EBook Gratuito

APPRENDIMENTO scikit-learn

Free unaffiliated eBook created from
Stack Overflow contributors.

#scikit-learn

Sommario

Di

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

Capitolo 1: Iniziare con scikit-learn

Osservazioni

`scikit-learn` è una libreria open source generica per l'analisi dei dati scritta in python. È basato su altre librerie python: NumPy, SciPy e matplotlib

`scikit-learn` contiene una serie di implementazioni per diversi algoritmi popolari di machine learning.

Examples

Installazione di scikit-learn

L'attuale versione stabile di scikit-learn richiede :

- Python (> = 2.6 o> = 3.3),
- NumPy (> = 1.6.1),
- SciPy (> = 0,9).

Per la maggior parte di installazione `pip` gestore di pacchetti python può installare python e tutte le sue dipendenze:

```
pip install scikit-learn
```

Tuttavia, per i sistemi Linux si consiglia di utilizzare il gestore pacchetti `conda` per evitare possibili processi di compilazione

```
conda install scikit-learn
```

Per verificare di avere `scikit-learn` , esegui in shell:

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

Installazione di Windows e Mac OSX:

Canopy e Anaconda hanno entrambi una versione recente di *scikit-learn* , oltre a un ampio set di librerie scientifiche Python per Windows, Mac OSX (anche per Linux).

Addestrare un classificatore con convalida incrociata

Utilizzo del set di dati dell'iride:

```
import sklearn.datasets
```

```
iris_dataset = sklearn.datasets.load_iris()
X, y = iris_dataset['data'], iris_dataset['target']
```

I dati vengono suddivisi in set di treni e test. Per fare ciò utilizziamo la funzione di utilità `train_test_split` per dividere a caso sia `X` che `y` (vettori di dati e di destinazione) con l'opzione `train_size=0.75` (i set di allenamento contengono il 75% dei dati).

I set di dati di addestramento vengono inseriti in un classificatore dei vicini più vicino k . Il metodo di `fit` del classificatore adatta il modello ai dati.

```
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)
```

Infine, prevedendo la qualità del campione di prova:

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

Utilizzando una coppia di treni e set di test potremmo ottenere una stima parziale della qualità del classificatore a causa della scelta arbitraria della suddivisione dei dati. Utilizzando la *convalida incrociata* possiamo adattare il classificatore su sottoinsiemi di treni / test diversi e ottenere una media su tutti i risultati di precisione. La funzione `cross_val_score` adatta a un classificatore ai dati di input utilizzando la convalida incrociata. Può prendere come input il numero di diverse suddivisioni (pieghe) da utilizzare (5 nell'esempio seguente).

```
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)
```

Creazione di pipeline

La ricerca di schemi nei dati spesso procede in una catena di fasi di elaborazione dei dati, ad esempio selezione delle caratteristiche, normalizzazione e classificazione. In `sklearn` , viene utilizzata una pipeline di fasi.

Ad esempio, il codice seguente mostra una pipeline composta da due fasi. Il primo ridimensiona le funzioni e il secondo allena un classificatore sul set di dati aumentato risultante:

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

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

Una volta che la pipeline è stata creata, puoi usarla come una fase normale (in base ai passaggi

specifici). Qui, ad esempio, la pipeline si comporta come un classificatore. Di conseguenza, possiamo usarlo come segue:

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

Interfacce e convenzioni

Diverse operazioni con i dati vengono eseguite utilizzando classi speciali.

La maggior parte delle classi appartiene a uno dei seguenti gruppi:

- algoritmi di classificazione (derivati da `sklearn.base.ClassifierMixin`) per risolvere problemi di classificazione
- algoritmi di regressione (derivati da `sklearn.base.RegressorMixin`) per risolvere il problema della ricostruzione di variabili continue (problema di regressione)
- trasformazioni di dati (derivate da `sklearn.base.TransformerMixin`) che preelaborano i dati

I dati sono memorizzati in `numpy.array` s (ma altri oggetti tipo `pandas.DataFrame` come `pandas.DataFrame` s sono accettati se sono convertibili in `numpy.array` s)

Ogni oggetto nei dati è descritto da un insieme di caratteristiche, la convenzione generale è che il campione di dati è rappresentato con un array, dove la prima dimensione è id di esempio di dati, la seconda dimensione è id di caratteristica.

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

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

Nel set di dati `sklearn` sopra riportato contiene 5 oggetti ciascuno descritto da 2 caratteristiche.

Set di dati di esempio

Per facilità di test, `sklearn` fornisce alcuni set di dati `sklearn.datasets` nel modulo `sklearn.datasets` . Ad esempio, carichiamo il set di dati dell'iride di Fisher:

```
import sklearn.datasets
iris_dataset = sklearn.datasets.load_iris()
iris_dataset.keys()
['target_names', 'data', 'target', 'DESCR', 'feature_names']
```

Puoi leggere la descrizione completa, i nomi delle funzioni e i nomi delle classi (`target_names`).

Quelli sono memorizzati come stringhe.

Siamo interessati ai dati e alle classi, memorizzati in campi `data` e `target` . Per convenzione quelli sono indicati come `X` e `y`

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

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

Le forme di `X` e `y` dicono che ci sono 150 campioni con 4 caratteristiche. Ogni campione appartiene a una delle seguenti classi: 0, 1 o 2.

`X` e `y` possono ora essere utilizzati nell'addestramento di un classificatore, chiamando il metodo `fit()` del classificatore.

Ecco l'elenco completo dei set di dati forniti dal modulo `sklearn.datasets` con le loro dimensioni e destinazione d'uso:

Carica con	Descrizione	Taglia	uso
`load_boston()`	Dataset dei prezzi delle case di Boston	506	regressione
`load_breast_cancer()`	Dataset Wisconsin del cancro al seno	569	classificazione (binaria)
`load_diabetes()`	Dataset per il diabete	442	regressione
`load_digits(n_class)`	Set di dati Digits	1797	classificazione
`load_iris()`	Set di dati Iris	150	classificazione (multi-classe)
`load_linnerud()`	Set di dati Linnerud	20	regressione multivariata

Nota che (fonte: http://scikit-learn.org/stable/datasets/) :

> Questi set di dati sono utili per illustrare rapidamente il comportamento dei vari algoritmi implementati nello scikit. Tuttavia, sono spesso troppo piccoli per essere rappresentativi dei compiti di apprendimento automatico del mondo reale.

Oltre a questi set di dati di esempio giocattolo `sklearn.datasets` , `sklearn.datasets` fornisce anche funzioni di utilità per il caricamento di set di dati esterni:

- `load_mlcomp` per caricare i dataset di esempio dal repository mlcomp.org (si noti che i set di

dati devono essere scaricati prima). Ecco un esempio di utilizzo.

- `fetch_lfw_pairs` e `fetch_lfw_people` per il caricamento del set di dati `fetch_lfw_people` etichettate in the Wild (LFW) da http://vis-www.cs.umass.edu/lfw/ , utilizzato per la verifica del volto (o del riconoscimento facciale). Questo set di dati è più grande di 200 MB. Ecco un esempio di utilizzo.

Leggi Iniziare con scikit-learn online: https://riptutorial.com/it/scikit-learn/topic/1035/iniziare-con-scikit-learn

Capitolo 2: Classificazione

Examples

Utilizzo di macchine vettoriali di supporto

Support vector machines è una famiglia di algoritmi che tentano di passare un iperpiano (possibilmente di dimensione elevata) tra due insiemi di punti etichettati, in modo tale che la distanza dei punti dal piano sia ottimale in un certo senso. Gli SVM possono essere utilizzati per la classificazione o la regressione (corrispondenti a `sklearn.svm.SVC` e `sklearn.svm.SVR` , rispettivamente.

Esempio:

Supponiamo di lavorare in uno spazio 2D. Innanzitutto, creiamo alcuni dati:

```
import numpy as np
```

Ora creiamo *x* e *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
```

Si noti che *x* è composto da due gaussiani: uno centrato intorno *(0, 0)* e uno centrato attorno *(1, 1)* .

Per costruire un classificatore, possiamo usare:

```
from sklearn import svm

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

Controlliamo la previsione per *(0, 0)* :

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

La previsione è che la classe è 0.

Per la regressione, possiamo fare allo stesso modo:

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

RandomForestClassifier

Una foresta casuale è un meta-estimatore che si adatta a un certo numero di classificatori di alberi decisionali su vari sottocampioni del set di dati e utilizza la media per migliorare l'accuratezza predittiva e il controllo eccessivo.

Un semplice esempio di utilizzo:

Importare:

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

Definire i dati del treno e i dati di destinazione:

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

I valori nella `target` rappresentano l'etichetta che si desidera prevedere.

Avvia un oggetto RandomForest ed esegui learn (fit):

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

prevedere:

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

Analizzando i rapporti di classificazione

Creare un report di testo che mostri le principali metriche di classificazione, inclusi precisione e richiamo , f1-score (la media armonica di precisione e richiamo) e supporto (il numero di osservazioni di quella classe nel set di allenamento).

Esempio dai documenti `sklearn` :

```
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))
```

Produzione -

```
             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

Aumento graduale per la classificazione. Il Gradient Boosting Classifier è un insieme additivo di un modello base il cui errore viene corretto in iterazioni successive (o stadi) mediante l'aggiunta di alberi di regressione che correggono i residui (l'errore della fase precedente).

Importare:

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

Creare alcuni dati di classificazione del giocattolo

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()

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

Cerchiamo di suddividere questi dati in training e set di test.

```
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)
```

Crea un'istanza di un modello `GradientBoostingClassifier` utilizzando i parametri predefiniti.

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

Fateci segnare sul set di prova

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

Di default ci sono 100 stimatori costruiti

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

Questo può essere controllato impostando `n_estimators` su un valore diverso durante il tempo di inizializzazione.

Un albero decisionale

Un albero decisionale è un classificatore che utilizza una sequenza di regole verbose (come a> 7) che può essere facilmente compresa.

L'esempio sotto forma un classificatore ad albero decisionale usando tre vettori di caratteristiche

di lunghezza 3, e quindi predice il risultato per un quarto vettore di feature finora sconosciuto, il cosiddetto vettore di test.

```
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
```

L'output può essere visualizzato usando:

```
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")
```

Classificazione usando la regressione logistica

In LR Classifier, le probabilità che descrivono i possibili esiti di una singola prova sono modellate usando una funzione logistica. È implementato nella libreria `linear_model`

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

L'implementazione di sklearn LR può adattarsi alla regressione logistica binaria, One-vs-Rest o multinomiale con la regolarizzazione opzionale L2 o L1. Ad esempio, consideriamo una classificazione binaria su un set di dati sklearn campione

```
from sklearn.datasets import make_hastie_10_2

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

Dove X è un `n_samples X 10` array ey sono le etichette target -1 o +1.

Utilizzare la suddivisione del test del treno per dividere i dati di input in set di allenamento e test (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)
```

L'utilizzo di LR Classifier è simile ad altri esempi

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

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

Usa la matrice di confusione per visualizzare i risultati

```
from sklearn.metrics import confusion_matrix

confusion_matrix(predicted, labels_test)
```

Leggi Classificazione online: https://riptutorial.com/it/scikit-learn/topic/2468/classificazione

Capitolo 3: Receiver Operating Characteristic (ROC)

Examples

Introduzione a ROC e AUC

Esempio di metrica Receiver Operating Characteristic (ROC) per valutare la qualità di uscita del classificatore.

Le curve ROC presentano tipicamente una velocità positiva reale sull'asse Y e una percentuale di falsi positivi sull'asse X. Ciò significa che l'angolo in alto a sinistra della trama è il punto "ideale" - un tasso di falsi positivi pari a zero e un vero tasso positivo di uno. Questo non è molto realistico, ma significa che un'area più ampia sotto la curva (AUC) di solito è migliore.

Anche la "pendenza" delle curve ROC è importante, poiché è ideale per massimizzare il tasso positivo reale riducendo al minimo il tasso di falsi positivi.

Un semplice esempio:

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

Valori arbitrari `y` : in questo caso i valori target previsti (`model.predict(x_test)`):

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

Il punteggio è l'accuratezza media dei dati di test e delle etichette `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])
```

Calcola la curva ROC e l'AUC:

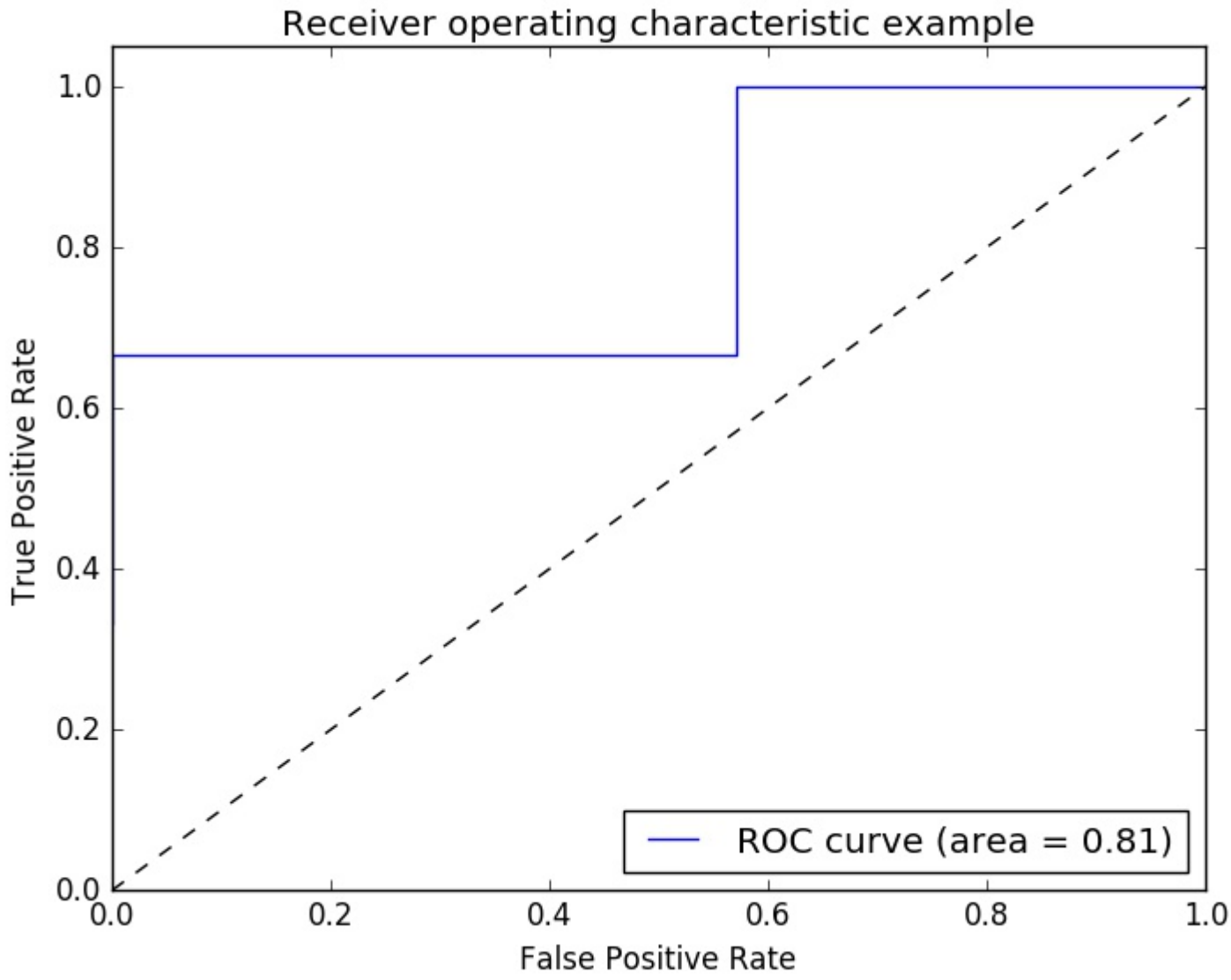
```
fpr, tpr, thresholds = metrics.roc_curve(y, scores, pos_label=2)
roc_auc = metrics.auc(fpr, tpr)
```

Tracciare:

```
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()
```

Produzione:



Nota: le fonti sono state prese da questi link1 e link2

Punteggio ROC-AUC con override e cross validation

Uno ha bisogno delle probabilità previste per calcolare il punteggio ROC-AUC (area sotto la curva). `cross_val_predict` utilizza i metodi di `predict` dei classificatori. Per poter ottenere il punteggio ROC-AUC, si può semplicemente sottoclasse il classificatore, sovrascrivendo il metodo di `predict` , in modo che si `predict_proba` come `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)))
```

produzione:

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

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

Capitolo 4: Regressione

Examples

Minimi quadrati ordinari

Least Squares ordinari è un metodo per trovare la combinazione lineare di caratteristiche che meglio si adatta al risultato osservato nel senso seguente.

Se il vettore dei risultati da prevedere è y , e le variabili esplicative formano la matrice X , allora OLS troverà la soluzione del vettore β

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

dove $y\hat{} = X\beta$ è la previsione lineare.

In sklearn, questo viene fatto usando `sklearn.linear_model.LinearRegression` .

Contesto dell'applicazione

L'OLS dovrebbe essere applicato solo ai problemi di regressione, in genere non è adatto per problemi di classificazione: Contrasto

- E 'una email spam? (Classfication)
- Qual è la relazione lineare tra gli upvotes dipende dalla lunghezza della risposta? (Regressione)

Esempio

`LinearRegression` un modello lineare con un po 'di rumore, quindi vediamo se `LinearRegression` gestisce per ricostruire il modello lineare.

Per prima cosa generiamo la matrice `X` :

```
import numpy as np

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

Ora genereremo la `y` come una combinazione lineare di `X` con un po 'di rumore:

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

Si noti che la vera combinazione lineare che genera `y` è data da `beta.

Per provare a ricostruire questo da solo `X` e `y` , facciamo:

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

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

Si noti che questo vettore è molto simile alla `beta` .

Leggi Regressione online: https://riptutorial.com/it/scikit-learn/topic/5190/regressione

Capitolo 5: Riduzione della dimensionalità (selezione delle caratteristiche)

Examples

Riduzione della dimensione con analisi dei componenti principali

Principal Component Analysis trova sequenze di combinazioni lineari delle caratteristiche. La prima combinazione lineare massimizza la varianza delle caratteristiche (soggetto a un vincolo di unità). Ognuna delle seguenti combinazioni lineari massimizza la varianza delle caratteristiche nel sottospazio ortogonale a quella estesa dalle precedenti combinazioni lineari.

Una tecnica di riduzione delle dimensioni comune consiste nell'usare solo le *k* prima di tali combinazioni lineari. Supponiamo che le caratteristiche siano una matrice *X* di *n* righe e *m* colonne. Le prime *k* combinazioni lineari formano una matrice β_k di *m* righe e *k* colonne. Il prodotto *X* β ha *n* righe e *k* colonne. Pertanto, la matrice risultante β *k* può essere considerata una riduzione da *m* a *k* dimensioni, mantenendo le parti ad alta varianza della matrice originale *X*.

In `scikit-learn` , PCA viene eseguito con `sklearn.decomposition.PCA` . Ad esempio, supponiamo di iniziare con una matrice 100 X 7, costruita in modo che la varianza sia contenuta solo nelle prime due colonne (ridimensionando le ultime 5 colonne):

```
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)))
```

Eseguiamo una riduzione a 2 dimensioni:

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

Ora controlliamo i risultati. Innanzitutto, ecco le combinazioni lineari:

```
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]])
```

Si noti come i primi due componenti di ciascun vettore siano diversi ordini di grandezza più grandi degli altri, mostrando che l'PCA ha riconosciuto che la varianza è contenuta principalmente nelle prime due colonne.

Per verificare il rapporto tra la varianza spiegata da questo PCA, possiamo esaminare `pca.explained_variance_ratio_` :

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

Leggi Riduzione della dimensionalità (selezione delle caratteristiche) online: https://riptutorial.com/it/scikit-learn/topic/4829/riduzione-della-dimensionalita--selezione-delle-caratteristiche-

Capitolo 6: Selezione del modello

Examples

Convalida incrociata

Imparare i parametri di una funzione di predizione e testarla sugli stessi dati è un errore metodologico: un modello che ripeterebbe semplicemente le etichette dei campioni che aveva appena visto avrebbe un punteggio perfetto ma non riuscirebbe a prevedere qualcosa di utile su ancora- dati non visti. Questa situazione è chiamata **overfitting** . Per evitarlo, è pratica comune eseguire un esperimento di apprendimento automatico (supervisionato) per `X_test, y_test` una parte dei dati disponibili come **test set** `X_test, y_test` . Si noti che la parola "esperimento" non è destinata a denotare solo l'uso accademico, perché anche nelle impostazioni commerciali l'apprendimento automatico inizia di solito sperimentalmente.

In scikit, impara una divisione casuale in allenamento e i set di test possono essere rapidamente calcolati con la funzione helper train_test_split . Carichiamo il set di dati dell'iride per adattarlo a una macchina vettoriale di supporto lineare:

```
>>> 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,))
```

Ora possiamo campionare rapidamente un set di allenamento mentre tratteniamo il 40% dei dati per testare (valutare) il nostro classificatore:

```
>>> 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,))
```

Ora, dopo aver impostato i set di allenamento e di test, usiamolo:

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

K-Fold Cross Validation

La cross-validation K-fold è un processo sistematico per ripetere più volte la procedura di split treno / test, al fine di ridurre la varianza associata a una singola prova di split treno / test. In pratica

dividi l'intero set di dati in "pieghe" uguali alle dimensioni di K, e ogni piega viene usata una volta per testare il modello e K-1 volte per addestrare il modello.

Le tecniche di piegatura multiple sono disponibili con la libreria scikit. Il loro utilizzo dipende dalle caratteristiche dei dati di input. Alcuni esempi sono

K-Fold

In pratica dividi l'intero set di dati in "pieghe" uguali alle dimensioni di K, e ogni piega viene usata una volta per testare il modello e K-1 volte per addestrare il modello.

```
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` è una variazione di k-fold che restituisce pieghe stratificate: ogni set contiene approssimativamente la stessa percentuale di campioni di ciascuna classe di destinazione del set completo

ShuffleSplit

Utilizzato per generare un numero definito dall'utente di scissioni di serie di dati indipendenti treno / prova. I campioni vengono prima mescolati e quindi suddivisi in un paio di treni e set di test.

```
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` è una variazione di ShuffleSplit, che restituisce suddivisioni stratificate, vale a dire che crea divisioni mantenendo la stessa percentuale per ogni classe di destinazione come nel set completo.

Altre tecniche di piegatura come Leave One / p Out e TimeSeriesSplit (una variazione di K-fold) sono disponibili nella libreria del modello sci_selection.

Leggi Selezione del modello online: https://riptutorial.com/it/scikit-learn/topic/4901/selezione-del-modello

Capitolo 7: Selezione delle funzionalità

Examples

Rimozione delle caratteristiche di bassa varianza

Questa è una tecnica di selezione delle caratteristiche molto semplice.

L'idea alla base è che se una caratteristica è costante (cioè ha 0 varianza), non può essere utilizzata per trovare modelli interessanti e può essere rimossa dal set di dati.

Di conseguenza, un approccio euristico all'eliminazione delle feature consiste nel rimuovere prima tutte le feature la cui varianza è al di sotto di una soglia (bassa).

Costruendo l' esempio nella documentazione , supponiamo di iniziare

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

Ci sono *3* funzioni booleane qui, ognuna con *6* istanze. Supponiamo di voler rimuovere quelli che sono costanti in almeno l'80% delle istanze. Alcuni calcoli di probabilità mostrano che queste caratteristiche dovranno avere una varianza inferiore a *0.8 * (1 - 0.8)* . Di conseguenza, possiamo usare

```
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]])
```

Nota come è stata rimossa la prima caratteristica.

Questo metodo dovrebbe essere usato con cautela perché una bassa varianza non significa necessariamente che una caratteristica sia "poco interessante". Considera il seguente esempio in cui costruiamo un set di dati che contiene *3* funzioni, le prime due consistono in variabili distribuite casualmente e la terza in variabili uniformemente distribuite.

```
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 ]])
```

Ora la prima funzione è stata rimossa a causa della sua bassa varianza, mentre la terza caratteristica (che è la più poco interessante) è stata mantenuta. In questo caso sarebbe stato più appropriato considerare un *coefficiente di variazione* perché questo è indipendente dal ridimensionamento.

Leggi Selezione delle funzionalità online: https://riptutorial.com/it/scikit-learn/topic/4909/selezione-delle-funzionalita

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con scikit-learn	Alleo, Ami Tavory, Community, Gabe, Gal Dreiman, panty, Sean Easter, user2314737
2	Classificazione	Ami Tavory, Drew, Gal Dreiman, hashcode55, Mechanic, Raghav RV, Sean Easter, tfv, user6903745, Wayne Werner
3	Receiver Operating Characteristic (ROC)	Gal Dreiman, Gorkem Ozkaya
4	Regressione	Ami Tavory, draco_alpine
5	Riduzione della dimensionalità (selezione delle caratteristiche)	Ami Tavory, DataSwede, Gal Dreiman, Sean Easter, user2314737
6	Selezione del modello	Gal Dreiman, Mechanic
7	Selezione delle funzionalità	Ami Tavory, user2314737