infinitos dentro de suas escalas, podendo serem
representados então em diversas escalas diferentes. Se houver medidas com
escalas de valor muito diferentes, a distância calculada pelo algoritmo será
influenciada podendo gerar resultados errôneos.

#Importing librarie
from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

### Creating KNN Model

#### Agora iremos aplicar nossos dados ao algoritmo KNN

#Importando o modelo KNN
from sklearn.neighbors import KNeighborsClassifier

# Definindo o valor de vizinhos
classifier = KNeighborsClassifier(n_neighbors=5)
```



```
#Treinando o modelo, com dados de treinamento
classifier.fit(X_train, y_train)

#### Prevendo os valores de Y para os dados de teste (X_test)
y_pred = classifier.predict(X_test)

### Avaliando o Algoritmo
##### Analisando e validando os resultados obtidos

# Importando métricas para validação do modelo
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Imprimindo a matriz confusa
print("Matriz Confusa: ")
print(confusion_matrix(y_test, y_pred), "\n")

# Imprimindo o relatório de classificação
print("Relatório de classificação: \n", classification_report(y_test,
y_pred))

# Imprimindo o quão acurado foi o modelo
print('Acurácia do modelo: ' , accuracy_score(y_test, y_pred))

### Loop para gerar testes com diferentes valores de vizinho (K)
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 10):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

### Comparando o Error Rate gerado de valores K diferentes
plt.figure(figsize=(12, 6))
plt.plot(range(1, 10), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')

### Aplicando melhor parâmetro para K encontrado
# Treinando o modelo KNN com o melhor parâmetro para K
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=1)
classifier.fit(X_train, y_train)

# Aplicando os valores de teste novamente
y_pred = classifier.predict(X_test)

# Importando métricas para validação do modelo
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Imprimindo a matriz confusa
print("Matriz Confusa: ")
print(confusion_matrix(y_test, y_pred), "\n")

# Imprimindo o relatório de classificação
print("Relatório de classificação: \n", classification_report(y_test,
y_pred))

# Imprimindo o quão acurado foi o modelo
print('Acurácia do modelo: ' , accuracy_score(y_test, y_pred))
```

## Naive Bayes

[Script Python](#)

[Dataset 1](#)

[Dataset 2](#)

### ## Naive Bayes

```
#### Carregando as bibliotecas
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, accuracy_score
from yellowbrick.classifier import ConfusionMatrix

#### Carregando conjunto de dados
credito = pd.read_csv('credit.csv')
credito.shape

# Apresentando as 5 primeiras linhas
credito.head()

#### Alterando para um formato de matriz
```

```
previsores = credito.iloc[:,0:20].values
classe = credito.iloc[:,20].values

#### Transformação dos atributos categóricos em atributos numéricos
labelencoder1 = LabelEncoder()
previsores[:,0] = labelencoder1.fit_transform(previsores[:,0])

labelencoder2 = LabelEncoder()
previsores[:,2] = labelencoder2.fit_transform(previsores[:,2])

labelencoder3 = LabelEncoder()
previsores[:, 3] = labelencoder3.fit_transform(previsores[:, 3])

labelencoder4 = LabelEncoder()
previsores[:, 5] = labelencoder4.fit_transform(previsores[:, 5])

labelencoder5 = LabelEncoder()
previsores[:, 6] = labelencoder5.fit_transform(previsores[:, 6])

labelencoder6 = LabelEncoder()
previsores[:, 8] = labelencoder6.fit_transform(previsores[:, 8])

labelencoder7 = LabelEncoder()
previsores[:, 9] = labelencoder7.fit_transform(previsores[:, 9])

labelencoder8 = LabelEncoder()
previsores[:, 11] = labelencoder8.fit_transform(previsores[:, 11])

labelencoder9 = LabelEncoder()
previsores[:, 13] = labelencoder9.fit_transform(previsores[:, 13])

labelencoder10 = LabelEncoder()
previsores[:, 14] = labelencoder10.fit_transform(previsores[:, 14])

labelencoder11 = LabelEncoder()
previsores[:, 16] = labelencoder11.fit_transform(previsores[:, 16])

labelencoder12 = LabelEncoder()
previsores[:, 18] = labelencoder12.fit_transform(previsores[:, 18])

labelencoder13 = LabelEncoder()
previsores[:, 19] = labelencoder13.fit_transform(previsores[:, 19])

#### Segregando o conjunto de dados entre dados de treino e teste
X_treinamento, X_teste, y_treinamento, y_teste = train_test_split(previsores,
                                                                    classe,
```

test\_size =

```
0.3,

random_state = 0)
X_teste

#### Criando o modelo
# naive_bayes = GaussianNB()
naive_bayes.fit(X_treinamento, y_treinamento)

#### Geração de previsões
previsoes = naive_bayes.predict(X_teste)
previsoes

#### Matriz de confusão
confusao = confusion_matrix(y_teste, previsoes)
confusao

# Calculando a taxa de acerto e de erro
taxa_acerto = accuracy_score(y_teste, previsoes)
taxa_erro = 1 - taxa_acerto
taxa_acerto

# Gerando gráfico da matrix de confusão
v = ConfusionMatrix(GaussianNB())
v.fit(X_treinamento, y_treinamento)
v.score(X_teste, y_teste)
v.poof()

#### Previsão de novos registros

# Carregando novos dados
novo_credito = pd.read_csv('novo_credito.csv')
novo_credito.shape

#### Transformação dos atributos categóricos em atributos numéricos
novo_credito = novo_credito.iloc[:,0:20].values
novo_credito[:,0] = labelencoder1.transform(novo_credito[:,0])
novo_credito[:, 2] = labelencoder2.transform(novo_credito[:, 2])
novo_credito[:, 3] = labelencoder3.transform(novo_credito[:, 3])
novo_credito[:, 5] = labelencoder4.transform(novo_credito[:, 5])
novo_credito[:, 6] = labelencoder5.transform(novo_credito[:, 6])
novo_credito[:, 8] = labelencoder6.transform(novo_credito[:, 8])
novo_credito[:, 9] = labelencoder7.transform(novo_credito[:, 9])
novo_credito[:, 11] = labelencoder8.transform(novo_credito[:, 11])
novo_credito[:, 13] = labelencoder9.transform(novo_credito[:, 13])
```

```
novo_credito[:, 14] = labelencoder10.transform(novo_credito[:, 14])
novo_credito[:, 16] = labelencoder11.transform(novo_credito[:, 16])
novo_credito[:, 18] = labelencoder12.transform(novo_credito[:, 18])
novo_credito[:, 19] = labelencoder13.transform(novo_credito[:, 19])
```

```
#### Resultado da previsão
naive_bayes.predict(novo_credito)
```

## Otimização de Hiperparâmetros

### Script Python

```
### Otimização de Hiperparâmetros

# Definindo variável seed
SEED = 123456

#### Importando a base dados da própria biblioteca Sklearn
from sklearn.datasets import load_breast_cancer

# Carregando dataset
df = load_breast_cancer()

# Importando biblioteca pandas
import pandas as pd

# Formatando dataset em um DataFrame e apresentado a 5 primeiras linhas do
mesmo
df_feature = pd.DataFrame(data=df['data'], columns=df['feature_names'])
df_feature.head()

# Definindo a variável target
df_targets = pd.Series(data=df['target'], name='benign')

# Apresentando os valores únicos da variável target
df_targets.unique()

# Atribuindo features na variável X e target na variável y
X = df_feature
y = df_targets

#### Carregando o algoritmo de Árvores de Decisão para ser o algoritmo na
qual vamos aplicar a otimização
# Carregando as bibliotecas
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_validate
import numpy as np
```

```
# Criando o modelo
modelo_tree = DecisionTreeClassifier()

# Aplicando a técnica de cross validade
results = cross_validate(modelo_tree, X, y, cv=5,
                          scoring=('accuracy'),
                          return_train_score=True)
print(f"mean_train_score {np.mean(results['train_score']):.2f}")
print(f"mean_test_score {np.mean(results['test_score']):.2f}")

#### Abordando o uso da otimização através do Grid Search
# Carregando a biblioteca GridSearchCV
from sklearn.model_selection import GridSearchCV

# Definindo parâmetros
relacao_parametros = {
    "max_depth" : [3, 5],
    "min_samples_split" : [32, 64, 128],
    "min_samples_leaf" : [32, 64, 128],
    "criterion" : ["gini", "entropy"]
}

# Criando modelo
modelo_tree = DecisionTreeClassifier()

# Aplicando o algoritmo com parâmetros definidos anteriormente
clf = GridSearchCV(modelo_tree, relacao_parametros, cv=5,
                   return_train_score=True, scoring='accuracy')

# Treinando o modelo
search = clf.fit(X, y)

# Capturando os resultados e os índices dos melhores parâmetros
results_GridSearchCV = search.cv_results_
indice_melhores_parametros = search.best_index_

# Apresentando a média de score de treino e teste produzida
print(f"mean_train_score {results_GridSearchCV['mean_train_score']
[indice_melhores_parametros]:.2f}")
print(f"mean_test_score {results_GridSearchCV['mean_test_score']
[indice_melhores_parametros]:.2f}")

# Apresentação dos parâmetros
results_GridSearchCV['params'][indice_melhores_parametros]
```

```
#### Abordando o uso da otimização através do Random Search
# Carregando as variáveis
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Definindo relação de parâmetros
relacao_parametros_2 = {
    "max_depth" : randint(1, 10),
    "min_samples_split" : randint(32, 129),
    "min_samples_leaf" : randint(32, 129),
    "criterion" : ["gini", "entropy"]
}

# Criação do modelo
modelo_tree = DecisionTreeClassifier()

clf = RandomizedSearchCV(modelo_tree, relacao_parametros_2,
    random_state=SEED, cv=5, return_train_score=True, n_iter=10,
    scoring='accuracy')
search = clf.fit(X, y)
results_RandomizedSearchCV = search.cv_results_
indice_melhores_parametros = search.best_index_

# Apresentando a média de score de treino e teste produzida
print(f"mean_train_score {results_RandomizedSearchCV['mean_train_score']
[indice_melhores_parametros]:.2f}")
print(f"mean_test_score {results_RandomizedSearchCV['mean_test_score']
[indice_melhores_parametros]:.2f}")

# Apresentação dos parâmetros
results_RandomizedSearchCV['params'][indice_melhores_parametros]
```

## Materiais complementares

- Programação Dinâmica at Youtube: [Primeiros passos com Scikit-Learn | Machine Learning #04](#)
- Let's Code channel at Youtube: [Machine Learning além das previsões](#)
- Let's Code channel at Youtube: [Como instalar bibliotecas no Python](#)
- [Scikit-Learn: Machine Learning in Python — scikit-learn 1.0.2 documentation..](#) Acessado 23 de fevereiro de 2022.

## Referências

- [Scikit-Learn: Machine Learning in Python — scikit-learn 1.0.2 documentation..](#) Acessado 23 de fevereiro de 2022.



< Tópico anterior

Próximo Tópico >