Scikit Learn

Denis Henrique dos Santos
Fabrício Steinle Amoroso
Vinícius de Paula Pilan
Vinicius Machado Coutinho

Introdução

Sobre o Scikit Learn

SciKit-learn (ou sklearn) é um biblioteca open source de Machine Learning que suporta aprendizado supervisionado e não supervisionado.

O sklearn contém ferramentas para fitting de modelos, pré-processamento de dados, seleção e avaliação de modelos e muitas outras utilidades

$ pip install -U scikit-learn

O que ele pode fazer?

Datasets do sklearn

`sklearn.datasets.`**`make_classification`**(***n_samples=100***, ***n_features=20***, *******, ***n_informative=2***, ***n_redundant=2***, ***n_repeated=0***, ***n_classes=2***, *n_clusters_per_class=2*, *weights=None*, *flip_y=0.01*, *class_sep=1.0*, *hypercube=True*, *shift=0.0*, *scale=1.0*, *shuffle=True*, ***random_state=None***)

O sklearn possui ferramentas de carregar datasets já prontos, como o iris dataset, ou gerar datasets artificiais.

```
from sklearn.datasets import make_classification
X, y = make_classification(
    n_samples=6, n_features=5, n_redundant=1, n_informative=3,
     n_repeated=1, random_state=7, n_classes=2
)
```

```
X =
[[ 0.14741644  0.69067053 -1.09762854  1.1773073   0.14741644]
 [-2.83575085  0.01654092  0.83799014  0.34119595 -2.83575085]
 [-2.34106011 -0.10235294  0.85610283  0.09239558 -2.34106011]
 [ 0.83385386  1.09319984 -1.23435372  1.18433946  0.83385386]
 [-0.48278352  0.04382366  0.31268489 -0.07886236 -0.48278352]
 [-0.62882041 -1.14558206 -1.13981832  0.84001321 -0.62882041]]
Média = [-0.8845241   0.09938332 -0.24417045  0.59273153 -0.8845241 ]
Desvio Padrão = [1.30213203 0.69974732 0.93119989 0.50291178 1.30213203]
```

Pré-processamento

O scikit-learn possui ferramentas para auxiliar no pré-processamento de dados, processo importante para alcançar resultados de qualidade.

- StandardScaler
- MinMaxScaler
- OrdinalEncoder
- OneHotEncoder
- Binarizer
- SimpleImputer

StandardScaler

`sklearn.preprocessing.`**`StandardScaler`**(*, *copy=True*, *with_mean=True*, *with_std=True*)

O StandardScaler é um método de padronização que procura transformar os valores do dataset com o intuito de atingir os valores 0 e 1 para média e desvio padrão respectivamente.

```
from sklearn.preprocessing import StandardScaler
StdScaler = StandardScaler().fit(X)
X_StandardScaler = StdScaler.transform(X)
```

```
X =
[[ 0.14741644  0.69067053 -1.09762854  1.1773073   0.14741644]
 [-2.83575085  0.01654092  0.83799014  0.34119595 -2.83575085]
 [-2.34106011 -0.10235294  0.85610283  0.09239558 -2.34106011]
 [ 0.83385386  1.09319984 -1.23435372  1.18433946  0.83385386]
 [-0.48278352  0.04382366  0.31268489 -0.07886236 -0.48278352]
 [-0.62882041 -1.14558206 -1.13981832  0.84001321 -0.62882041]]
Média = [-0.8845241   0.09938332 -0.24417045  0.59273153 -0.8845241 ]
Desvio Padrão = [1.30213203 0.69974732 0.93119989 0.50291178 1.30213203]
```

```
X_scaled =
[[ 0.79250069  0.84500102 -0.91651437  1.16238235  0.79250069]
 [-1.4984861  -0.11838902  1.16211417 -0.50015844 -1.4984861 ]
 [-1.11857782 -0.28829873  1.18156509 -0.99487815 -1.11857782]
 [ 1.31966492  1.42025054 -1.06334126  1.17636523  1.31966492]
 [ 0.30852522 -0.07939962  0.59799765 -1.33541091  0.30852522]
 [ 0.19637309 -1.7791642  -0.96182127  0.49169993  0.19637309]]
Média = [-5.55111512e-17  0.00000000e+00  7.40148683e-17  1.38777878e-16
 -5.55111512e-17]
Desvio Padrão = [1. 1. 1. 1. 1.]
```

MinMaxScaler

`sklearn.preprocessing.`**`MinMaxScaler`**(***feature_range=(0, 1)***, *, *copy=True, clip=False*)

O MinMaxScaler escala os valores do dataset para um determinado intervalo escolhido.

```
from sklearn.preprocessing import MinMaxScaler
MinMax = MinMaxScaler(feature_range=(0,2)).fit(X)
X_MinMaxScaler = MinMax.transform(X)
```

```
X =
[[ 0.14741644  0.69067053 -1.09762854  1.1773073   0.14741644]
 [-2.83575085  0.01654092  0.83799014  0.34119595 -2.83575085]
 [-2.34106011 -0.10235294  0.85610283  0.09239558 -2.34106011]
 [ 0.83385386  1.09319984 -1.23435372  1.18433946  0.83385386]
 [-0.48278352  0.04382366  0.31268489 -0.07886236 -0.48278352]
 [-0.62882041 -1.14558206 -1.13981832  0.84001321 -0.62882041]]
Média = [-0.8845241   0.09938332 -0.24417045  0.59273153 -0.8845241 ]
Desvio Padrão = [1.30213203 0.69974732 0.93119989 0.50291178 1.30213203]
```

```
X_scaled =
[[1.62587937 1.64040328 0.13080892 1.98886614 1.62587937]
 [0.         1.03817436 1.98267106 0.66506919 0.        ]
 [0.26961528 0.93196137 2.         0.27114897 0.26961528]
 [2.         2.         0.         2.         2.        ]
 [1.28240915 1.0625472  1.4800964  0.         1.28240915]
 [1.20281644 0.         0.09044474 1.45483573 1.20281644]]
Média = [1.06345337 1.11218103 0.94733685 1.06332    1.06345337]
Desvio Padrão = [0.70968517 0.62511433 0.89090576 0.7962493  0.70968517]
```

OrdinalEncoder

`sklearn.preprocessing.`**`OrdinalEncoder`**(*, *categories='auto'*, *dtype=<class 'numpy.float64'>*, *handle_unknown='error'*, *unknown_value=None*, *encoded_missing_value=nan*)

O OrdinalEncoder é uma ferramenta de codificação que transforma categorias discretas em valores numéricos.

```
from sklearn.preprocessing import OrdinalEncoder
X = [['male','red','large'], ['female','blue','large'],
     ['male','brown','small'],['female','red','medium']]
OrdEncoder = OrdinalEncoder().fit(X)
X_OrdinalEncoder = OrdEncoder.transform(X)
```

```
[[1. 2. 0.]
 [0. 0. 0.]
 [1. 1. 2.]
 [0. 2. 1.]]
Categorias:
['female' 'male']
['blue' 'brown' 'red']
['large' 'medium' 'small']
```

OneHotEncoder

`sklearn.preprocessing.`**`OneHotEncoder`**(*, *categories='auto'*, *drop=None*, *sparse=True*, *dtype=<class 'numpy.float64'>*, *handle_unknown='error'*, *min_frequency=None*, *max_categories=None*)

O OneHotEncoder codifica semelhante ao OrdinalEncoder porém transforma o dataset em valores booleanos descrevendo se o dado pertence a tal categoria (one-hot-encoding)

```
from sklearn.preprocessing import OneHotEncoder
OneEncoder = OneHotEncoder().fit(X)
X_OneHotEncoder = OneEncoder.transform(X).toarray()
```

```
[[0. 1. 0. 0. 1. 1. 0. 0.]
 [1. 0. 1. 0. 0. 1. 0. 0.]
 [0. 1. 0. 1. 0. 0. 0. 1.]
 [1. 0. 0. 0. 1. 0. 1. 0.]]
Categorias:
['female' 'male']
['blue' 'brown' 'red']
['large' 'medium' 'small']
```

Binarizer

`sklearn.preprocessing.`**`Binarizer`**(*, ***`threshold=0.0`***, *copy=True*)

O Binarizer é uma ferramenta de discretização que faz uma binarização nos dados dividindo os valores em 0 e 1 dependendo se é maior ou não que um determinado valor.

```
from sklearn.preprocessing import Binarizer
X = [[7. , 2. , 9.],
     [10., 6.9, 0.],
     [5. , 4. , 6.],
     [5.0, 5.1, 4.9]]
X_Binarizer = Binarizer(threshold=5.).fit_transform(X)
```

```
X_scaled =
[[1. 0. 1.]
 [1. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]
```

SimpleImputer

`sklearn.impute.`**`SimpleImputer`**(*, ***missing_values=nan***, ***strategy='mean'***, *fill_value=None*, *verbose='deprecated'*, *copy=True*, *add_indicator=False*)

O SimpleImputer é uma ferramenta de imputação que substitui valores inválidos do dataset com valores obtidos a partir da estratégia escolhida.

```
from sklearn.impute import SimpleImputer
X = [[5. , 4. , 6. ],
     [-1., 6. , 4. ],
     [3. , 10., -1.],
     [4. , -1., 3. ]]
imp = SimpleImputer(missing_values=-1.,strategy='mean')
X_SimpleImputer = imp.fit_transform(X)
```

```
X_scaled =
[[ 5.          4.          6.        ]
 [ 4.          6.          4.        ]
 [ 3.         10.          4.33333333]
 [ 4.          6.66666667  3.        ]]
```

train_test_split

`sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)`

O train_test_split divide o dataset em grupos de treinamento e teste.

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.3)
```

```
Subconjunto de treinamento:
[[-0.48278352  0.04382366  0.31268489 -0.07886236 -0.48278352]
 [-2.83575085  0.01654092  0.83799014  0.34119595 -2.83575085]
 [ 0.14741644  0.69067053 -1.09762854  1.1773073   0.14741644]
 [-0.62882041 -1.14558206 -1.13981832  0.84001321 -0.62882041]] [1 0 1 0]
Subconjunto de teste:
[[-2.34106011 -0.10235294  0.85610283  0.09239558 -2.34106011]
 [ 0.83385386  1.09319984 -1.23435372  1.18433946  0.83385386]] [0 1]
```

Modelos de Machine Learning - Regressão

Modelo de Regressão com Florestas Aleatórias:

`sklearn.ensemble.`**`RandomForestRegressor`**(***n_estimators=100***, ***, ***criterion='squared_error'***, ***max_depth=None***, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=1.0*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *bootstrap=True*, *oob_score=False*, ***n_jobs=None***, ***random_state=None***, *verbose=0*, *warm_start=False*, *ccp_alpha=0.0*, *max_samples=None*)

Modelos de Machine Learning - Regressão

Exemplo - Modelo de Regressão com Florestas Aleatórias:

```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=300,
                              criterion='poisson',
                              max_depth=None,
                              random_state=10,
                              n_jobs=-1)

model.fit(X, y)

y_hat = model.predict(X)
```

Modelos de Machine Learning - Classificação

SVC - Classificador de SVM (máquina de vetores de suporte):

`sklearn.svm.`**SVC**(*, ***C=1.0***, ***kernel='rbf'***, *degree=3*, *gamma='scale'*, *coef0=0.0*, *shrinking=True*, ***probability=False***, ***tol=0.001***, *cache_size=200*, *class_weight=None*, *verbose=False*, *max_iter=-1*, *decision_function_shape='ovr'*, *break_ties=False*, ***random_state=None***)

Modelos de Machine Learning - Classificação

Exemplo - Classificador SVC:

```
from sklearn.svm import SVC

model = SVC(C=0.5,
            kernel='sigmoid',
            probability=True,
            tol=0.001,
            random_state=10)

model.fit(X, y)

y_hat = model.predict(X)
```

Outras implementações

Regressão:

Regressão Linear, Regressão de Crista (Ridge), ...

Classificação:

Regressão Logística, K vizinhos próximos (KNN), Árvore de Decisão, ...

Agrupamento:

K Means, ...

Métricas e Avaliação

- Precisão

`sklearn.metrics.`**`precision_score`**(*y_true*, *y_pred*, *, *labels=None*, *pos_label=1*, *average='binary'*, *sample_weight=None*, *zero_division='warn'*)

- Acurácia

`sklearn.metrics.`**`accuracy_score`**(*y_true*, *y_pred*, *, *normalize=True*, *sample_weight=None*)

- Recall

`sklearn.metrics.`**`recall_score`**(*y_true*, *y_pred*, *, *labels=None*, *pos_label=1*, *average='binary'*, *sample_weight=None*, *zero_division='warn'*)

```
= 2 * (precision * recall) /
(precision + recall)
```

- F-score

`sklearn.metrics.`**`f1_score`**(*y_true*, *y_pred*, *, *labels=None*, *pos_label=1*, *average='binary'*, *sample_weight=None*, *zero_division='warn'*)

- Matriz de Confusão

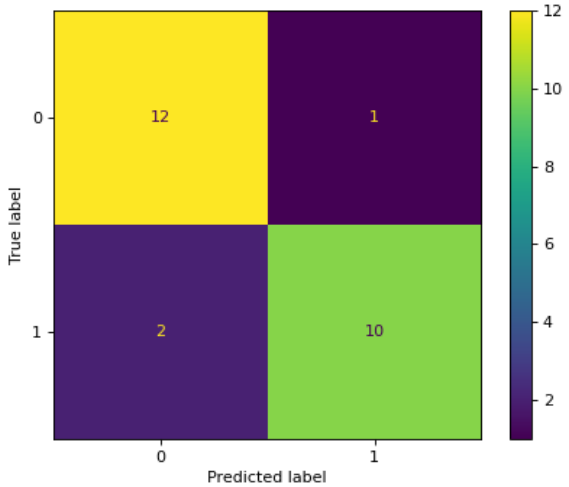
`sklearn.metrics.`**`confusion_matrix`**(*y_true*, *y_pred*, *, *labels=None*, *sample_weight=None*, *normalize=None*)

Métricas e Avaliação

Matriz de Confusão

```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
array([[2, 0, 0],
       [0, 0, 1],
       [1, 0, 2]])
```

```
>>> plot_confusion_matrix(clf, X_test, y_test)
>>> plt.show()
```



Classification metrics

See the Classification metrics section of the user guide for further details.

metrics.accuracy_score(y_true, y_pred, *[, ...])	Accuracy classification score.
metrics.auc(x, y)	Compute Area Under the Curve (AUC) using the trapezoidal rule.
metrics.average_precision_score(y_true, ...)	Compute average precision (AP) from prediction scores.
metrics.balanced_accuracy_score(y_true, ...)	Compute the balanced accuracy.
metrics.brier_score_loss(y_true, y_prob, *)	Compute the Brier score loss.
metrics.classification_report(y_true, y_pred, *)	Build a text report showing the main classification metrics.
metrics.cohen_kappa_score(y1, y2, *[, ...])	Compute Cohen's kappa: a statistic that measures inter-annotator agreement.
metrics.confusion_matrix(y_true, y_pred, *)	Compute confusion matrix to evaluate the accuracy of a classification.
metrics.dcg_score(y_true, y_score, *[, k, ...])	Compute Discounted Cumulative Gain.
metrics.det_curve(y_true, y_score[, ...])	Compute error rates for different probability thresholds.
metrics.f1_score(y_true, y_pred, *[, ...])	Compute the F1 score, also known as balanced F-score or F-measure.
metrics.fbeta_score(y_true, y_pred, *, beta)	Compute the F-beta score.
metrics.hamming_loss(y_true, y_pred, *[, ...])	Compute the average Hamming loss.
metrics.hinge_loss(y_true, pred_decision, *)	Average hinge loss (non-regularized).
metrics.jaccard_score(y_true, y_pred, *[, ...])	Jaccard similarity coefficient score.
metrics.log_loss(y_true, y_pred, *[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
metrics.matthews_corrcoef(y_true, y_pred, *)	Compute the Matthews correlation coefficient (MCC).
metrics.multilabel_confusion_matrix(y_true, ...)	Compute a confusion matrix for each class or sample.
metrics.ndcg_score(y_true, y_score, *[, k, ...])	Compute Normalized Discounted Cumulative Gain.
metrics.precision_recall_curve(y_true, ...)	Compute precision-recall pairs for different probability thresholds.
metrics.precision_recall_fscore_support(...)	Compute precision, recall, F-measure and support for each class.
metrics.precision_score(y_true, y_pred, *[, ...])	Compute the precision.
metrics.recall_score(y_true, y_pred, *[, ...])	Compute the recall.
metrics.roc_auc_score(y_true, y_score, *[, ...])	Compute Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
metrics.roc_curve(y_true, y_score, *[, ...])	Compute Receiver operating characteristic (ROC).
metrics.top_k_accuracy_score(y_true, y_score, *)	Top-k Accuracy classification score.
metrics.zero_one_loss(y_true, y_pred, *[, ...])	Zero-one classification loss.

Regression metrics

See the Regression metrics section of the user guide for further details.

`metrics.explained_variance_score`(y_true, ...)	Explained variance regression score function.
`metrics.max_error`(y_true, y_pred)	The max_error metric calculates the maximum residual error.
`metrics.mean_absolute_error`(y_true, y_pred, *)	Mean absolute error regression loss.
`metrics.mean_squared_error`(y_true, y_pred, *)	Mean squared error regression loss.
`metrics.mean_squared_log_error`(y_true, y_pred, *)	Mean squared logarithmic error regression loss.
`metrics.median_absolute_error`(y_true, y_pred, *)	Median absolute error regression loss.
`metrics.mean_absolute_percentage_error`(...)	Mean absolute percentage error (MAPE) regression loss.
`metrics.r2_score`(y_true, y_pred, *[, ...])	R^2 (coefficient of determination) regression score function.
`metrics.mean_poisson_deviance`(y_true, y_pred, *)	Mean Poisson deviance regression loss.
`metrics.mean_gamma_deviance`(y_true, y_pred, *)	Mean Gamma deviance regression loss.
`metrics.mean_tweedie_deviance`(y_true, y_pred, *)	Mean Tweedie deviance regression loss.
`metrics.d2_tweedie_score`(y_true, y_pred, *)	D^2 regression score function, fraction of Tweedie deviance explained.
`metrics.mean_pinball_loss`(y_true, y_pred, *)	Pinball loss for quantile regression.
`metrics.d2_pinball_score`(y_true, y_pred, *)	D^2 regression score function, fraction of pinball loss explained.
`metrics.d2_absolute_error_score`(y_true, ...)	D^2 regression score function, fraction of absolute error explained.

Plotting

See the Visualizations section of the user guide for further details.

metrics.plot_confusion_matrix(estimator, X, ...)	DEPRECATED: Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2.
metrics.plot_det_curve(estimator, X, y, *[, ...])	DEPRECATED: Function plot_det_curve is deprecated in 1.0 and will be removed in 1.2.
metrics.plot_precision_recall_curve(...[, ...])	DEPRECATED: Function `plot_precision_recall_curve` is deprecated in 1.0 and will be removed in 1.2.
metrics.plot_roc_curve(estimator, X, y, *[, ...])	DEPRECATED: Function `plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2.

metrics.ConfusionMatrixDisplay(...[, ...])	Confusion Matrix visualization.
metrics.DetCurveDisplay(*, fpr, fnr[, ...])	DET curve visualization.
metrics.PrecisionRecallDisplay(precision, ...)	Precision Recall visualization.
metrics.RocCurveDisplay(*, fpr, tpr[, ...])	ROC Curve visualization.
calibration.CalibrationDisplay(prob_true, ...)	Calibration curve (also known as reliability diagram) visualization.

Exemplo de uso da ferramenta

Notebook com a criação de um classificador usando sklearn:

https://colab.research.google.com/drive/1gHqxLe5b90_gsmS0518ztH769TYbMWBg?usp=sharing

Referências

https://scikit-learn.org/stable/