EBook Gratis

APRENDIZAJE scikit-learn

Free unaffiliated eBook created from
Stack Overflow contributors.

#scikit-learn

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con scikit-learn

Observaciones

`scikit-learn` es una biblioteca de código abierto de propósito general para el análisis de datos escrito en python. Se basa en otras bibliotecas de python: NumPy, SciPy y matplotlib

`scikit-learn` contiene una serie de implementaciones para diferentes algoritmos populares de aprendizaje automático.

Examples

Instalación de scikit-learn

La versión estable actual de scikit-learn requiere :

- Python (> = 2.6 o> = 3.3),
- NumPy (> = 1.6.1),
- Ciencia ficción (> = 0.9).

Durante la mayor instalación `pip` gestor de paquetes Python puede instalar Python y todas sus dependencias:

```
pip install scikit-learn
```

Sin embargo, para los sistemas Linux, se recomienda utilizar el `conda` paquetes `conda` para evitar posibles procesos de compilación.

```
conda install scikit-learn
```

Para verificar que tiene `scikit-learn` , ejecute en shell:

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

Instalación de Windows y Mac OSX:

Canopy y Anaconda *entregan* una versión reciente de *scikit-learn* , además de un amplio conjunto de bibliotecas científicas de Python para Windows, Mac OSX (también relevante para Linux).

Entrena un clasificador con validación cruzada

Usando el conjunto de datos del iris:

```
import sklearn.datasets
```

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

Los datos se dividen en conjuntos de trenes y pruebas. Para hacer esto, usamos la función de utilidad `train_test_split` para dividir tanto `X` como `y` (datos y vectores objetivo) al azar con la opción `train_size=0.75` (los conjuntos de entrenamiento contienen el 75% de los datos).

Los conjuntos de datos de entrenamiento se introducen en un clasificador de vecinos más cercano a k . El método de `fit` del clasificador ajustará el modelo a los datos.

```
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)
```

Finalmente prediciendo la calidad en la muestra de prueba:

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

Al utilizar un par de conjuntos de trenes y pruebas, podríamos obtener una estimación sesgada de la calidad del clasificador debido a la elección arbitraria de la división de datos. Mediante el uso *de la validación cruzada* , podemos ajustar el clasificador en diferentes subconjuntos de trenes / pruebas de los datos y hacer un promedio de todos los resultados de precisión. La función `cross_val_score` ajusta un clasificador a los datos de entrada mediante validación cruzada. Puede tomar como entrada el número de diferentes divisiones (pliegues) que se utilizarán (5 en el ejemplo a continuación).

```
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)
```

Creando tuberías

La búsqueda de patrones en los datos a menudo se realiza en una cadena de pasos de procesamiento de datos, por ejemplo, selección de características, normalización y clasificación. En `sklearn` , una tubería de etapas se utiliza para esto.

Por ejemplo, el siguiente código muestra una tubería que consta de dos etapas. La primera escala las características y la segunda entrena un clasificador en el conjunto de datos aumentado resultante:

```
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 vez que se crea la tubería, puede usarla como una etapa regular (dependiendo de sus pasos específicos). Aquí, por ejemplo, la tubería se comporta como un clasificador. En consecuencia, podemos usarlo de la siguiente manera:

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

Interfaces y convenciones:

Diferentes operaciones con datos se realizan utilizando clases especiales.

La mayoría de las clases pertenecen a uno de los siguientes grupos:

- algoritmos de clasificación (derivados de `sklearn.base.ClassifierMixin`) para resolver problemas de clasificación
- algoritmos de regresión (derivados de `sklearn.base.RegressorMixin`) para resolver el problema de la reconstrucción de variables continuas (problema de regresión)
- Transformaciones de datos (derivadas de `sklearn.base.TransformerMixin`) que preprocesan los datos

Los datos se almacenan en `numpy.array` s (pero se `numpy.array` otros objetos similares a una `pandas.DataFrame` como `pandas.DataFrame` s si son convertibles a `numpy.array` s)

Cada objeto en los datos se describe mediante un conjunto de características. La convención general es que la muestra de datos se representa con una matriz, donde la primera dimensión es la identificación de la muestra de datos, la segunda dimensión es la identificación de la característica.

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

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

En `sklearn` convenciones de `sklearn` el conjunto de datos anterior contiene 5 objetos, cada uno descrito por 2 características.

Conjuntos de datos de muestra

Para facilitar la prueba, `sklearn` proporciona algunos conjuntos de datos `sklearn.datasets` en el módulo `sklearn.datasets` . Por ejemplo, carguemos datos de iris de Fisher:

```
import sklearn.datasets
```

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

Puede leer la descripción completa, los nombres de las características y los nombres de las clases (`target_names`). Esos se almacenan como cadenas.

Nos interesan los datos y las clases, que se almacenan en los `data` y `target` campos de `target` . Por convención, se denotan como `X` y `y`

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

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

Las formas de `X` e `y` dicen que hay 150 muestras con 4 características. Cada muestra pertenece a una de las siguientes clases: 0, 1 o 2.

Ahora se pueden usar `X` e `y` para entrenar a un clasificador, llamando al método `fit()` del clasificador.

Aquí está la lista completa de conjuntos de datos proporcionados por el módulo `sklearn.datasets` con su tamaño y uso previsto:

Cargar con	Descripción	tamaño	Uso
`load_boston()`	Conjunto de datos de precios de la vivienda de Boston	506	regresión
`load_breast_cancer()`	Conjunto de datos de Wisconsin sobre el cáncer de mama	569	clasificación (binario)
`load_diabetes()`	Conjunto de datos de la diabetes	442	regresión
`load_digits(n_class)`	Conjunto de datos de dígitos	1797	clasificación
`load_iris()`	Conjunto de datos de iris	150	clasificación (multi-clase)
`load_linnerud()`	Conjunto de datos linnerud	20	regresión multivariada

Tenga en cuenta que (fuente: http://scikit-learn.org/stable/datasets/) :

> Estos conjuntos de datos son útiles para ilustrar rápidamente el comportamiento de los diversos algoritmos implementados en el scikit. Sin embargo, a menudo son

demasiado pequeños para ser representativos de las tareas de aprendizaje automático del mundo real.

Además de estos conjuntos de datos de muestra de juguetes `sklearn.datasets` , `sklearn.datasets` también proporciona funciones de utilidad para cargar conjuntos de datos externos:

- `load_mlcomp` para cargar conjuntos de datos de muestra desde el repositorio mlcomp.org (tenga en cuenta que los conjuntos de datos deben descargarse antes). Aquí hay un ejemplo de uso.
- `fetch_lfw_pairs` y `fetch_lfw_people` para cargar el conjunto de datos de pares de `fetch_lfw_pairs` `fetch_lfw_people` the Wild (LFW) de http://vis-www.cs.umass.edu/lfw/ , que se utiliza para la verificación de la cara (resp. reconocimiento de la cara). Este conjunto de datos es más grande que 200 MB. Aquí hay un ejemplo de uso.

Lea Empezando con scikit-learn en línea: https://riptutorial.com/es/scikit-learn/topic/1035/empezando-con-scikit-learn

Capítulo 2: Característica de funcionamiento del receptor (ROC)

Examples

Introducción a ROC y AUC

Ejemplo de métrica de Característica operativa del receptor (ROC) para evaluar la calidad de salida del clasificador.

Las curvas ROC suelen presentar una tasa de verdaderos positivos en el eje Y, y una tasa de falsos positivos en el eje X. Esto significa que la esquina superior izquierda de la gráfica es el punto "ideal": una tasa de falsos positivos de cero y una verdadera tasa de positivos de uno. Esto no es muy realista, pero sí significa que un área más grande debajo de la curva (AUC) suele ser mejor.

La "inclinación" de las curvas ROC también es importante, ya que es ideal para maximizar la tasa de verdaderos positivos mientras minimiza la tasa de falsos positivos.

Un ejemplo simple:

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

Valores de `y` arbitrarios: en el caso real, estos son los valores de destino pronosticados (`model.predict(x_test)`):

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

Las puntuaciones son la precisión media en los datos de prueba y las etiquetas `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])
```

Calcula la curva ROC y la AUC:

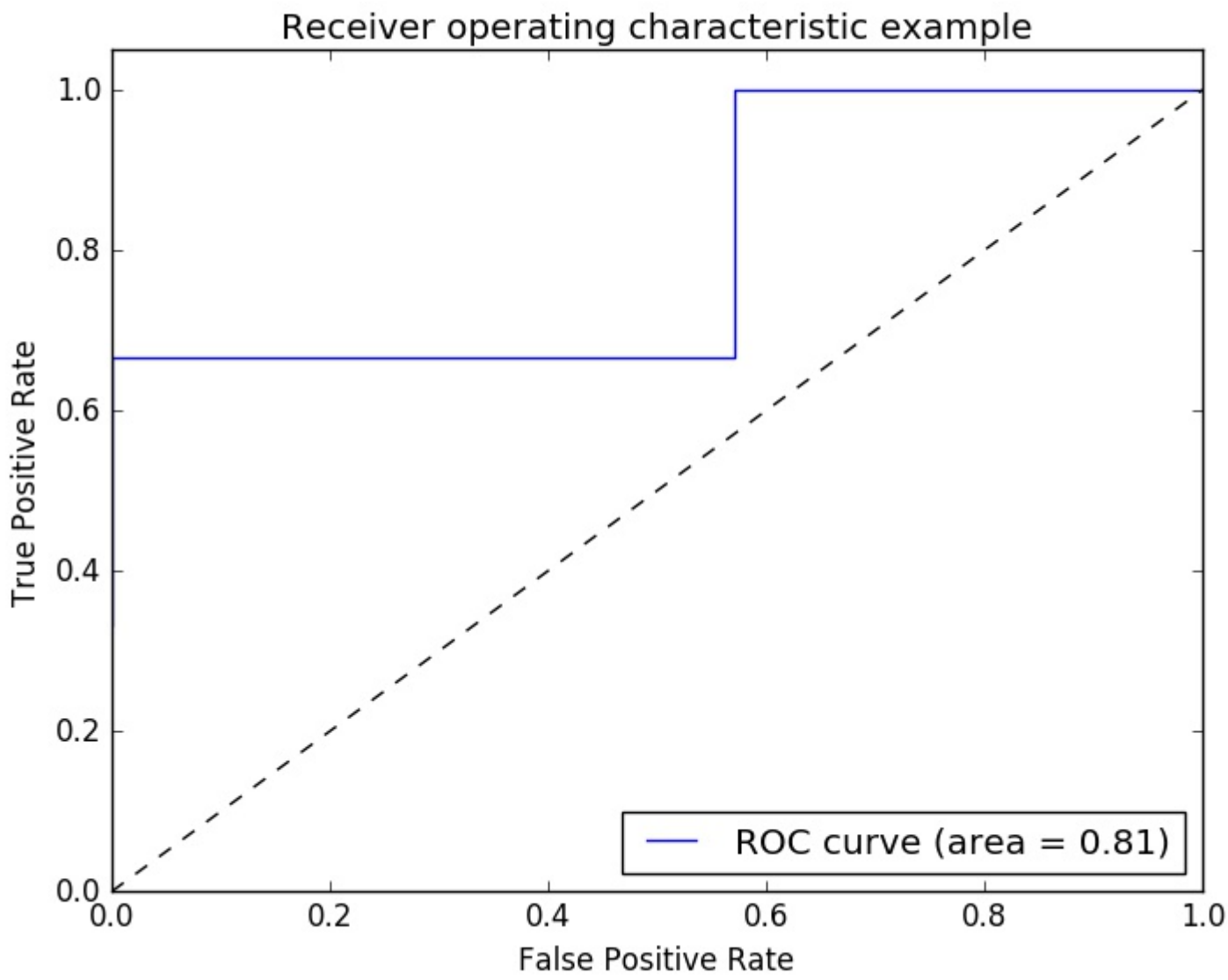
```
fpr, tpr, thresholds = metrics.roc_curve(y, scores, pos_label=2)
roc_auc = metrics.auc(fpr, tpr)
```

Trazar

```
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()
```

Salida:



Nota: las fuentes fueron tomadas de estos link1 y link2

Puntaje ROC-AUC con invalidación y validación cruzada

Uno necesita las probabilidades pronosticadas para calcular la puntuación ROC-AUC (área bajo la curva). El `cross_val_predict` utiliza los métodos de `predict` de los clasificadores. Para poder obtener el puntaje ROC-AUC, uno puede simplemente subclasificar el clasificador, anulando el método de `predict` , para que actúe como `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)))
```

salida:

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

Lea Característica de funcionamiento del receptor (ROC) en línea: https://riptutorial.com/es/scikit-learn/topic/5945/caracteristica-de-funcionamiento-del-receptor--roc-

Capítulo 3: Clasificación

Examples

Uso de máquinas de vectores de soporte

Las máquinas de vectores de soporte son una familia de algoritmos que intentan pasar un hiperplano (posiblemente de alta dimensión) entre dos conjuntos de puntos etiquetados, de modo que la distancia de los puntos desde el plano sea óptima en algún sentido. Las SVM se pueden usar para clasificación o regresión (correspondientes a `sklearn.svm.SVC` y `sklearn.svm.SVR` , respectivamente).

Ejemplo:

Supongamos que trabajamos en un espacio 2D. Primero, creamos algunos datos:

```
import numpy as np
```

Ahora creamos *x* y *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
```

Tenga en cuenta que *x* se compone de dos gaussianos: uno centrado alrededor *(0, 0)* y otro centrado alrededor *(1, 1)* .

Para construir un clasificador, podemos utilizar:

```
from sklearn import svm

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

Veamos la predicción para *(0, 0)* :

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

La predicción es que la clase es 0.

Para la regresión, igualmente podemos hacer:

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

RandomForestClassifier

Un bosque aleatorio es un meta estimador que se ajusta a una serie de clasificadores de árboles de decisión en varias submuestras del conjunto de datos y utiliza el promedio para mejorar la precisión predictiva y el ajuste excesivo del control.

Un ejemplo de uso simple:

Importar:

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

Definir los datos del tren y los datos de destino:

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

Los valores en `target` representan la etiqueta que desea predecir.

Inicie un objeto RandomForest y realice el aprendizaje (ajuste):

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

Predecir:

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

Análisis de informes de clasificación

Cree un informe de texto que muestre las principales métricas de clasificación, incluidas la precisión y la recuperación , la puntuación f1 (la media armónica de la precisión y la recuperación) y el soporte (el número de observaciones de esa clase en el conjunto de entrenamiento).

Ejemplo de documentos de `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))
```

Salida -

```
             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

Gradient Boosting para la clasificación. El clasificador de aumento de gradiente es un conjunto aditivo de un modelo base cuyo error se corrige en iteraciones (o etapas) sucesivas mediante la adición de árboles de regresión que corrigen los residuos (el error de la etapa anterior).

Importar:

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

Crear algunos datos de clasificación de juguetes

```
from sklearn.datasets import load_iris

iris_dataset = load_iris()

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

Vamos a dividir estos datos en conjunto de entrenamiento y pruebas.

```
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)
```

Cree una instancia de un modelo `GradientBoostingClassifier` utilizando los parámetros predeterminados.

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

Vamos a anotarlo en el conjunto de prueba

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

Por defecto hay 100 estimadores construidos.

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

Esto se puede controlar estableciendo `n_estimators` en un valor diferente durante el tiempo de inicialización.

Un árbol de decisión

Un árbol de decisión es un clasificador que usa una secuencia de reglas detalladas (como a> 7) que se pueden entender fácilmente.

El siguiente ejemplo entrena a un clasificador de árbol de decisión usando tres vectores de características de longitud 3, y luego predice el resultado para un cuarto vector de características hasta ahora desconocido, el llamado vector de prueba.

```
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
```

La salida se puede visualizar 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")
```

Clasificación mediante regresión logística

En el clasificador LR, las probabilidades que describen los posibles resultados de un solo ensayo se modelan utilizando una función logística. Se implementa en la librería `linear_model`

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

La implementación de Sklearn LR puede ajustarse a regresión logística binaria, One-vs-Rest o multinomial con la regularización opcional de L2 o L1. Por ejemplo, consideremos una clasificación binaria en un conjunto de datos de sklearn de muestra

```
from sklearn.datasets import make_hastie_10_2

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

Donde X es una matriz de `n_samples X 10` y y son las etiquetas de destino -1 o +1.

Utilice la división de prueba de tren para dividir los datos de entrada en conjuntos de prueba y entrenamiento (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)
```

Usar el clasificador LR es similar a otros ejemplos

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

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

Usa la matriz de confusión para visualizar resultados.

```
from sklearn.metrics import confusion_matrix

confusion_matrix(predicted, labels_test)
```

Lea Clasificación en línea: https://riptutorial.com/es/scikit-learn/topic/2468/clasificacion

Capítulo 4: Reducción de la dimensionalidad (selección de características)

Examples

Reduciendo la dimensión con el análisis de componentes principales

El análisis de componentes principales encuentra secuencias de combinaciones lineales de las características. La primera combinación lineal maximiza la varianza de las características (sujeto a una restricción de unidad). Cada una de las siguientes combinaciones lineales maximiza la varianza de las características en el subespacio ortogonal a la que abarca las combinaciones lineales anteriores.

Una técnica de reducción de dimensión común es usar solo las *k* primeras de tales combinaciones lineales. Supongamos que las entidades son una matriz *X* de *n* filas y *m* columnas. Las primeras *k* combinaciones lineales forman una matriz β_k de *m* filas *yk* columnas. El producto *X* β tiene *n* filas *yk* columnas. Por lo tanto, la matriz β *k* resultante puede considerarse una reducción de las dimensiones de *m* a *k* , reteniendo las partes de alta varianza de la matriz *X* original.

En `scikit-learn` , PCA se realiza con `sklearn.decomposition.PCA` . Por ejemplo, supongamos que comenzamos con una matriz de 100 X 7, construida de modo que la varianza esté contenida solo en las dos primeras columnas (reduciendo las cinco últimas columnas):

```
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)))
```

Realicemos una reducción a 2 dimensiones:

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

Ahora vamos a ver los resultados. Primero, aquí están las combinaciones lineales:

```
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]])
```

Observe que los dos primeros componentes de cada vector son varios órdenes de magnitud más grandes que los otros, lo que demuestra que la PCA reconoció que la varianza está contenida

principalmente en las dos primeras columnas.

Para verificar la relación de la varianza explicada por este PCA, podemos examinar `pca.explained_variance_ratio_` :

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

Lea Reducción de la dimensionalidad (selección de características) en línea: https://riptutorial.com/es/scikit-learn/topic/4829/reduccion-de-la-dimensionalidad--seleccion-de-caracteristicas-

Capítulo 5: Regresión

Examples

Mínimos cuadrados ordinarios

Ordinary Least Squares es un método para encontrar la combinación lineal de características que mejor se ajuste al resultado observado en el siguiente sentido.

Si el vector de resultados a predecir es y , y las variables explicativas forman la matriz X , entonces OLS encontrará el vector β resolviendo

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

donde $y\hat{} = X\beta$ es la predicción lineal.

En sklearn, esto se hace usando `sklearn.linear_model.LinearRegression` .

Contexto de aplicación

OLS solo debe aplicarse a problemas de regresión, generalmente no es adecuado para problemas de clasificación: Contraste

- ¿Es un correo electrónico spam? (Clasificación)
- ¿Cuál es la relación lineal entre upvotes depende de la longitud de la respuesta? (Regresión)

Ejemplo

`LinearRegression` un modelo lineal con algo de ruido, luego veamos si `LinearRegression` arregla para reconstruir el modelo lineal.

Primero generamos la matriz `X` :

```
import numpy as np

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

Ahora generaremos la `y` como una combinación lineal de `X` con algo de ruido:

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

Tenga en cuenta que la verdadera combinación lineal que genera `y` está dada por `beta.

Para tratar de reconstruir esto a partir de `X` e `y` solo, hagamos:

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

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

Tenga en cuenta que este vector es muy similar a la `beta` .

Lea Regresión en línea: https://riptutorial.com/es/scikit-learn/topic/5190/regresion

Capítulo 6: Selección de características

Examples

Eliminación de características de baja variación

Esta es una técnica de selección de características muy básica.

Su idea subyacente es que si una característica es constante (es decir, tiene una variación de 0), no se puede usar para encontrar patrones interesantes y se puede eliminar del conjunto de datos.

En consecuencia, un enfoque heurístico para la eliminación de características es eliminar primero todas las características cuya varianza esté por debajo de algún umbral (bajo).

Partiendo del ejemplo en la documentación , supongamos que empezamos con

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

Hay *3* características booleanas aquí, cada una con *6* instancias. Supongamos que deseamos eliminar aquellos que son constantes en al menos el 80% de las instancias. Algunos cálculos de probabilidad muestran que estas características deberán tener una varianza inferior a *0.8 * (1 - 0.8)* . En consecuencia, podemos utilizar

```
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]])
```

Observe cómo se eliminó la primera característica.

Este método se debe usar con precaución porque una variación baja no significa necesariamente que una característica no sea interesante. Considere el siguiente ejemplo en el que construimos un conjunto de datos que contiene *3* características, las dos primeras consisten en variables distribuidas al azar y la tercera en variables distribuidas uniformemente.

```
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 ]])
```

Ahora, la primera característica se ha eliminado debido a su baja variación, mientras que la tercera característica (que es la más interesante) se ha mantenido. En este caso, hubiera sido más apropiado considerar un *coeficiente de variación* porque eso es independiente de la escala.

Lea Selección de características en línea: https://riptutorial.com/es/scikit-learn/topic/4909/seleccion-de-caracteristicas

Capítulo 7: Selección de modelo

Examples

Validación cruzada

Aprender los parámetros de una función de predicción y probarla con los mismos datos es un error metodológico: un modelo que simplemente repetiría las etiquetas de las muestras que acaba de ver tendría una puntuación perfecta pero no podría predecir nada útil aún. datos invisibles Esta situación se llama **sobreajuste** . Para evitarlo, es una práctica común cuando se realiza un experimento de aprendizaje automático (supervisado) para mantener parte de los datos disponibles como un **conjunto de prueba** `X_test, y_test` . Tenga en cuenta que la palabra "experimento" no pretende indicar solo el uso académico, porque incluso en entornos comerciales, el aprendizaje automático generalmente comienza de manera experimental.

En scikit-learn, una división aleatoria en entrenamiento y conjuntos de pruebas se puede calcular rápidamente con la función de ayuda train_test_split . Carguemos el conjunto de datos del iris para que se ajuste a una máquina de vectores de soporte lineal:

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

Ahora podemos muestrear rápidamente un conjunto de entrenamiento mientras tenemos el 40% de los datos para probar (evaluar) nuestro clasificador:

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

Ahora, después de que tengamos conjuntos de trenes y pruebas, utilicemos:

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

Validación cruzada K-Fold

La validación cruzada de K-fold es un proceso sistemático para repetir el procedimiento de división de tren / prueba varias veces, con el fin de reducir la varianza asociada con un ensayo

único de división de tren / prueba. Básicamente, se divide todo el conjunto de datos en K "pliegues" de igual tamaño, y cada pliegue se usa una vez para probar el modelo y K-1 veces para entrenar el modelo.

Múltiples técnicas de plegado están disponibles con la biblioteca de scikit. Su uso depende de las características de los datos de entrada. Algunos ejemplos son

K-Fold

Básicamente, se divide todo el conjunto de datos en K "pliegues" de igual tamaño, y cada pliegue se usa una vez para probar el modelo y K-1 veces para entrenar el modelo.

```
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` es una variación de k-fold que devuelve pliegues estratificados: cada conjunto contiene aproximadamente el mismo porcentaje de muestras de cada clase objetivo que el conjunto completo

ShuffleSplit

Se utiliza para generar un número definido por el usuario de divisiones independientes de conjuntos de datos de prueba / tren. Las muestras se barajan primero y luego se dividen en un par de conjuntos de prueba y tren.

```
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` es una variación de ShuffleSplit, que devuelve divisiones estratificadas, es decir, que crea divisiones al preservar el mismo porcentaje para cada clase objetivo que en el conjunto completo.

Otras técnicas de plegado como Leave One / p Out, y TimeSeriesSplit (una variación de K-fold) están disponibles en la biblioteca scikit model_selection.

Lea Selección de modelo en línea: https://riptutorial.com/es/scikit-learn/topic/4901/seleccion-de-modelo

Creditos

S. No	Capítulos	Contributors
1	Empezando con scikit-learn	Alleo, Ami Tavory, Community, Gabe, Gal Dreiman, panty, Sean Easter, user2314737
2	Característica de funcionamiento del receptor (ROC)	Gal Dreiman, Gorkem Ozkaya
3	Clasificación	Ami Tavory, Drew, Gal Dreiman, hashcode55, Mechanic, Raghav RV, Sean Easter, tfv, user6903745, Wayne Werner
4	Reducción de la dimensionalidad (selección de características)	Ami Tavory, DataSwede, Gal Dreiman, Sean Easter, user2314737
5	Regresión	Ami Tavory, draco_alpine
6	Selección de características	Ami Tavory, user2314737
7	Selección de modelo	Gal Dreiman, Mechanic