REGRESSÃO

Simples

Fórmula:

$$P = b + m * v$$

Característica:
Uma única variável exploratória.

Hiperparâmetro:
Nenhum.

Código:

```
regressor= LinearRegression()
regressor.fit(X_train,y_train)
prev = regressor.predict(X_test)
```

Métricas de erro:
MAE – `mean_absolute_error(y_test,prev)`

Métricas de desempenho:

```
regressor.score(X_train, y_train)
regressor.score(X_test, y_test)
```

Múltipla

Fórmula:

$$P = b + m1 * v1 + m2 * v2 + ... + mn * vn$$

Característica:
Duas ou mais variáveis exploratórias.

Hiperparâmetro:
Nenhum.

Código:

```
regressor= LinearRegression()
regressor.fit(X_train,y_train)
prev = regressor.predict(X_test)
```

Métricas de erro:
MAE – `mean_absolute_error(y_test,prev)`

Métricas de desempenho:

```
regressor.score(X_train, y_train)
regressor.score(X_test, y_test)
```

Polinomial

Fórmula:

$$P = C + m1 * v1 + m2 * v1^2 + ... + mn * v1\text{^}n$$

Característica:
Utilizamos quando a relação com o target não é linear, ou seja, curva.

Hiperparâmetro:
Degree (grau do polinômio)

Código:

```
from sklearn.preprocessing import
PolynomialFeatures
poly = PolynomialFeatures(degree
= 2)
X_train =
poly.fit_transform(X_train)
X_test = poly.transform(X_test)
Regressor = LinearRegression()
regressor.fit(X_train, y_train)
prev = regressor.predict(X_test)
```

Métricas de erro:
MAE – `mean_absolute_error(y_test,prev)`

Métricas de desempenho:

```
regressor.score(X_train, y_train)
regressor.score(X_test, y_test)
```

Elastic Net

Característica:
Algoritmo regularizador. Reduz overfitting. Usa Ridge para controlar exageros Usa Lasso para eliminar variáveis inúteis. Requer StandardScalar

Hiperparâmetro:
Alpha (Força da regularização)
l1_ratio 0 a 1 (Equilíbrio entre Lasso e Ridge)

Código:

```
from sklearn.linear_model import
ElasticNet
Regressor = ElasticNet(alpha=0.1,
l1_ratio=0.5, random_state=0)
regressor.fit(X_train, y_train)
prev = regressor.predict(X_test)
```

Métricas de erro:
MAE – `mean_absolute_error(y_test,prev)`

Métricas de desempenho:

```
regressor.score(X_train, y_train)
regressor.score(X_test, y_test)
```

Classificação

Naive Bayes

Fórmula:

	História do crédito			Dívida		Garantias		Renda anual		
Risco de crédito	Boa 5	Desconhecida 5	Ruim 4	Alta 7	Baixa 7	Nenhuma 11	Adequada 3	< 15 3	>= 15 <= 35 4	> 35 7
Alto 6/14	1/6	2/6	3/6	4/6	2/6	6/6	0	3/6	2/6	1/6
Moderado 3/14	1/3	1/3	1/3	1/3	2/3	2/3	1/3	0	2/3	1/3
Baixo 5/14	3/5	2/5	0	2/5	3/5	3/5	2/5	0	0	5/5

Característica:
Decora a tabela de probabilidade, não considera a relação entre features.

Hiperparâmetro:
Nenhum.

Código:

```
naive = GaussianNB()
naive.fit(X_train, y_train)
prev = naive.predict(X_test)
```

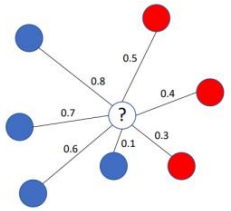
Métricas de erro:
Matriz Confusão –

```
cm = ConfusionMatrix(naive_census)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Métricas de desempenho:

```
accuracy_score(y_test, prev)
```

KNN



$$DE(x,y) = \sqrt{\sum_{i}^{p}(x_i - y_i)^2}$$

Característica:
Calcula a distância euclidiana dos dados a serem previstos com os existentes e verifica a classe de acordo com os vizinhos mais próximos.

Hiperparâmetro:
n_neighbors (número de vizinhos)

Código:

```
knn =
KNeighborsClassifier(n_neighbors=
10)
knn.fit(X_train, y_train)
prev = knn.predict(X_test)
```

Métricas de erro:
Matriz Confusão –

```
cm = ConfusionMatrix(knn)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Métricas de desempenho:

```
accuracy_score(y_test, prev)
```

Regressão Logística

$$y = b_0 + b_1 * x$$ x = idade

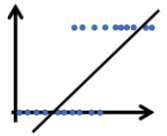
Equação da reta

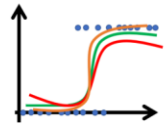
$$p = \frac{1}{1 + e^{-y}}$$

Função sigmoide

$$\log(\frac{p}{1-p}) = b_0 + b_1 * x$$

Transformação "logit"





Característica:
Utiliza função sigmoid para encontrar a melhor linha e encaixar os dados.

Hiperparâmetro:
max_iter (Número máximo de iterações)

Código:

```
log =
LogisticRegression(max_iter=100)
log.fit(X_train, y_train)
prev = log.predict(X_test)
```

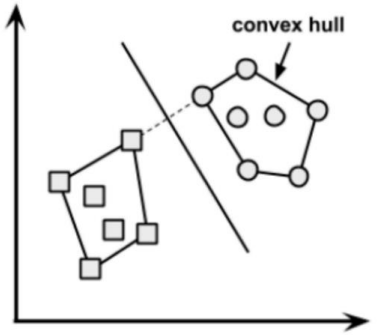
Métricas de erro:
Matriz Confusão –

```
cm = ConfusionMatrix(log)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Métricas de desempenho:

```
accuracy_score(y_test, prev)
```

SVM



Característica:
Cria um hiperplano com base nos vetores de suporte para dividir as classes.

Hiperparâmetro:
Kernel (Transformação dos dados)
C (Penalização dos erros)

Código:

```
svm = SVC(kernel='rbf',C=2)
svm.fit(X_train, y_train)
prev = svm.predict(X_test)
```

Métricas de erro:
Matriz Confusão –

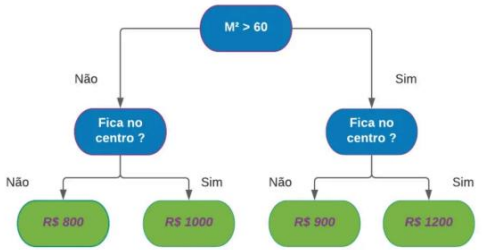
```
cm = ConfusionMatrix(svm)
cm.fit(X_train, y_train)
cm.score(X_test, y_test)
```

Métricas de desempenho:

```
accuracy_score(y_test, prev)
```

Regressão e Classificação

Árvore de Decisão



Característica:
Uma única variável exploratória.

Hiperparâmetro:
Nenhum.

Código:

```
tree= DecisionTreeRegressor()
tree.fit(X_train,y_train)
prev = tree.predict(X_test)
```

Random Forest

Característica:
Consulta muitas árvores e tira a média.

Hiperparâmetro:
n_estimators (número de consultas)

Código:

```
random=RandomForestRegressor(n_es
timators=100)

random.fit(X_random.predict(X_tes
t)
```

```
DecisionTreeRegressor()
DecisionTreeClassifier()


RandomForestRegressor()
RandomForestClassifier()
```

Encoder

Label Encoding

Color	Size	Price
Red	Small	10
Green	Medium	20
Blue	Large	30
Red	Large	25
Green	Small	15

Category	Encoded Value
Color: Red	0
Color: Green	1
Color: Blue	2

Category	Encoded Value
Size: Small	0
Size: Medium	1
Size: Large	2

Color_Encoded	Size_Encoded	Price
0	0	10
1	1	20
2	2	30
0	2	25
1	0	15

Característica:
Cada classe terá seu rótulo transformado para um valor numérico.

Código:

```
encoder = LabelEncoder()

X[:,1] =
encoder.fit_transform(X[:,1])
```

One-Hot Encoding

Color	Size	Price
Red	Small	10
Green	Medium	20
Blue	Large	30
Red	Large	25
Green	Small	15

Color_Red	Color_Green	Color_Blue	Size_Small	Size_Medium	Size_Large	Price
1	0	0	1	0	0	10
0	1	0	0	1	0	20
0	0	1	0	0	1	30
1	0	0	0	0	1	25
0	1	0	1	0	0	15

Característica:
Cada categoria é transformada em outro atributo: um valor binário informa a ocorrência.

Código:

```
onehot=
ColumnTransformer(transformers=[('
OneHot', OneHotEncoder(),
[c1,c2,c3])],
remainder='passthrough’)

X= onehot.fit_transform(X)
```

Label encoding	One-hot encoding
Há ordem (progr. Junior, Pleno, Sênior)	Não há ordem
Grande Número de categorias, não da pra usar One-hot encoding	Número de categorias é pequeno

Padronização - StandardScaler

O StandardScaler é como um "tradutor de escalas" que diz para o computador:

- Achar o Ponto Médio (Média);
- Centralizar no Zero;
- Ajustar a "Esticada" (Desvio Padrão);

Pessoa	Idade (Anos)	Salário (R$)
A	25	3.000
B	60	50.000
C	40	6.000

Pessoa	Idade (Transformada)	Salário (Transformado)
A	-1,2	-0,8
B	1,8	2,1
C	-0,1	-0,4

Código:

```
scaler = StandardScaler()
X_treinamento = scaler.fit_transform(X_treinamento)
X_teste = scaler.transform(X_teste)
```

Encoder

Label Encoding

Color	Size	Price
Red	Small	10
Green	Medium	20
Blue	Large	30
Red	Large	25
Green	Small	15

Category	Encoded Value
Color: Red	0
Color: Green	1
Color: Blue	2

Category	Encoded Value
Size: Small	0
Size: Medium	1
Size: Large	2

Color_Encoded	Size_Encoded	Price
0	0	10
1	1	20
2	2	30
0	2	25
1	0	15

Característica:
Cada classe terá seu rótulo transformado para um valor numérico.

Código:

```
encoder = LabelEncoder()

X[:,1] =
encoder.fit_transform(X[:,1])
```

One-Hot Encoding

Color	Size	Price
Red	Small	10
Green	Medium	20
Blue	Large	30
Red	Large	25
Green	Small	15

Color_Red	Color_Green	Color_Blue	Size_Small	Size_Medium	Size_Large	Price
1	0	0	1	0	0	10
0	1	0	0	1	0	20
0	0	1	0	0	1	30
1	0	0	0	0	1	25
0	1	0	1	0	0	15

Característica:
Cada categoria é transformada em outro atributo: um valor binário informa a ocorrência.

Código:

```
onehot=
ColumnTransformer(transformers=[('
OneHot', OneHotEncoder(),
[c1,c2,c3])],
remainder='passthrough’)

X= onehot.fit_transform(X)
```

Label encoding	One-hot encoding
Há ordem (progr. Junior, Pleno, Sênior)	Não há ordem
Grande Número de categorias, não da pra usar One-hot encoding	Número de categorias é pequeno

Otimização

PCA

Característica:
Técnica utilizada para reduzir o número de variáveis de um conjunto de dados, mantendo o máximo possível da informação essencial, cria atributos artificiais.

Hiperparâmetro:
Número de componentes

Código:

```
pca = PCA(n_components=5)
X_pca =
pca.fit_transform(X_train)
X_teste_pca =
pca.transform(X_test)
```

Select Atributes

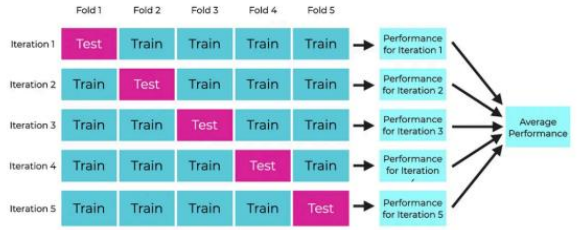
Característica:
Testa todas as combinações possíveis de atributos para encontrar o subconjunto que gera o melhor desempenho do modelo.

Hiperparâmetro:
K (Número de atributos)
F_classif (Para atributos numéricos.
Chi2 (Para atributos categóricos)

Código:

```
selecao = SelectKBest(f_classif,
k=7)
X = selecao.fit_transform(X, y)
```

Cross Validation



Característica:
Avalia o desempenho de um modelo de aprendizado de máquina de forma mais confiável evitando que ele pareça melhor (ou pior) do que realmente é.

Hiperparâmetro:
Número de divisões
Embaralhamento

Código:

```
kfold = KFold(n_splits=10,
shuffle=True, random_state=1)
svm = SVC(kernel = 'rbf', C =
1.5)
scores = cross_val_score(svm, X,
y, cv = kfold)
```

Grid Search

Característica:
Testa várias combinações de parâmetros de um modelo de machine learning e descobrie qual configuração gera o melhor desempenho.

Hiperparâmetro:
Algoritmo

Código:

```
parametros = {'C': [1.0, 1.5,
2.0],
 'kernel': ['rbf', 'linear',
'poly', 'sigmoid']}
grid_search =
GridSearchCV(estimator=SVC(),
param_grid=parametros)

grid_search.fit(X, y)

melhores_parametros =
grid_search.best_params_

melhor_resultado =
grid_search.best_score_

print(melhores_parametros)
print(melhor_resultado)
```

JOGO DOS 7 ERROS

OU MAIS

```
import pandas as pd
from sklearn.preprocessing import
LabelEncoder
from sklearn.model_selection import
train_testsplit
from sklearn.naive_bayes import GaussianNB

base = pd.read("census.csv")

X = base.iloc[:, 0:14].value
y = base.iloc[:, 0:14].values

lb = LabelEncoder()
X[:,1] = lb.fit_transform(X[:,1])

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.8)

model = GaussianNB()
model.fit(X, y_train)

pred = model.predict(X_test)
print(pred)
```

```
import pandas as pd
from sklearn.preprocessing import
LabelEncoder
from sklearn.model_selection import
train_testsplit
from sklearn.naive_bayes import GaussiosanNB

base = pd.read("census.csv")

X = base.iloc[:, 0:14].value
y = base.iloc[:, 0:14].values

lb = LabelEncoder()
X = lb.fit_transform(X[:,1])

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.8)

model = GaussianNB()
model.fit(X, y_train)

pred = model.predict(X_test)
print(pred)
```

```
import pandas as pd
from sklearn.preprocessing import
LabelEncoder
from sklearn.model_selection import
train_test_split
from sklearn.naive_bayes import GaussianNB

base = pd.read_csv("census.csv")

X = base.iloc[:, 0:14].value
y = base.iloc[:, 14].values

lb = LabelEncoder()
X[:,1] = lb.fit_transform(X[:,1])

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.8)

model = GaussianNB()
model.fit(X_train, y_train)

pred = model.predict(X_test)
print(pred)
```

```
import pandas as pd
from sklearn.datasets import
load_breast_cancer
from sklearn.model_selection import
train_test_split
from sklearn.linear_model import
LogisticRegressor
from sklearn.metrics import accuracy

dados = load_breast_cancer()
X = dados.data
y = dados.target

X_train, X_test, y_train, y_test =
train_test_split(X, y, test=0.2)

modelo = LogisticRegressor()
modelo.justify(X_train, y_train)

previsoes = modelo.preview(X_test)

print(acuracy_score(y_train, previsoes))
```

```
import pandas as pd
from sklearn.datasets import
load_breast_cancer
from sklearn.model_selection import
train_test_split
from sklearn.linear_model import
LogisticRegressor
from sklearn.metrics import accuracy

dados = load_breast_cancer()
X = dados.data
y = dados.targets

X_train, X_test, y_train, y_test =
train_test_split(X, y, test=0.2)

modelo = LogisticRegressor()
modelo.justify(X_train, y_train)

previsoes = modelo.preview(X_test)

print(acuracy_score(y_train, previsoes))
```

```
import pandas as pd
from sklearn.datasets import
load_breast_cancer
from sklearn.model_selection import
train_test_split
from sklearn.linear_model import
LogisticRegression
from sklearn.metrics import accuracy

dados = load_breast_cancer()
X = dados.data
y = dados.target

X_train, X_test, y_train, y_test =
train_test_split(X, y, test=0.2)

modelo = LogisticRegression()
modelo.fit(X_train, y_train)

previsoes = modelo.predict(X_test)

print(accuracy_score(y_test, previsoes))
```

```
from sklearn.neighbors import
KNeighborsClassifier
from sklearn.modelSelection import
train_test_split
from sklearn import datasets


iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test =
train_test_split(X, y, teste=0.3)

knn = KNeighborsClassifier(n_neighbors="5")
knn.fit_transform (X_test, y_train)

pred = knn.predict(X_train)
print(pred)

print(knn.score(X_test,y_test))
```

```
from sklearn.neighbors import
KNeighborsClassifier
from sklearn.modelSelection import
train_test_split
from sklearn import datasets


iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test =
train_test_split(X, y, teste=0.3)

knn = KNeighborsClassifier(n_neighbors="5")
knn.fit_transform (X_test, y_train)

pred = knn.predict(X_train)
print(pred)

print(knn.score(X_test,y_test))
```

```
from sklearn.neighbors import
KNeighborsClassifier
from sklearn.model_selection import
train_test_split
from sklearn import datasets


iris = datasets.load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3)

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

pred = knn.predict(X_test)
print(pred)

print(accuracy_score(y_test,pred))
```

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import
train_test_split
from sklearn.datasets import
load_breast_cancer

dados = load_breast_cancer()

X = dados.data.values
y = dados.targetes

pca = PCA(n_components=0)
X_pca = pca.fit_transformes(X)

X_train, X_test, y_train, y_test =
train_test_split(y, X, teste_size=20)

modelo = SVC(kernel="linear")
modelo.train(X_train, y_train)

pred = modelo.predict(X_treino)
print(pred)
```

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import
train_test_split
from sklearn.datasets import
load_breast_cancer

dados = load_breast_cancer()

X = dados.data.values
y = dados.targetes

pca = PCA(n_components=0)
X_pca = pca.fit_transformes(X)

X_train, X_test, y_train, y_test =
train_test_split(y, X, teste_size=20)

modelo = SVC(kernel="linear")
modelo.train(X_train, y_train)

pred = modelo.predict(X_treino)
print(pred)
```

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.model_selection import
train_test_split
from sklearn.datasets import
load_breast_cancer

dados = load_breast_cancer()

X = dados.data
y = dados.target

pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)

X_train, X_test, y_train, y_test =
train_test_split(X_pca, y, test_size=0.2)

modelo = SVC(kernel="linear")
modelo.fit(X_train, y_train)

pred = modelo.predict(X_treino)
print(pred)
```

```
import pandas as pd
import seaborn
import matplotlib as plt

df = pandas.load_csv("credit.csv")

corr = df.corrcoef()

plt.figure(figsize=(5,5))
seaborn.heat(corr, anotation=True,
cmap="blues")
plt.titulo("Matriz de Correlação")
plt.shaw()
```

```
import pandas as pd
import seaborn
import matplotlib as plt

df = pandas.load_csv("credit.csv")

corr = df.corrcoef()

plt.figure(figsize=(5,5))
seaborn.heat(corr, anotation=True,
cmap="blues")
plt.titulo("Matriz de Correlação")
plt.shaw()
```

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

df = pandas.read_csv("credit.csv")

corr = df.corr()

plt.figure(figsize=(5,5))
seaborn.heatmap(corr, annotation=True,
cmap="blues")
plt.title("Matriz de Correlação")
plt.show()
```