Jônatas Trabuco Belotti
RA: 230260
jonatas.t.belotti@hotmail.com
Repositório GitHub: https://github.com/jonatastbelotti/mo432-trab3

Leia

Leia o arquivo dados3.csv. O arquivo é um banco de dados conhecido sobre credito bancario na Australia, mas com alguns dos atributos categóricos originais. A descrição dos dados pode ser encontrada aqui. O atributo de saída é V15 (classes 1 e 2).

```
import pandas as pd

df = pd.read_csv("/content/dados3.csv")
df
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15
0	1	22.08	11.460	2	k	bb	1.585	0	0	0	1	g	100	1213	0
1	0	22.67	7.000	2	c	bb	0.165	0	0	0	0	g	160	1	0
2	0	29.58	1.750	1	k	bb	1.250	0	0	0	1	g	280	1	0
3	0	21.67	11.500	1	j	j	0.000	1	1	11	1	g	0	1	1
4	1	20.17	8.170	2	aa	bb	1.960	1	1	14	0	g	60	159	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	1	31.57	10.500	2	x	bb	6.500	1	0	0	0	g	0	1	1
686	1	20.67	0.415	2	c	bb	0.125	0	0	0	0	g	0	45	0
687	0	18.83	9.540	2	aa	bb	0.085	1	0	0	0	g	100	1	1
688	0	27.42	14.500	2	x	h	3.085	1	1	1	0	g	120	12	1
689	1	41.00	0.040	2	e	bb	0.040	0	1	1	0	s	560	1	1

690 rows × 15 columns

Preprocessamento

Faça a conversão dos atributos categóricos (V5, V6 e V12) para numéricos, usando one-hot-encoder/dummy variables.

```
for col in ["V5", "V6", "V12"]:
  dummies = pd.get_dummies(df[col])
  df = pd.concat([df, dummies], axis=1)
```

```
  df = df.drop([col], axis=1)

df
```

	V1	V2	V3	V4	V7	V8	V9	V10	V11	V13	V14	V15	aa	c	cc	d
0	1	22.08	11.460	2	1.585	0	0	0	1	100	1213	0	0	0	0	0
1	0	22.67	7.000	2	0.165	0	0	0	0	160	1	0	0	1	0	0
2	0	29.58	1.750	1	1.250	0	0	0	1	280	1	0	0	0	0	0
3	0	21.67	11.500	1	0.000	1	1	11	1	0	1	1	0	0	0	0
4	1	20.17	8.170	2	1.960	1	1	14	0	60	159	1	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
685	1	31.57	10.500	2	6.500	1	0	0	0	0	1	1	0	0	0	0
686	1	20.67	0.415	2	0.125	0	0	0	0	0	45	0	0	1	0	0
687	0	18.83	9.540	2	0.085	1	0	0	0	100	1	1	1	0	0	0
688	0	27.42	14.500	2	3.085	1	1	1	0	120	12	1	0	0	0	0
689	1	41.00	0.040	2	0.040	0	1	1	0	560	1	1	0	0	0	0

690 rows × 37 columns

Faça o centering e scaling para todos os atributos.

```
from sklearn.preprocessing import StandardScaler

# Separando os dados em entradas e saídas
saidas = df["V15"].to_numpy()
entradas = df.drop(["V15"], axis=1).to_numpy()

# Fazendo centering e scaling
scaler = StandardScaler()
scaler.fit(entradas)
entradas = scaler.transform(entradas)

entradas.shape, saidas.shape
```

```
((690, 36), (690,))
```

Execuções

Para cada um dos classificadores abaixo aplique Validação Cruzada do tipo 5-fold. Utilize AUC como medida de erro de todos os classificadores. A busca de hiperparametros será aleatória. O problema especificará um intervalo para os hiperparametros. Use uma distribuição uniforme para escolher valores neste intervalo. Se houver mais de um hiperparametro, escolha 10 combinações aleatórias deles.

Reporte o valor do AUC da melhor combinação de hiperparametros e o valor dos hiperparametros encontrados para cada classificador. Também reporte os valores do AUC para os classificadores do Scikit-learn com os valores padrão de hiperparâmetros.

Regressão Logística (sem regularização)

Não tem hiperparâmetro. Portanto, foi implementada apenas a validação cruzada.

```
from sklearn.model_selection import cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

resultados = cross_validate(LogisticRegression(penalty="none"), entradas, saidas, 
resultados = resultados['test_AUC']
print("AUC: %s" % str(resultados))
print("AUC médio: %s" % str(np.mean(resultados)))
```

```
AUC: [0.94783905 0.84713647 0.91271024 0.85059423 0.95140068]
AUC médio: 0.9019361326974954
```

Regressão Logística com Regularização L2

Realizar a busca dos seguintes hiperparâmetros:

- c: 10 números aleatórios entre 10^{-3} e 10^{3}, uniforme no expoente.

```
from sklearn.model_selection import RandomizedSearchCV, cross_validate
from scipy.stats import loguniform
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

# Definindo o intervalo dos parâmetros a serem utilizados
parametros = {
    'C': 10 ** np.random.uniform(-3, 3, 10)
}

# Definindo métrica de erro
erro = {"AUC": make_scorer(roc_auc_score, greater_is_better=True, needs_proba=True

# Buscando valor do hiperparâmetro
randomSCV = RandomizedSearchCV(LogisticRegression(penalty="l2", max_iter=200), para
randomSCV.fit(entradas, saidas)
C = randomSCV.best_params_['C']

# Treinando o classificador novamente com o valor encontrado do hiperparâmetro
resultados = cross_validate(LogisticRegression(penalty="l2", C=C, max_iter=200), en
resultados = resultados['test_AUC']
```

```
print("Resultado com C = %.6f" % C)
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))

# Treinando o classificador novamente com o valor padrão do hiperparâmetro
resultados = cross_validate(LogisticRegression(penalty="l2", max_iter=200), entrada
resultados = resultados['test_AUC']
print("\nResultado com C padrão")
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))
```

```
Resultado com C = 0.008634
 AUC: [0.94315521 0.88439429 0.95230999 0.90152801 0.94864177]
 AUC médio: 0.926005852812889

Resultado com C padrão
 AUC: [0.94911646 0.86140089 0.94592293 0.88900679 0.94927844]
 AUC médio: 0.9189451020464965
```

LDA

Não tem hiperparâmetro.

```
from sklearn.model_selection import cross_validate
from scipy.stats import loguniform
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

resultados = cross_validate(LinearDiscriminantAnalysis(), entradas, saidas, cv=5, s
resultados = resultados['test_AUC']
print("AUC: %s" % str(resultados))
print("AUC médio: %s" % str(np.mean(resultados)))
```

```
AUC: [0.94038748 0.87438791 0.95827124 0.89431239 0.94418506]
AUC médio: 0.9223088157632675
```

QDA

Não tem hiperparâmetros.

```
from sklearn.model_selection import cross_validate
from scipy.stats import loguniform
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

resultados = cross_validate(QuadraticDiscriminantAnalysis(), entradas, saidas, cv=5
resultados = resultados['test_AUC']
print("AUC: %s" % str(resultados))
print("AUC médio: %s" % str(np.mean(resultados)))
```

```
AUC: [0.91015542 0.74430488 0.84681712 0.77949915 0.85356537]
AUC médio: 0.8268683854485019
/usr/local/lib/python3.6/dist-packages/sklearn/discriminant_analysis.py:691:
  warnings.warn("Variables are collinear")
/usr/local/lib/python3.6/dist-packages/sklearn/discriminant_analysis.py:691:
  warnings.warn("Variables are collinear")
/usr/local/lib/python3.6/dist-packages/sklearn/discriminant_analysis.py:691:
  warnings.warn("Variables are collinear")
/usr/local/lib/python3.6/dist-packages/sklearn/discriminant_analysis.py:691:
  warnings.warn("Variables are collinear")
/usr/local/lib/python3.6/dist-packages/sklearn/discriminant_analysis.py:691:
  warnings.warn("Variables are collinear")
```

SVM Linear

Selecione 10 valores aleatórios ente:

- C: entre 2^{-5} e 2^{15}.

```
from sklearn.model_selection import RandomizedSearchCV, cross_validate
from scipy.stats import loguniform
from sklearn.svm import LinearSVC
from sklearn.metrics import make_scorer, roc_auc_score
from sklearn.exceptions import ConvergenceWarning
from warnings import simplefilter
import numpy as np

# Ignorando warnings no treinamento dos classificadores
simplefilter("ignore", category=ConvergenceWarning)

# Definindo o intervalo dos parâmetros a serem utilizados
parametros = {
    'C': 2 ** np.random.uniform(-5, 15, 10)
}

# Definindo métrica de erro
erro = {"AUC": make_scorer(roc_auc_score, greater_is_better=True)}

# Buscando valor do hiperparâmetro
randomSCV = RandomizedSearchCV(LinearSVC(), parametros, scoring=erro, refit="AUC",
randomSCV.fit(entradas, saidas)
C = randomSCV.best_params_['C']

# Treinando o classificador novamente com o valor encontrado do hiperparâmetro
resultados = cross_validate(LinearSVC(C=C), entradas, saidas, cv=5, scoring=erro)
resultados = resultados['test_AUC']
print("Resultado com C = %.6f" % C)
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))

# Treinando o classificador novamente com o valor padrão do hiperparâmetro
resultados = cross_validate(LinearSVC(), entradas, saidas, cv=5, scoring=erro)
resultados = resultados['test_AUC']
```

```
print("\nResultado com C padrão")
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))
```

```
Resultado com C = 0.569580
 AUC: [0.89716841 0.79465616 0.88929104 0.82512733 0.88582343]
 AUC médio: 0.8584132739425122

Resultado com C padrão
 AUC: [0.89716841 0.79465616 0.88109432 0.82512733 0.88582343]
 AUC médio: 0.8567739296802172
```

SVM com kernel RBF

Selecione 10 duplas aleatórias ente:

- C: entre 2^{-5} e 2^{15}, uniforme no expoente
- gamma: entre 2^{-9} e 2^{3}, uniforme no expoente

```
from sklearn.model_selection import RandomizedSearchCV, cross_validate
from scipy.stats import loguniform
from sklearn.svm import SVC
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

# Definindo o intervalo dos parâmetros a serem utilizados
parametros = {
    'C': 2 ** np.random.uniform(-5, 15, 10),
    'gamma': 2 ** np.random.uniform(-9, 3, 10)
}

# Definindo métrica de erro
erro = {"AUC": make_scorer(roc_auc_score, greater_is_better=True)}

# Buscando valor do hiperparâmetro
randomSCV = RandomizedSearchCV(SVC(), parametros, scoring=erro, refit="AUC", cv=5,
randomSCV.fit(entradas, saidas)
C = randomSCV.best_params_['C']
gamma = randomSCV.best_params_['gamma']

# Treinando o classificador novamente com o valor encontrado do hiperparâmetro
resultados = cross_validate(SVC(C=C, gamma=gamma), entradas, saidas, cv=5, scoring=
resultados = resultados['test_AUC']
print("Resultado com C = %.6f e gamma = %.6f" % (C, gamma))
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))

# Treinando o classificador novamente com o valor padrão do hiperparâmetro
resultados = cross_validate(SVC(), entradas, saidas, cv=5, scoring=erro)
resultados = resultados['test_AUC']
print("\nResultado com C e gamma padrão")
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))
```

```
print("AUC médio: %s" % str(np.mean(resultados)))
```

```
Resultado com C = 0.256584 e gamma = 0.026496
 AUC: [0.88247818 0.75505642 0.87598467 0.83467742 0.8696944 ]
 AUC médio: 0.8435782168511997

Resultado com C e gamma padrão
 AUC: [0.88418139 0.75675963 0.88758782 0.80390492 0.88433786]
 AUC médio: 0.843354326516257
```

Naive Bayes

Não tem busca de hiperparâmetros.

```
from sklearn.model_selection import cross_validate
from scipy.stats import loguniform
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

resultados = cross_validate(GaussianNB(), entradas, saidas, cv=5, scoring={"AUC": r
resultados = resultados['test_AUC']
print("AUC: %s" % str(resultados))
print("AUC médio: %s" % str(np.mean(resultados)))
```

```
AUC: [0.90674899 0.77517564 0.86235895 0.81833616 0.87754669]
AUC médio: 0.8480332875118425
```

KNN

Faça a busca dos seguintes hiperparâmetros:

- K: 10 números aleatórios impares entre 1 e 301.

```
from sklearn.model_selection import RandomizedSearchCV, cross_validate
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

# Definindo o intervalo dos parâmetros a serem utilizados
parametros = {
    'n_neighbors': np.arange(1, 302, 2)
}

# Definindo métrica de erro
erro = {"AUC": make_scorer(roc_auc_score, greater_is_better=True)}

# Buscando valor do hiperparâmetro
randomSCV = RandomizedSearchCV(KNeighborsClassifier(), parametros, scoring=erro, r
randomSCV.fit(entradas, saidas)
K = randomSCV.best_params_['n_neighbors']
```

```
# Treinando o classificador novamente com o valor encontrado do hiperparâmetro
resultados = cross_validate(KNeighborsClassifier(n_neighbors=K), entradas, saidas,
resultados = resultados['test_AUC']
print("Resultado com K = %.0f" % (K))
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))

# Treinando o classificador novamente com o valor padrão do hiperparâmetro
resultados = cross_validate(KNeighborsClassifier(), entradas, saidas, cv=5, scorin
resultados = resultados['test_AUC']
print("\nResultado com K padrão")
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))
```

```
Resultado com K = 115
 AUC: [0.82989142 0.79880775 0.82478178 0.78162139 0.77207131]
 AUC médio: 0.8014347289549771

Resultado com K padrão
 AUC: [0.82031084 0.75505642 0.83638493 0.76209677 0.78395586]
 AUC médio: 0.7915609627645865
```

MLP

Busca dos seguintes hiperparêmetros:

- Neurônios na camada do meio: de 5 a 20, de três em três.

```
from sklearn.model_selection import GridSearchCV, cross_validate
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

# Definindo o intervalo dos parâmetros a serem utilizados
parametros = {
    'hidden_layer_sizes': np.arange(5, 20+1, 5)
}

# Definindo métrica de erro
erro = {"AUC": make_scorer(roc_auc_score, greater_is_better=True)}

# Buscando valor do hiperparâmetro
grid = GridSearchCV(estimator=MLPClassifier(), param_grid=parametros, scoring=erro
grid.fit(entradas, saidas)
hidden_layer_sizes = grid.best_params_['hidden_layer_sizes']

# Treinando o classificador novamente com o valor encontrado do hiperparâmetro
resultados = cross_validate(MLPClassifier(hidden_layer_sizes=hidden_layer_sizes), 
resultados = resultados['test_AUC']
print("Resultado com %.0f neurônios na canada escondida" % (hidden_layer_sizes))
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))

# Treinando o classificador novamente com o valor padrão do hiperparâmetro
```

```
# Treinando o classificador novamente com o valor padrão do hiperparâmetro
resultados = cross_validate(MLPClassifier(), entradas, saidas, scoring=erro, cv=5,
resultados = resultados['test_AUC']
print("\nResultado com valores padrões")
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))
```

```
Resultado com 20 neurônios na canada escondida
 AUC: [0.88726847 0.79774324 0.87768789 0.84486418 0.84252971]
 AUC médio: 0.8500186966864304

Resultado com valores padrões
 AUC: [0.86438152 0.77144986 0.88247818 0.83022071 0.85865874]
 AUC médio: 0.8414378031998897
```

Arvore de decisão

Use prunning e faça a busca dos seguintes hiperparâmetros:

- ccp_alpha: 10 números aleatórios entre 0.0 e 0.04

```
from sklearn.model_selection import GridSearchCV, cross_validate
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

# Definindo o intervalo dos parâmetros a serem utilizados
parametros = {
    'ccp_alpha': np.random.uniform(0, 0.04, 10)
}

# Definindo métrica de erro
erro = {"AUC": make_scorer(roc_auc_score, greater_is_better=True)}

# Buscando valor do hiperparâmetro
grid = GridSearchCV(estimator=DecisionTreeClassifier(), param_grid=parametros, sco
grid.fit(entradas, saidas)
ccp_alpha = grid.best_params_['ccp_alpha']

# Treinando o classificador novamente com o valor encontrado do hiperparâmetro
resultados = cross_validate(DecisionTreeClassifier(ccp_alpha=ccp_alpha), entradas,
resultados = resultados['test_AUC']
print("Resultado com ccp_alpha = %.6f" % (ccp_alpha))
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))

# Treinando o classificador novamente com o valor padrão do hiperparâmetro
resultados = cross_validate(DecisionTreeClassifier(), entradas, saidas, scoring=er
resultados = resultados['test_AUC']
print("\nResultado com valores padrões")
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))
```

```
Resultado com ccp_alpha = 0.012680
 AUC: [0.88758782 0.82744305 0.89578454 0.81345501 0.88518676]
 AUC médio: 0.8618914359597374

Resultado com valores padrões
 AUC: [0.80593996 0.76495636 0.80253353 0.79796265 0.81472835]
 AUC médio: 0.7972241701074954
```

Random Forest

Use todas as combinações dos valores abaixo:

- n_estimators: use os valores 10, 100 e 1000.
- max_features: use os valores 5, 8 e 10.

```
from sklearn.model_selection import GridSearchCV, cross_validate
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

# Definindo o intervalo dos parâmetros a serem utilizados
parametros = {
    'n_estimators': [10, 100, 1000],
    'max_features': [5, 8, 10]
}

# Definindo métrica de erro
erro = {"AUC": make_scorer(roc_auc_score, greater_is_better=True)}

# Buscando valor do hiperparâmetro
grid = GridSearchCV(estimator=RandomForestClassifier(), param_grid=parametros, sco
grid.fit(entradas, saidas)
n_estimators = grid.best_params_['n_estimators']
max_features = grid.best_params_['max_features']

# Treinando o classificador novamente com o valor encontrado do hiperparâmetro
resultados = cross_validate(RandomForestClassifier(n_estimators=n_estimators, max_
resultados = resultados['test_AUC']
print("Resultado com n_estimators = %d e max_features = %d" % (n_estimators, max_fe
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))

# Treinando o classificador novamente com o valor padrão do hiperparâmetro
resultados = cross_validate(RandomForestClassifier(), entradas, saidas, scoring=er
resultados = resultados['test_AUC']
print("\nResultado com valores padrões")
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))
```

```
Resultado com n_estimators = 100 e max_features = 10
 AUC: [0.90536513 0.84011071 0.92005535 0.80241935 0.86523769]
```

GBM

Selecione 10 trinca aleatórias ente:

- n_estimators: de 5 a 100.
- learning_rate: de 0.01 a 0.3.
- max_depth: 2 ou 3.

```
from sklearn.model_selection import RandomizedSearchCV, cross_validate
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import make_scorer, roc_auc_score
import numpy as np

# Definindo o intervalo dos parâmetros a serem utilizados
parametros = {
    'n_estimators': np.random.uniform(5, 100, 10).astype("int32"),
    'learning_rate': np.random.uniform(0.01, 0.3, 10),
    'max_depth': [2, 3]
}

# Definindo métrica de erro
erro = {"AUC": make_scorer(roc_auc_score, greater_is_better=True)}

# Buscando valor do hiperparâmetro
randomSCV = RandomizedSearchCV(GradientBoostingClassifier(), parametros, scoring=e
randomSCV.fit(entradas, saidas)
n_estimators = randomSCV.best_params_['n_estimators']
learning_rate = randomSCV.best_params_['learning_rate']
max_depth = randomSCV.best_params_['max_depth']

# Treinando o classificador novamente com o valor encontrado do hiperparâmetro
resultados = cross_validate(GradientBoostingClassifier(n_estimators=n_estimators,
resultados = resultados['test_AUC']
print("Resultado com n_estimators = %d, learning_rate = %.6f e max_depth = %d" % (
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))

# Treinando o classificador novamente com o valor padrão do hiperparâmetro
resultados = cross_validate(GradientBoostingClassifier(), entradas, saidas, cv=5, 
resultados = resultados['test_AUC']
print("\nResultado com valores padrões")
print(" AUC: %s" % str(resultados))
print(" AUC médio: %s" % str(np.mean(resultados)))
```

```
Resultado com n_estimators = 92, learning_rate = 0.091268 e max_depth = 3
 AUC: [0.89408133 0.81552055 0.88418139 0.8089983  0.90110357]
 AUC médio: 0.8607770266973139

Resultado com valores padrões
 AUC: [0.88588461 0.7991271  0.87119438 0.81557725 0.89452462]
 AUC médio: 0.8532615913130261
```