Kostenloses eBook

LERNEN scikit-learn

Free unaffiliated eBook created from
Stack Overflow contributors.

#scikit-learn

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: scikit-learn

It is an unofficial and free scikit-learn ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official scikit-learn.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Scikit-Learn

Bemerkungen

`scikit-learn` ist eine allgemeine Open-Source-Bibliothek für die Datenanalyse in Python. Es basiert auf anderen Python-Bibliotheken: NumPy, SciPy und Matplotlib

`scikit-learn` enthält eine Reihe von Implementierungen für verschiedene gängige Algorithmen des maschinellen Lernens.

Examples

Installation von Scikit-Learn

Die aktuelle stabile Version von scikit-learn erfordert :

- Python (> = 2,6 oder> = 3,3),
- NumPy (> = 1.6.1),
- SciPy (> = 0,9).

Für die meisten Installations- `pip` Paketmanager Python und alle seine Abhängigkeiten installieren:

```
pip install scikit-learn
```

Bei Linux-Systemen wird jedoch empfohlen, den `conda` Package Manager zu verwenden, um mögliche `conda` zu vermeiden

```
conda install scikit-learn
```

Um zu überprüfen, ob Sie `scikit-learn` , führen Sie es in der Shell aus:

```
python -c 'import sklearn; print(sklearn.__version__)'
```

Windows- und Mac OSX-Installation:

Canopy und Anaconda *bieten* eine aktuelle Version von *Scikit-Learn* sowie eine große *Sammlung* wissenschaftlicher Python-Bibliotheken für Windows, Mac OSX (auch für Linux relevant).

Trainieren Sie einen Klassifikator mit Kreuzvalidierung

Verwenden des Iris-Datasets:

```
import sklearn.datasets
iris_dataset = sklearn.datasets.load_iris()
```

```
X, y = iris_dataset['data'], iris_dataset['target']
```

Die Daten werden in Zug- und Testsets aufgeteilt. Dazu verwenden wir die Utility-Funktion `train_test_split` , um `X` und `y` (Daten- und `train_size=0.75`) mit der Option `train_size=0.75` (zufällig 75% der Daten) zufällig zu `train_size=0.75` .

Trainingsdatensätze werden in einen k nächstgelegenen Nachbarnklassifizierer eingespeist. Das Verfahren `fit` des Klassifikators wird das Modell zu den Daten passen.

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.75)
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
```

Zum Schluss Vorhersage der Qualität der Testprobe:

```
clf.score(X_test, y_test) # Output: 0.94736842105263153
```

Durch die Verwendung eines Paares von Zug- und Testsätzen könnten wir aufgrund der willkürlichen Auswahl der Datenaufteilung eine verzerrte Einschätzung der Qualität des Klassifizierers erhalten. Durch die Verwendung *der Kreuzvalidierung können* wir den Klassifikator an verschiedene Zug- / Test-Teilmengen der Daten anpassen und einen Durchschnitt aller Genauigkeitsergebnisse machen. Die Funktion `cross_val_score` passt einen Klassifikator an die Eingabedaten unter Verwendung der Kreuzvalidierung an. Es kann die Anzahl der zu verwendenden Splits (Falten) (5 im folgenden Beispiel) als Eingabe übernehmen.

```
from sklearn.cross_validation import cross_val_score
scores = cross_val_score(clf, X, y, cv=5)
print(scores)
# Output: array([ 0.96666667,  0.96666667,  0.93333333,  0.96666667,  1.        ])
print "Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() / 2)
# Output: Accuracy: 0.97 (+/- 0.03)
```

Pipelines erstellen

Das Auffinden von Mustern in Daten erfolgt häufig in einer Kette von Datenverarbeitungsschritten, z. B. Merkmalsauswahl, Normalisierung und Klassifizierung. In `sklearn` wird dazu eine Pipeline von Stufen verwendet.

Der folgende Code zeigt beispielsweise eine Pipeline, die aus zwei Stufen besteht. Die erste skaliert die Features und die zweite bildet einen Klassifizierer für das resultierende erweiterte Dataset aus:

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier

pipeline = make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=4))
```

Nachdem die Pipeline erstellt wurde, können Sie sie wie eine reguläre Phase verwenden (abhängig von den jeweiligen Schritten). Hier verhält sich beispielsweise die Pipeline wie ein Klassifikator. Folglich können wir es wie folgt verwenden:

```
# fitting a classifier
pipeline.fit(X_train, y_train)
# getting predictions for the new data sample
pipeline.predict_proba(X_test)
```

Schnittstellen und Konventionen:

Verschiedene Operationen mit Daten werden mit speziellen Klassen durchgeführt.

Die meisten Klassen gehören zu einer der folgenden Gruppen:

- Klassifizierungsalgorithmen (abgeleitet von `sklearn.base.ClassifierMixin`) zur Lösung von Klassifizierungsproblemen
- Regressionsalgorithmen (abgeleitet von `sklearn.base.RegressorMixin`) zur Lösung des Problems der Rekonstruktion kontinuierlicher Variablen (Regressionsproblem)
- Datentransformationen (abgeleitet von `sklearn.base.TransformerMixin`), die die Daten vorverarbeiten

Daten werden in `numpy.array` s gespeichert (andere Array-ähnliche Objekte wie `pandas.DataFrame` s werden jedoch akzeptiert, wenn diese in `numpy.array` s konvertierbar sind).

Jedes Objekt in den Daten wird durch eine Reihe von Merkmalen beschrieben. Die allgemeine Konvention besteht darin, dass Datenmuster mit einem Array dargestellt werden, wobei die erste Dimension die Datenmuster-ID und die zweite Dimension die Feature-ID ist.

```
import numpy
data = numpy.arange(10).reshape(5, 2)
print(data)

Output:
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
```

In den `sklearn` Konventionen enthält das `sklearn` Dataset 5 Objekte, die jeweils durch 2 Merkmale beschrieben werden.

Beispieldatensätze

Um das Testen zu `sklearn` stellt `sklearn` einige integrierte Datensätze im Modul `sklearn.datasets` . Lassen Sie uns beispielsweise Fisher's Iris-Dataset laden:

```
import sklearn.datasets
iris_dataset = sklearn.datasets.load_iris()
iris_dataset.keys()
```

```
['target_names', 'data', 'target', 'DESCR', 'feature_names']
```

Sie können die vollständige Beschreibung, die Namen der Features und die Namen der Klassen (`target_names`) `target_names` . Diese werden als Zeichenfolgen gespeichert.

Wir sind an den Daten und Klassen interessiert, die in `data` und `target` gespeichert sind. Konventionell werden diese als `X` und `y`

```
X, y = iris_dataset['data'], iris_dataset['target']
X.shape, y.shape
((150, 4), (150,))
```

```
numpy.unique(y)
array([0, 1, 2])
```

Formen von `X` und `y` besagen, dass es 150 Proben mit 4 Merkmalen gibt. Jedes Beispiel gehört zu einer der folgenden Klassen: 0, 1 oder 2.

`X` und `y` können jetzt beim Trainieren eines Klassifikators verwendet werden, indem die Methode `fit()` des Klassifizierers `fit()` wird.

Hier ist die vollständige Liste der vom Modul `sklearn.datasets` bereitgestellten Datensätze mit ihrer Größe und beabsichtigten Verwendung:

Laden mit	Beschreibung	Größe	Verwendungszweck
`load_boston()`	Boston-Hauspreis-Datensatz	506	Regression
`load_breast_cancer()`	Brustkrebs Wisconsin-Datensatz	569	Klassifizierung (binär)
`load_diabetes()`	Diabetes-Datensatz	442	Regression
`load_digits(n_class)`	Ziffern-Datensatz	1797	Einstufung
`load_iris()`	Iris-Datensatz	150	Klassifizierung (Multi-Class)
`load_linnerud()`	Linnerud-Datensatz	20	multivariate Regression

Beachten Sie, dass (Quelle: http://scikit-learn.org/stable/datasets/) :

> Diese Datensätze sind hilfreich, um das Verhalten der verschiedenen im Scikit implementierten Algorithmen schnell zu veranschaulichen. Sie sind jedoch oft zu klein, um für maschinelle Lernaufgaben in der realen Welt repräsentativ zu sein.

Neben diesen integrierten Beispieldatensätzen für Spielzeug bietet `sklearn.datasets` auch `sklearn.datasets` zum Laden externer Datensätze:

- `load_mlcomp` zum Laden von Beispiel-Datasets aus dem mlcomp.org- Repository (Beachten Sie, dass die Datasets zuvor heruntergeladen werden müssen). Hier ist ein Verwendungsbeispiel.
- `fetch_lfw_pairs` und `fetch_lfw_people` zum Laden von LFW-Paare (`fetch_lfw_people` Faces in the Wild) -Paar von http://vis-www.cs.umass.edu/lfw/ , zur Gesichtsüberprüfung (bzw. Gesichtserkennung). Dieser Datensatz ist größer als 200 MB. Hier ist ein Verwendungsbeispiel.

Erste Schritte mit Scikit-Learn online lesen: https://riptutorial.com/de/scikit-learn/topic/1035/erste-schritte-mit-scikit-learn

Kapitel 2: Dimensionsreduzierung (Feature-Auswahl)

Examples

Reduzierung der Dimension durch Hauptkomponentenanalyse

Die Hauptkomponentenanalyse findet Folgen linearer Kombinationen der Merkmale. Die erste lineare Kombination maximiert die Varianz der Merkmale (abhängig von einer Einheitsbeschränkung). Jede der folgenden Linearkombinationen maximiert die Varianz der Merkmale in dem Unterraum, der orthogonal zu dem ist, der von den vorherigen Linearkombinationen aufgespannt wurde.

Eine gemeinsame Dimension Reduktionstechnik ist nur die *k* ersten solchen Linearkombinationen zu verwenden. Angenommen, die Merkmale sind eine Matrix *X* aus *n* Zeilen und *m* Spalten. Die ersten *k* Linearkombinationen bilden eine Matrix *& bgr; _k* aus *m* Zeilen und *k* Spalten. Das Produkt *Xβ* hat *n* Zeilen und *k* Spalten. Somit kann die resultierende Matrix *β k* als eine Reduktion von *m* auf *k* Dimensionen betrachtet werden, wobei die hochvarianzigen Teile der ursprünglichen Matrix *X erhalten* bleiben.

In `scikit-learn` wird PCA mit `sklearn.decomposition.PCA` . Angenommen, wir beginnen mit einer 100 x 7-Matrix, die so konstruiert ist, dass die Varianz nur in den ersten beiden Spalten enthalten ist (durch Skalieren der letzten 5 Spalten):

```
import numpy as np
np.random.seed(123) # we'll set a random seed so that our results are reproducible
X = np.hstack((np.random.randn(100, 2) + (10, 10), 0.001 * np.random.randn(100, 5)))
```

Lassen Sie uns auf 2 Dimensionen reduzieren:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
```

Lassen Sie uns nun die Ergebnisse überprüfen. Hier sind zunächst die Linearkombinationen:

```
pca.components_
# array([[ -2.84271217e-01,  -9.58743893e-01,  -8.25412629e-05,
#           1.96237855e-05,  -1.25862328e-05,   8.27127496e-05,
#          -9.46906600e-05],
#        [ -9.58743890e-01,   2.84271223e-01,  -7.33055823e-05,
#          -1.23188872e-04,  -1.82458739e-05,   5.50383246e-05,
#           1.96503690e-05]])
```

Beachten Sie, dass die ersten beiden Komponenten in jedem Vektor um mehrere Größenordnungen größer sind als die anderen. Dies zeigt, dass die PCA erkannt hat, dass die

Varianz hauptsächlich in den ersten beiden Spalten enthalten ist.

Um das Verhältnis der durch diese PCA erklärten Varianz zu überprüfen, können wir `pca.explained_variance_ratio_` untersuchen:

```
pca.explained_variance_ratio_
# array([ 0.57039059,  0.42960728])
```

Dimensionsreduzierung (Feature-Auswahl) online lesen: https://riptutorial.com/de/scikit-learn/topic/4829/dimensionsreduzierung--feature-auswahl-

Kapitel 3: Einstufung

Examples

Verwenden von Support Vector Machines

Support-Vektor-Maschinen sind eine Familie von Algorithmen, die versuchen, eine (möglicherweise hochdimensionale) Hyperebene zwischen zwei markierten Punktesätzen zu übergeben, so dass der Abstand der Punkte von der Ebene in gewissem Sinne optimal ist. SVMs können für die Klassifizierung oder Regression verwendet werden (entsprechend `sklearn.svm.SVC` bzw. `sklearn.svm.SVR`).

Beispiel:

Angenommen, wir arbeiten in einem 2D-Raum. Zuerst erstellen wir einige Daten:

```
import numpy as np
```

Jetzt erstellen wir *x* und *y* :

```
x0, x1 = np.random.randn(10, 2), np.random.randn(10, 2) + (1, 1)
x = np.vstack((x0, x1))

y = [0] * 10 + [1] * 10
```

Beachten Sie, dass *x* aus zwei Gaussianern besteht: einer um *(0, 0)* und einer um *(1, 1)* .

Um einen Klassifikator zu erstellen, können wir Folgendes verwenden:

```
from sklearn import svm

svm.SVC(kernel='linear').fit(x, y)
```

Sehen wir uns die Vorhersage für *(0, 0) an* :

```
>>> svm.SVC(kernel='linear').fit(x, y).predict([[0, 0]])
array([0])
```

Die Vorhersage lautet, dass die Klasse 0 ist.

Für die Regression können wir auf ähnliche Weise tun:

```
svm.SVR(kernel='linear').fit(x, y)
```

RandomForestClassifier

Eine zufällige Gesamtstruktur ist ein Metaschätzer, der eine Reihe von

Entscheidungsbaumklassifizierern auf verschiedene Unterabtastungen des Datensatzes anpasst und die Mittelwertbildung verwendet, um die Vorhersagegenauigkeit und die Kontrolle der Anpassung zu verbessern.

Ein einfaches Anwendungsbeispiel:

Einführen:

```
from sklearn.ensemble import RandomForestClassifier
```

Zugdaten und Zieldaten definieren:

```
train = [[1,2,3],[2,5,1],[2,1,7]]
target = [0,1,0]
```

Die Werte in `target` für das Label, das Sie vorhersagen möchten.

Initiieren Sie ein RandomForest-Objekt und führen Sie das Learn (Fit) aus:

```
rf = RandomForestClassifier(n_estimators=100)
rf.fit(train, target)
```

Vorhersagen:

```
test = [2,2,3]
predicted = rf.predict(test)
```

Klassifizierungsberichte analysieren

Erstellen Sie einen Textbericht mit den wichtigsten Klassifizierungsmetriken, einschließlich Genauigkeit und Rückruf , f1-Score (harmonischer Mittelwert für Genauigkeit und Rückruf) und Unterstützung (Anzahl der Beobachtungen dieser Klasse im Trainingssatz).

Beispiel aus `sklearn` docs :

```
from sklearn.metrics import classification_report
y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

Ausgabe -

```
             precision    recall  f1-score   support

    class 0       0.50      1.00      0.67         1
    class 1       0.00      0.00      0.00         1
    class 2       1.00      0.67      0.80         3

avg / total       0.70      0.60      0.61         5
```

GradientBoostingClassifier

Steigungsverstärkung zur Klassifizierung. Der Gradientenverstärkungsklassifizierer ist ein additives Ensemble eines Basismodells, dessen Fehler in aufeinanderfolgenden Iterationen (oder Stufen) durch Hinzufügen von Regressionsbäumen korrigiert werden, die die Residuen korrigieren (den Fehler der vorherigen Stufe).

Einführen:

```
from sklearn.ensemble import GradientBoostingClassifier
```

Erstellen Sie einige Spielzeugklassifizierungsdaten

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()

X, y = iris_dataset.data, iris_dataset.target
```

Lassen Sie uns diese Daten in Trainings- und Testsets aufteilen.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)
```

Instanziieren Sie ein `GradientBoostingClassifier` Modell mit den Standardparametern.

```
gbc = GradientBoostingClassifier()
gbc.fit(X_train, y_train)
```

Lassen Sie uns es auf dem Test-Set bewerten

```
# We are using the default classification accuracy score
>>> gbc.score(X_test, y_test)
1
```

Standardmäßig sind 100 Schätzer erstellt

```
>>> gbc.n_estimators
100
```

Dies kann gesteuert werden, indem `n_estimators` während der Initialisierungszeit auf einen anderen Wert gesetzt wird.

Ein Entscheidungsbaum

Ein Entscheidungsbaum ist ein Klassifizierer, der eine Folge von ausführlichen Regeln (wie a> 7) verwendet, die leicht verständlich sind.

Das folgende Beispiel trainiert einen Entscheidungsbaum-Klassifizierer unter Verwendung von drei Merkmalsvektoren der Länge 3 und sagt dann das Ergebnis für einen bislang unbekannten vierten Merkmalsvektor, den sogenannten Testvektor, voraus.

```
from sklearn.tree import DecisionTreeClassifier

# Define training and target set for the classifier
train = [[1,2,3],[2,5,1],[2,1,7]]
target = [10,20,30]

# Initialize Classifier.
# Random values are initialized with always the same random seed of value 0
# (allows reproducible results)
dectree = DecisionTreeClassifier(random_state=0)
dectree.fit(train, target)

# Test classifier with other, unknown feature vector
test = [2,2,3]
predicted = dectree.predict(test)

print predicted
```

Ausgabe kann visualisiert werden mit:

```
import pydot
import StringIO

dotfile = StringIO.StringIO()
tree.export_graphviz(dectree, out_file=dotfile)
(graph,)=pydot.graph_from_dot_data(dotfile.getvalue())
graph.write_png("dtree.png")
graph.write_pdf("dtree.pdf")
```

Klassifizierung mit logistischer Regression

In LR Classifier werden die Wahrscheinlichkeiten, die die möglichen Ergebnisse einer einzelnen Studie beschreiben, mithilfe einer logistischen Funktion modelliert. Es ist in der `linear_model` Bibliothek implementiert

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

Die Implementierung von sklearn LR kann für die binäre, One-vs-Rest- oder multinomiale logistische Regression mit optionaler L2- oder L1-Regularisierung verwendet werden. Betrachten wir zum Beispiel eine binäre Klassifizierung in einem Beispiel-Sklearn-Dataset

```
from sklearn.datasets import make_hastie_10_2

X,y = make_hastie_10_2(n_samples=1000)
```

Dabei ist X ein Array mit `n_samples X 10` und y ist die Zielbezeichnung -1 oder +1.

Verwenden Sie den Train-Test-Split, um die Eingabedaten in Trainings- und Testsets aufzuteilen (70% -30%).

```
from sklearn.model_selection import train_test_split
#sklearn.cross_validation in older scikit versions

data_train, data_test, labels_train, labels_test = train_test_split(X,y, test_size=0.3)
```

Die Verwendung des LR-Klassifikators ähnelt anderen Beispielen

```
# Initialize Classifier.
LRC = LogisticRegression()
LRC.fit(data_train, labels_train)

# Test classifier with the test data
predicted = LRC.predict(data_test)
```

Verwende die Verwirrungsmatrix, um die Ergebnisse zu visualisieren

```
from sklearn.metrics import confusion_matrix

confusion_matrix(predicted, labels_test)
```

Einstufung online lesen: https://riptutorial.com/de/scikit-learn/topic/2468/einstufung

Kapitel 4: Merkmalsauswahl

Examples

Low-Varance Feature Entfernung

Dies ist eine sehr grundlegende Feature-Auswahlmethode.

Der Grundgedanke ist, dass ein Feature, wenn es konstant ist (dh eine Varianz von 0 hat), nicht zum Auffinden interessanter Muster verwendet werden kann und aus dem Datensatz entfernt werden kann.

Ein heuristischer Ansatz zur Beseitigung von Merkmalen besteht daher darin, zuerst alle Merkmale zu entfernen, deren Varianz unter einem (niedrigen) Schwellenwert liegt.

Bauen Sie das Beispiel in der Dokumentation auf . Nehmen wir an, wir beginnen mit

```
X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
```

Es gibt hier *3* boolesche Funktionen mit jeweils *6* Instanzen. Angenommen, wir möchten diejenigen entfernen, die in mindestens 80% der Fälle konstant sind. Einige Wahrscheinlichkeitsrechnungen zeigen, dass diese Merkmale eine Abweichung von weniger als *0,8 * (1 - 0,8) aufweisen müssen* . Folglich können wir verwenden

```
from sklearn.feature_selection import VarianceThreshold
sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
sel.fit_transform(X)
# Output: array([[0, 1],
                 [1, 0],
                 [0, 0],
                 [1, 1],
                 [1, 0],
                 [1, 1]])
```

Beachten Sie, wie das erste Feature entfernt wurde.

Diese Methode sollte mit Vorsicht angewendet werden, da eine geringe Varianz nicht unbedingt bedeutet, dass eine Funktion "uninteressant" ist. Betrachten Sie das folgende Beispiel, in dem wir ein Dataset erstellen, das *3* Features enthält. Die ersten beiden bestehen aus zufällig verteilten Variablen und die dritten aus gleichmäßig verteilten Variablen.

```
from sklearn.feature_selection import VarianceThreshold
import numpy as np

# generate dataset
np.random.seed(0)

feat1 = np.random.normal(loc=0, scale=.1, size=100) # normal dist. with mean=0 and std=.1
feat2 = np.random.normal(loc=0, scale=10, size=100) # normal dist. with mean=0 and std=10
```

```
feat3 = np.random.uniform(low=0, high=10, size=100) # uniform dist. in the interval [0,10)
data = np.column_stack((feat1,feat2,feat3))

data[:5]
# Output:
# array([[  0.17640523,  18.83150697,   9.61936379],
#        [  0.04001572, -13.47759061,   2.92147527],
#        [  0.0978738 , -12.70484998,   2.4082878 ],
#        [  0.22408932,   9.69396708,   1.00293942],
#        [  0.1867558 , -11.73123405,   0.1642963 ]])

np.var(data, axis=0)
# Output: array([  1.01582662e-02,   1.07053580e+02,   9.07187722e+00])

sel = VarianceThreshold(threshold=0.1)
sel.fit_transform(data)[:5]
# Output:
# array([[ 18.83150697,   9.61936379],
#        [-13.47759061,   2.92147527],
#        [-12.70484998,   2.4082878 ],
#        [  9.69396708,   1.00293942],
#        [-11.73123405,   0.1642963 ]])
```

Nun wurde das erste Feature aufgrund seiner geringen Varianz entfernt, während das dritte Feature (das ist das uninteressanteste) beibehalten wurde. In diesem Fall wäre es sinnvoller gewesen, einen *Variationskoeffizienten* zu berücksichtigen, da er unabhängig von der Skalierung ist.

Merkmalsauswahl online lesen: https://riptutorial.com/de/scikit-learn/topic/4909/merkmalsauswahl

Kapitel 5: Modellauswahl

Examples

Kreuzvalidierung

Das Erlernen der Parameter einer Vorhersagefunktion und das Testen derselben mit denselben Daten ist ein methodologischer Fehler: Ein Modell, das nur die Bezeichnungen der gerade wiedergegebenen Stichproben wiederholt, hätte eine perfekte Punktzahl, würde jedoch nichts Nützliches für noch nicht vorhersagen unsichtbare Daten. Diese Situation wird als **Überanpassung bezeichnet** . Um dies zu vermeiden, ist es üblich , wenn ein (überwachtes) maschinelles Lernen Experiment durchführt Teil der zur Verfügung stehenden Daten zu halten, wie ein **Test** - **Set** `X_test, y_test` . Beachten Sie, dass das Wort „Experiment“ nicht nur für die akademische Verwendung gedacht ist, da selbst in kommerziellen Umgebungen maschinelles Lernen normalerweise experimentell beginnt.

Beim Scikit-Learn kann eine zufällige Aufteilung in Trainings- und Test-Sets mit der Hilfsfunktion train_test_split schnell berechnet werden. Laden wir die Iris-Datei, um eine lineare Unterstützungsvektor-Maschine darauf zu laden:

```
>>> import numpy as np
>>> from sklearn import cross_validation
>>> from sklearn import datasets
>>> from sklearn import svm

>>> iris = datasets.load_iris()
>>> iris.data.shape, iris.target.shape
((150, 4), (150,))
```

Wir können jetzt schnell einen Trainingssatz testen und dabei 40% der Daten zur Prüfung (Bewertung) unseres Klassifikators heraushalten:

```
>>> X_train, X_test, y_train, y_test = cross_validation.train_test_split(
...     iris.data, iris.target, test_size=0.4, random_state=0)

>>> X_train.shape, y_train.shape
((90, 4), (90,))
>>> X_test.shape, y_test.shape
((60, 4), (60,))
```

Nun, nachdem wir Trainings- und Test-Sets haben, können wir es verwenden:

```
>>> clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
>>> clf.score(X_test, y_test)
```

K-Fold Cross Validation

Die K-fache Kreuzvalidierung ist ein systematischer Prozess, um das Train / Test-Split-Verfahren

mehrmals zu wiederholen, um die Varianz zu reduzieren, die mit einer einzelnen Testphase von Train / Test-Split verbunden ist. Sie teilen im Wesentlichen das gesamte Dataset in "gleich große" Falten auf, und jede Faltung wird einmal zum Testen des Modells und K-1-mal zum Trainieren des Modells verwendet.

Die Scikit-Bibliothek bietet mehrere Falttechniken an. Ihre Verwendung hängt von den Eigenschaften der Eingabedaten ab. Einige Beispiele sind

K-Fold

Sie teilen im Wesentlichen das gesamte Dataset in "gleich große" Falten auf, und jede Faltung wird einmal für das Testen des Modells und für K-1-Zeiten zum Trainieren des Modells verwendet.

```
from sklearn.model_selection import KFold
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([1, 2, 1, 2])
cv = KFold(n_splits=3, random_state=0)

for train_index, test_index in cv.split(X):
...    print("TRAIN:", train_index, "TEST:", test_index)

TRAIN: [2 3] TEST: [0 1]
TRAIN: [0 1 3] TEST: [2]
TRAIN: [0 1 2] TEST: [3]
```

`StratifiedKFold` ist eine Variation der k-Fache, die geschichtete Falten zurückgibt: Jeder Satz enthält ungefähr den gleichen Prozentsatz von Abtastwerten jeder Zielklasse wie der vollständige Satz

ShuffleSplit

Wird verwendet, um eine benutzerdefinierte Anzahl unabhängiger Zug- / Testdatensatzaufteilungen zu generieren. Die Proben werden zuerst gemischt und dann in ein Paar Zug- und Testsätze aufgeteilt.

```
from sklearn.model_selection import ShuffleSplit
X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([1, 2, 1, 2])
cv = ShuffleSplit(n_splits=3, test_size=.25, random_state=0)

for train_index, test_index in cv.split(X):
...    print("TRAIN:", train_index, "TEST:", test_index)

TRAIN: [3 1 0] TEST: [2]
TRAIN: [2 1 3] TEST: [0]
TRAIN: [0 2 1] TEST: [3]
```

`StratifiedShuffleSplit` ist eine Variation von ShuffleSplit, die geschichtete Aufteilungen zurückgibt, dh Splits erzeugt, indem für jede Zielklasse derselbe Prozentsatz wie im gesamten Satz

beibehalten wird.

Andere Falttechniken wie Leave One / P Out und TimeSeriesSplit (eine Variation der K-Faltung) sind in der Bibliothek scikit model_selection verfügbar.

Modellauswahl online lesen: https://riptutorial.com/de/scikit-learn/topic/4901/modellauswahl

Kapitel 6: Receiver Operating Characteristic (ROC)

Examples

Einführung in ROC und AUC

Beispiel einer Receiver Operating Characteristic (ROC) -Metrik zur Bewertung der Klassifizierer-Ausgabequalität.

ROC-Kurven weisen normalerweise eine echte positive Rate auf der Y-Achse und eine falsche positive Rate auf der X-Achse auf. Dies bedeutet, dass die obere linke Ecke des Diagramms der "ideale" Punkt ist - eine falsch positive Rate von Null und eine echte positive Rate von Eins. Dies ist nicht sehr realistisch, bedeutet jedoch, dass eine größere Fläche unter der Kurve (AUC) normalerweise besser ist.

Die "Steilheit" der ROC-Kurven ist ebenfalls wichtig, da sie ideal ist, um die wahre positive Rate zu maximieren und gleichzeitig die falsch positive Rate zu minimieren.

Ein einfaches Beispiel:

```
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
```

Beliebige `y` Werte - im realen Fall sind dies die vorhergesagten Zielwerte (`model.predict(x_test)`):

```
y = np.array([1,1,2,2,3,3,4,4,2,3])
```

Scores ist die mittlere Genauigkeit der angegebenen Testdaten und Labels (`model.score(X,Y)`):

```
scores = np.array([0.3, 0.4, 0.95,0.78,0.8,0.64,0.86,0.81,0.9, 0.8])
```

Berechnen Sie die ROC-Kurve und die AUC:

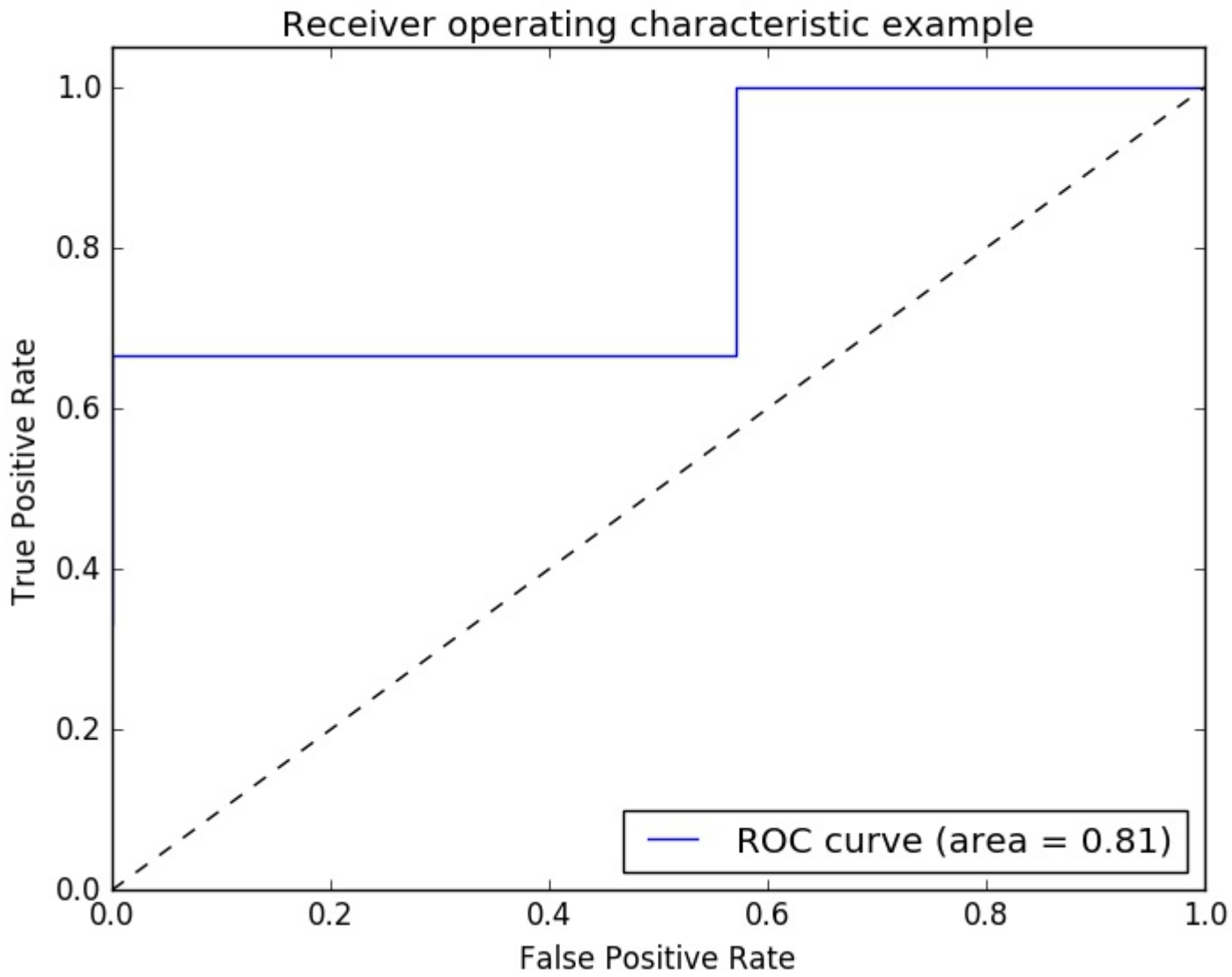
```
fpr, tpr, thresholds = metrics.roc_curve(y, scores, pos_label=2)
roc_auc = metrics.auc(fpr, tpr)
```

Plotten:

```
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic example')
plt.legend(loc="lower right")
plt.show()
```

Ausgabe:



Hinweis: Die Quellen wurden diesen Links1 und Link2 entnommen

ROC-AUC-Bewertung mit Überschreibung und Kreuzvalidierung

Man benötigt die vorhergesagten Wahrscheinlichkeiten, um den ROC-AUC-Wert (Bereich unter der Kurve) zu berechnen. Das `cross_val_predict` verwendet die `predict` der Klassifizierer. Um den ROC-AUC-Score erhalten zu können, kann man den Klassifizierer einfach subklassieren und die `predict` Methode überschreiben, so dass er wie `predict_proba` .

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import cross_val_predict
```

```
from sklearn.metrics import roc_auc_score

class LogisticRegressionWrapper(LogisticRegression):
    def predict(self, X):
        return super(LogisticRegressionWrapper, self).predict_proba(X)

X, y = make_classification(n_samples = 1000, n_features=10, n_classes = 2, flip_y = 0.5)

log_reg_clf = LogisticRegressionWrapper(C=0.1, class_weight=None, dual=False,
             fit_intercept=True)

y_hat = cross_val_predict(log_reg_clf, X, y)[:,1]

print("ROC-AUC score: {}".format(roc_auc_score(y, y_hat)))
```

Ausgabe:

```
ROC-AUC score: 0.724972396025
```

Receiver Operating Characteristic (ROC) online lesen: https://riptutorial.com/de/scikit-learn/topic/5945/receiver-operating-characteristic--roc-

Kapitel 7: Regression

Examples

Gewöhnliche kleinste Quadrate

Gewöhnliche kleinste Quadrate ist eine Methode zum Ermitteln der linearen Kombination von Merkmalen, die dem beobachteten Ergebnis im folgenden Sinne am besten entspricht.

Wenn der Vektor der Ergebnisse, die vorhergesagt werden sollen, *y* ist und die erklärenden Variablen die Matrix *X bilden* , dann wird OLS den Vektor *β*- Lösung finden

$min_{\beta} | y\hat{} - y |_2^2$

wobei $y\hat{} = X\beta$ die lineare Vorhersage ist.

In sklearn erfolgt dies mit `sklearn.linear_model.LinearRegression` .

Anwendungskontext

OLS sollte nur bei Regressionsproblemen angewendet werden, es ist im Allgemeinen für Klassifizierungsprobleme ungeeignet: Kontrast

- Ist eine E-Mail-Spam? (Klassifizierung)
- Welcher lineare Zusammenhang zwischen Upvotes hängt von der Länge der Antwort ab? (Regression)

Beispiel

Lassen Sie uns ein lineares Modell mit etwas Rauschen erzeugen und sehen Sie dann, `LinearRegression` LinearRegression das lineare Modell rekonstruieren kann.

Zuerst erzeugen wir die `X` Matrix:

```
import numpy as np

X = np.random.randn(100, 3)
```

Jetzt erzeugen wir das `y` als lineare Kombination von `X` mit etwas Rauschen:

```
beta = np.array([[1, 1, 0]])
y = (np.dot(x, beta.T) + 0.01 * np.random.randn(100, 1))[:, 0]
```

Man beachte, dass die wahre lineare Kombination, die `y` , durch beta angegeben wird.

Um dies allein aus `X` und `y` zu rekonstruieren, wollen wir folgendes tun:

```
>>> linear_model.LinearRegression().fit(x, y).coef_
```

```
array([  9.97768469e-01,   9.98237634e-01,   7.55016533e-04])
```

Beachten Sie, dass dieser Vektor der `beta` sehr ähnlich ist.

Regression online lesen: https://riptutorial.com/de/scikit-learn/topic/5190/regression

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Scikit-Learn	Alleo, Ami Tavory, Community, Gabe, Gal Dreiman, panty, Sean Easter, user2314737
2	Dimensionsreduzierung (Feature-Auswahl)	Ami Tavory, DataSwede, Gal Dreiman, Sean Easter, user2314737
3	Einstufung	Ami Tavory, Drew, Gal Dreiman, hashcode55, Mechanic, Raghav RV, Sean Easter, tfv, user6903745, Wayne Werner
4	Merkmalsauswahl	Ami Tavory, user2314737
5	Modellauswahl	Gal Dreiman, Mechanic
6	Receiver Operating Characteristic (ROC)	Gal Dreiman, Gorkem Ozkaya
7	Regression	Ami Tavory, draco_alpine