بسمی تعالی

موضوع: اموزش کتابخانه Scikit-Learn در پایتون

Scikit Learn از کتابخانه‌های متن‌باز، مفید، پرکاربرد و قدرتمند در زبان برنامه‌نویسی پایتون است که برای اهداف

یادگیری ماشین به کار می‌رود. این کتابخانه ابزارهای کاربردی زیادی به منظور یادگیری ماشین و مدل‌سازی آماری

داده‌ها همچون طبقه‌بندی (classification)، رگرسیون، خوشه‌بندی و کاهش ابعاد فراهم می‌کند. این کتابخانه که به طور

عمده توسط زبان پایتون ارائه شده، بر پایه‌ی کتابخانه‌های Numpy ،Scipy و Matplotlib طراحی شده است.

**نصب Scikit-Learn**

به منظور پیاده‌سازی الگوریتم‌ها و کدها در Scikit-Learn نیاز است تا موارد زیر را بر روی کامپیوتر خود نصب داشته

باشید:

* Python (>=3.5)
* NumPy (>= 1.11.0)
* Scipy (>= 0.17.0)
* Joblib (>= 0.11)
* :Matplotlib به منظور افزودن قابلیت رسم نمودار و اشکال.
* :Pandas (>= 0.18.0) جهت استفاده در مثال‌ها و کدهایی که از ساختار داده‌ی پانداس بهره می‌برند.

اگر Numpy و Scipy را نصب کرده‌اید، دو روش زیر آسان‌ترین راه برای نصب Scikit-Learn هستند:

* استفاده از دستور Pip:

در محیط خط فرمان Command به شکل زیر، Scikit را نصب کنید:

pip install -U scikit-learn

* استفاده از Conda:

به کمک دستور زیر می‌توان Scikit-Learn را در Conda نصب کرد:

conda install scikit-learn

البته شما می‌توانید توزیع‌های پایتونی مثل آناکوندا را نصب کنید و از آنجایی که پکیج‌هایی همچون Numpy، Scipy و

Scikit-Learn در این توزیع موجود هستند، کدهای خود را اجرا نمایید.

**ویژگی‌های Scikit-Learn**

علاوه بر کاربرد این کتابخانه در مواردی مانند بارگذاری، تغییر و دستکاری داده‌ها، Scikit-Learn در مدل‌سازی داده،

تمرکز ویژه‌ای دارد. برخی از مدل‌سازی‌های مشهور در این کتابخانه عبارتند از:

* الگوریتم‌های یادگیری نظارت شده (Superuised Learning)

تقریبا تمامی الگویتم‌های یادگیری نظارت شده معروف رگرسیون خطی، ماشین بردار پشتیبان (SUM)، درخت

تصمیم‌گیری و... در این کتابخانه موجود هستند.

* الگوریتم‌های یادگیری نظارت نشده (Unsuperuised Learning)

تمامی الگوریتم‌های نظارت نشده همچون روش‌های خوشه‌بندی، تحلیل فاکتوری، PCA، شبکه‌های عصبی نظارت نشده

و... بخشی از کتابخانه‌ی Superuised Learning می‌باشند.

* اعتبارسنجی یا Cross-Validation

به منظور کنترل دقت مدل نظارتی بر روی داده‌های تست، مورد استفاده قرار می‌گیرد.

* کاهش ابعاد و داده‌ها

به منظور کاهش ویژگی‌های داده‌ها جهت خلاصه‌سازی، مصورسازی و نیز انتخاب ویژگی مورد استفاده قرار می‌گیرد.

* کاهش ابعاد داده‌ها

به منظور کاهش ویژگی‌های داده‌ها جهت خلاصه‌سازی، مصورسازی و نیز انتخاب ویژگی مورد استفاده قرار می‌گیرد.

* روش‌های گروه‌بندی Ensemble Methods

همان‌طور که از نامش پیداست، این روش برای ترکیب مدل‌های یادگیری نظارت شده، به منظور پیش‌بینی برچسب

داده‌های تست به کار می‌رود.

* استخراج ویژگی Feature extraction

به منظور تعریف ویژگی‌های جدید از روی ویژگی‌های اصلی جهت استخراج داده‌های مفید به کار می‌رود.

* انتخاب ویژگی یا Feature selection

روشی جهت انتخاب ویژگی برای ایجاد مدل‌هایی با دقت بالاتر یا افزونگی کمتر است.

**فرایند مدل‌سازی در Scikit-Learn**

در این قسمت به شرح فرایند مدل‌سازی در Scikit-Learn می‌پردازیم. مدل‌سازی در Scikit-Learn شامل مراحل

بارگذاری داده، تقسیم داده، آموزش داده و تست آن است. در ادامه با جزییات بیشتری به شرح هر کدام از این مراحل

می‌پردازیم.

**بارگذاری داده Dataset Loading**

ویژگی متغیرهای مربوط به داده‌ها، feature یا ویژگی نامیده می‌شوند. گاه به ویژگی، پیش‌بینی‌کننده (predictor)،

ورودی (input) یا صفت (attribute) نیز می‌گویند.

ماتریس ویژگی: مجموعه‌ای از ویژگی‌ها تحت شرایطی است که بیش از یک ویژگی دارد.

نام ویژگی: لیستی از نام ویژگی‌هاست.

پاسخ یا ویژگی برچسب: متغیر خروجی که به متغیرهای ویژگی وابسته است. که به آن هدف (target)، برچسب

(label) یا خروجی (output) می‌گویند.

بردار پاسخ (Response Vector): بیانگر ستون پاسخ است. به طور کلی ما یک ستون پاسخ داریم.

نام‌های هدف: Target Name

بیانگر مقادیری است که ستون پاسخ می‌تواند داشته باشد.

Scikit-Learn مجموعه داده‌های نمونه‌ی محدودی همچون Iris و digit برای طبقه‌بندی (Classification) و قیمت خانه

در Boston برای رگرسیون دارد.

در زیر مثالی برای بارگذاری مجموعه داده‌ی iris داریم:

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

target\_names = iris.target\_names

print("Feature names:", feature\_names)

print("Target names:", target\_names)

print("\nFirst 10 rows of X:\n", X[:10])

که خروجی آن به شکل زیر است:

Feature names: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

Target names: ['setosa' 'versicolor' 'virginica']

First 10 rows of X:

[

[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3]

[5. 3.4 1.5 0.2]

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

]

**تقسیم مجموعه داده (Splitting the data set)**

برای کنترل دقت مدل استخراج‌شده، می‌توان مجموعه داده را به دو قسمت تقسیم کرد: مجموعه‌ی آموزشی

(training set) و مجموعه‌ی آزمایشی یا تست (testing set).

از داده‌ی آموزشی برای آموزش روش یادگیری ماشین و استخراج مدل آموزشی استفاده می‌شود. مدل استخراجی حاصل

روی مجموعه‌ی تست اعمال می‌شود تا دقت مدل بررسی شود.

در کد زیر داده را به نسبت 70 به 30 تقسیم کرده‌ایم. یعنی 70 درصد داده‌ها، داده‌ی آموزشی در نظر گرفته شده‌اند و

30 درصد به داده‌های تست اختصاص یافته‌اند.

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size = 0.3, random\_state = 1

)

print(X\_train.shape)

print(X\_test.shape)

print(y\_train.shape)

print(y\_test.shape)

که خروجی به شکل زیر است:

(105, 4)

(45, 4)

(105,)

(45,)

همان طور که در کد بالا می‌بینید تابع ()train\_test\_split برای تقسیم مجموعه داده به کار گرفته می‌شود. آرگومان‌های

این تابع به شرح زیر هستند:

* X ماتریس ویژگی و Y بردار پاسخ است.
* Test\_size بیانگر نسبت داده‌های تست به کل مجموعه داده است. در مثال بالا، 0.3 از 150 سطر موجود برای
* X را به عنوان داده‌ی تست در نظر گرفتیم. پس 45 سطر X متعلق به داده‌ی تست و بقیه متعلق به داده‌های
* آموزشی است.
* :Random\_state  اگر مقدار این پارامتر را برابر هر عددی صحیحی در نظر بگیریم (فرقی نمی‌کند 1، 4، 42....) در آن صورت در هر مرحله‌ای که الگوریتم برای تولید مدل تکرار می‌شود، داده‌ها به نسبت مساوی با مرحله‌ی قبل به داده‌های آموزشی و تست تقسیم می‌شوند. به طور مثال در هرمرحله به طور تصادفی 0.7 داده‌ها به داده‌ی آموزشی و 0.3 آن به داده‌ی تست اختصاص می‌یابد.

**آموزش مدل**

بعد از تقسیم داده‌ها، می‌توانیم داده‌ها را برای یادگیری مدل‌های پیش‌بینی به کار بگیریم.

همان طور که گفته شد، Scikit-Learn الگوریتم‌های یادگیری ماشین گسترده‌ای دارد که رابط یا اینترفیس ثابتی برای

مدل کردن، پیش‌بینی دقت و فراخوانی برای تمام الگوریتم‌ها ارائه می‌دهد.

مثال زیر نحوه‌ی آموزش طبقه‌بندی KNN )نزدیکترین همسایه k( را توسط Scikit نمایش می‌دهد. )جزییات الگوریتم

KNN در بخش‌های بعدی شرح داده شده است(

from sklearn.datasets import load\_iris

iris = load\_iris()

X = iris.data

y = iris.target

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size = 0.4, random\_state=1

)

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

classifier\_knn = KNeighborsClassifier(n\_neighbors = 3)

classifier\_knn.fit(X\_train, y\_train)

y\_pred = classifier\_knn.predict(X\_test)

*# Finding accuracy by comparing actual response values(y\_test)with predicted response value(y\_pred)*

print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))

*# Providing sample data and the model will make prediction out of that data*

sample = [[5, 5, 3, 2], [2, 4, 3, 5]]

preds = classifier\_knn.predict(sample)

pred\_species = [iris.target\_names[p] for p in preds] print("Predictions:", pred\_species)

خروجی کد به شکل زیر است:

Accuracy: 0.9833333333333333

Predictions: ['versicolor', 'virginica']

**پایداری مدل: Model Persistence**

وقتی با اعمال الگوریتمی بر روی مجموعه داده‌های آموزشی مدلی را استخراج می‌کنیم، انتظار داریم این مدل برای

استفاده‌های آتی نیز پایدار بماند. از این رو مدل را چندین مرتبه بازآموزی (retrain) می‌کنیم. این کار را می‌توان به

کمک ویژگی‌های dump و load از پکیج joblib انجام داد. مثال زیر را در نظر بگیرید که در آن مدل یادگرفته شده از

مثال قبل (classifier\_knn) را برای استفاده‌های آتی ذخیره می‌کنیم:

from sklearn.externals import joblib

joblib.dump(classifier\_knn, 'iris\_classifier\_knn.joblib')

قطعه کد بالا، مدل را در فایلی تحت عنوان iris\_classifier\_knn.joblib ذخیره می‌کند.

حال می‌توان آبجکت ایجاد شده را با کمک دستور زیر دوباره بارگذاری کرد.

joblib.load('iris\_classifier\_knn.joblib')

**پیش‌پردازش داده Preprocessing the Data:**

از آنجایی که در فرم خام داده‌ها با مقادیر زیادی داده سر و کار داریم، باید قبل از به‌کارگیری داده به عنوان ورودی

الگوریتم یادگیری ماشین، آن‌ها را آماده‌سازی کنیم و به داده‌هایی مناسب و معنی‌دار تبدیل کنیم. این فرایند، پیش‌پردازش

داده‌ها نامیده می‌شود.

Scikit-Learn از پکیج preprocessing برای این منظور استفاده می‌کند. پکیج sklearn.preprocessing چندین تابع

و کلاس تبدیل کننده‌ی رایج برای تبدیل بردار ویژگی و داده‌ی خام به داده‌های قابل استفاده توسط تخمین‌گرها را فراهم

می‌کند. به طور کلی، الگوریتم‌های یادگیری با استفاده از استانداردسازی داده‌ها، می‌توانند عملکرد بهتری داشته باشند.

اگر برخی داده‌های پرت در مجموعه داده وجود داشته باشد، مقیاس‌دهی داده‌ها (scaling) و استفاده از توابع تبدیل کننده‌ی

قوی‌تر مناسب است.

برای الگوریتم‌های تخمین‌گر تعریف شده در مجموعه‌ی scikit-learn استانداردسازی مجموعه داده‌، عمل رایج و مورد

نیاز محسوب می‌شود. اگر مقادیر ویژگی‌های ورودی الگوریتم تخمین‌گر شباهت زیادی به توزیع نرمال استاندارد داده

(منظور از توزیع نرمال: داده‌هایی با توزیع گوسی با میانگین صفر و واریانس واحد) نداشته باشد، باعث می‌شود که

تخمین‌گر رفتار خوب و خروجی مناسبی روی داده‌ها نداشته باشد.

در عمل، اکثراً شکل و نحوه‌ی توزیع داده‌ها را در نظر نمی‌گیریم و  ابتدا مقدار میانگین داده‌ها را  از آن‌ها کم می‌کنیم و

سپس حاصل را بر انحراف معیار تقسیم کرده و به این صورت داده‌ها را، متمرکز یا centralized می‌کنیم. به عنوان

مثال بسیاری از عناصر مورد استفاده در تابع هدف الگوریتم یادگیری ماشین )مانند RBF Kernel در (SVMفرض

می‌کنند که  مرکز تمام ویژگی‌ها حول صفر است و دارای پراکندگی و واریانس یکسان‌اند. اگر ویژگی دارای واریانس

بزرگتر از سایرین باشد، ممکن است بر عملکرد تابع هدف مسلط شود و سبب شود تا تخمین‌گر به درستی از مقادیر سایر

ویژگی‌ها برای یادگیری خود استفاده نکند.

این پکیج تکنیک‌های زیر را شامل می‌شود:

**1. باینری کردن**

این تکنیک پیش‌پردازش زمانی به کار می‌رود که می‌خواهیم مقادیر عددی را به مقادیر بولین (منطقی) تبدیل کنیم.

import numpy as np

from sklearn import preprocessing

Input\_data = np.array(

[2.1, -1.9, 5.5],

[-1.5, 2.4, 3.5],

[0.5, -7.9, 5.6],

[5.9, 2.3, -5.8]]

)

data\_binarized = preprocessing.Binarizer(threshold=0.5).transform(input\_data)

print("\nBinarized data:\n", data\_binarized)

پارامتر threshhold\_value=0.5 برای این است که داده‌هایی با مقدار بزرگتر از 0.5 به 1 و مقدار کوچکتر یا مساوی

0.5 به صفر تبدیل شوند.

Binarized data:

[

[ 1. 0. 1.]

[ 0. 1. 1.]

[ 0. 0. 1.]

[ 1. 1. 0.]

]

**2.  حذف میانگین یا Mean Removal**

این روش برای حذف میانگین داده‌ها از بردار ویژگی است که منجر به نرمال‌سازی ویژگی‌ها با محدودیت صفر می‌شود.

import numpy as np

from sklearn import preprocessing

Input\_data = np.array(

[2.1, -1.9, 5.5],

[-1.5, 2.4, 3.5],

[0.5, -7.9, 5.6],

[5.9, 2.3, -5.8]]

)

*#displaying the mean and the standard deviation of the input data*

print("Mean =", input\_data.mean(axis=0))

print("Stddeviation = ", input\_data.std(axis=0))

*#Removing the mean and the standard deviation of the input data*

data\_scaled = preprocessing.scale(input\_data)

print("Mean\_removed =", data\_scaled.mean(axis=0))

print("Stddeviation\_removed =", data\_scaled.std(axis=0))

خروجی کد بالا به شکل زیر است:

Mean = [ 1.75 -1.275 2.2 ]

Stddeviation = [ 2.71431391 4.20022321 4.69414529]

Mean\_removed = [ 1.11022302e-16 0.00000000e+00 0.00000000e+00]

Stddeviation\_removed = [ 1. 1. 1.]

**3. مقیاس‌دهی یا scaling**

تابع scale روشی سریع و راحت برای انجام عمل مقیاس‌دهی به کار می‌برد. به کد زیر توجه کنید:

from sklearn import preprocessing

import numpy as np

X\_train = np.array([[ 1., -1., 2.],

[ 2., 0., 0.],

[ 0., 1., -1.]])

X\_scaled = preprocessing.scale(X\_train)

X\_scaled

خروجی کد به شکل زیر است:

array([[ 0. ..., -1.22..., 1.33...],

[ 1.22..., 0. ..., -0.26...],

[-1.22..., 1.22..., -1.06...]])

داده‌ی مقیاس‌بندی شده‌ی حاصل، میانگین صفر و واریانس واحد دارد:

>>> X\_scaled.mean(axis=0)

array([0., 0., 0.])

>>> X\_scaled.std(axis=0)

array([1., 1., 1.])

ماژول preprocessing کلاسی تحت عنوان stanardscaler دارد که حذف میانگین و تقسیم واریانس را بر روی

داده‌های آموزشی اجرا می‌کند و توسط transformer API خود امکان اعمال حذف همین مقدار میانگین و تقسیم همان

واریانس محاسبه شده‌ی حاصل از داده‌های آموزشی را بر روی داده‌های تست فراهم می‌کند.

>>> scaler = preprocessing.StandardScaler().fit(X\_train)

>>> scaler

StandardScaler()

>>> scaler.mean\_

array([1. ..., 0. ..., 0.33...])

>>> scaler.scale\_

array([0.81..., 0.81..., 1.24...])

>>> scaler.transform(X\_train)

array([[ 0. ..., -1.22..., 1.33...],

[ 1.22..., 0. ..., -0.26...],

[-1.22..., 1.22..., -1.06...]])

در زیر همین تبدیل را روی داده‌های تست اعمال می‌کنیم:

>>> X\_test = [[-1., 1., 0.]]

>>> scaler.transform(X\_test)

array([[-2.44..., 1.22..., -0.26...]])

اگر به هر دلیلی بخواهیم در کلاس standardscaler  امکان حذف میانگین و یا تقسیم شدن داده به انحراف معیار را حذف کنیم، به ترتیب می‌توانیم تنظیمات را به صورت with\_mean=False و with\_std=False اعمال کنیم.

یک روش استاندارد جایگزین مقیاس‌دهی به ویژگی‌ها، مقداردهی به آن‌ها در بازه‌ی حداقلی و حداکثری (معمولاً بین 0 و 1) است. روش دیگر، تعیین حداکثر مقدار مطلق یک ویژگی به اندازه‌ی واحد است.

**4. نرمال‌سازی**

نرمال‌سازی بردار ویژگی برای این که ویژگی‌ها در مقیاس یکسانی قرار گیرند به کار می‌آید. دو روش نرمال‌سازی داریم:

* **نرمال‌سازی L1**

این روش که به آن حداقل انحراف مطلق نیز می‌گویند، مقادیر ویژگی‌ها را به گونه‌ای تغییر می‌دهد که مجموع مقادیر مطلق در هر سطر حداکثر 1 باقی بماند.

در مثال زیر اجرای نرمال‌سازی L1 را روی داده‌های ورودی می‌بینیم:

import numpy as np

from sklearn import preprocessing

Input\_data = np.array(

[

[2.1, -1.9, 5.5],

[-1.5, 2.4, 3.5],

[0.5, -7.9, 5.6],

[5.9, 2.3, -5.8]

]

)

data\_normalized\_l1 = preprocessing.normalize(input\_data, norm='l1')

print("\nL1 normalized data:\n", data\_normalized\_l1)

خروجی حاصل به شکل زیر است:

L1 normalized data:

[

[ 0.22105263 -0.2 0.57894737]

[-0.2027027 0.32432432 0.47297297]

[ 0.03571429 -0.56428571 0.4 ]

[ 0.42142857 0.16428571 -0.41428571]

]

* **نرمال‌سازی L2**

این روش که حداقل مربعات نیز نامیده می‌شود، مقادیر را به گونه‌ای تغییر می‌دهد که مجموع مربعات در هر سطر حداکثر یک باقی بماند. به مثال زیر توجه کنید:

import numpy as np

from sklearn import preprocessing

Input\_data = np.array(

[

[2.1, -1.9, 5.5],

[-1.5, 2.4, 3.5],

[0.5, -7.9, 5.6],

[5.9, 2.3, -5.8]

]

)

data\_normalized\_l2 = preprocessing.normalize(input\_data, norm='l2')

print("\nL1 normalized data:\n", data\_normalized\_l2)

خروجی حاصل به شکل زیر است:

L2 normalized data:

[

[ 0.33946114 -0.30713151 0.88906489]

[-0.33325106 0.53320169 0.7775858 ]

[ 0.05156558 -0.81473612 0.57753446]

[ 0.68706914 0.26784051 -0.6754239 ]

]

**پیاده‌سازی  SVM در Scikit-Learn**

ماشین بردار پشتیبان (Support Vector Machine (SVM از جمله متدهای قدرتمند طبقه‌بندی نظارت شده است

. SVM در ابعاد بالا انعطاف‌پذیر است و در مسائل طبقه‌بندی به طور گسترده استفاده می‌شود. SVM از نظر مصرف

حافظه کاراست؛ زیرا زیرمجموعه‌ای از نقاط آموزشی را برای توابع تصمیم‌گیری به کار می‌گیرد. مهم‌ترین هدف

SVM تقسیم مجموعه داده به تعدادی از کلاس‌ها به منظور به دست آوردن ابرصفحه با بیشترین حاشیه

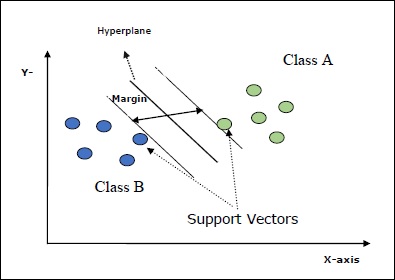
(Maximum Marginal Hyperplane: MMH) است که در دو مرحله صورت می‌گیرد.

* ماشین بردار پشتیبان در ابتدا ابرصفحاتی را تولید می‌کند که نقاط را به شکل صحیح تقسیم می‌کند.
* سپس میان ابرصفحات آن ابرصفحه‌ای را برمی‌گزیند که نقاط را به بهترین شکل ممکن جدا می‌کند.

برخی ویژگی‌های مهم در خصوص SVM به شرح زیر است:

* بردارهای پشتیبان (Support Vector): نقاطی هستند که به ابرصفحه نزدیک‌ترند. بردارهای پشتیبان در شناسایی خطوط جداساز کمک کننده‌ هستند.
* ابرصفحه (Hyper Plane): صفحه‌ای که نقاط را به کلاس‌‌های مختلف تقسیم می‌کند.
* حاشیه (Margin): فاصله‌ی خالی میان نقاط مرزی دو کلاس یا همان بردارهای پشتیبان، را حاشیه‌ یا لبه یا همان Margin گویند.

با توجه به شکل زیر درک بهتری از مفاهیم و تعاریف ارائه شده در بالا را به دست می‌آورید:



SVM در Scikit-Learn از هر دو نوع ورودی نقاط به صورت متراکم و پراکنده پشتیبانی می‌کند.

**طبقه‌بندی با SVM**

Scikit-learn سه کلاس با نام‌های SVC، NuSVC و LinearSVC دارد که می‌توانند طبقه‌بندی چند کلاسه

(multivlass\_class) را انجام دهند. برای نمونه طبقه‌بندی از طریق SVC را بررسی می‌کنیم.

SVC طبقه‌بندی چند کلاسه را از طریق مقایسه و طبقه‌بندی به صورت یک به یک (یک برچسب در مقایسه با سایر

نمونه‌ها) انجام می‌دهد.

به مثال زیر توجه کنید:

import numpy as np

X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])

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

from sklearn.svm import SVC

SVCClf = SVC(kernel = 'linear',gamma = 'scale', shrinking = False,)

SVCClf.fit(X, y)

همانند سایر طبقه‌بندی‌ها، SVC با دو آرایه مدل می‌شود:

* یک آرایه‌ی X که نگهدارنده‌ی نمونه‌های آموزشی است و در اندازه‌ی [n\_samples, n\_features] است.
* یک آرایه‌ی Y که مقادیر هدف یا target را نگهداری می‌کند. کلاس برچسب برای نمونه‌های آزمایشی است و در
* اندازه‌ی [n\_samples] است.

توضیح پارامترهای به کار رفته در این کد به شکل زیر است:

svc از کلاس svm است.

Kernel از جنس string است، مقداری اختیاری است و پیش فرض آن ‘rbf’ است. این پارامتر تعیین کننده‌ی

نوع kernel به کار گرفته شده در الگوریتم است. می‌تواند یکی از مقادیر ‘linear’، ‘poly’، ‘rbf’،

‘sigmoid’ و ‘precomputed’ را بگیرد که مقدار پیش‌فرض ‘rbf’ است.

Gamma ضریب کرنل برای کرنل‌های ‘rbf’، ‘poly’ و ‘sigmoid’ است و مقادیر {‘scale’,’auto’} یا اعشاری

می‌گیرد.

Shrinking مقداری Boolean است که اختیاری است و به طور پیش‌فرض True است.

در نهایت تخمین‌گر را بر روی ورودی X و ستون خروجی Yمدل می‌کنیم (با اعمال تابع fit)، و خروجی به شکل زیر

است:

SVC(C = 1.0, cache\_size = 200, class\_weight = None, coef0 = 0.0,

decision\_function\_shape = 'ovr', degree = 3, gamma = 'scale', kernel = 'linear',

max\_iter = -1, probability = False, random\_state = None, shrinking = False,

tol = 0.001, verbose = False)

حال می‌توانیم به کمک صفت coef\_ از این کلاس، وزن اختصاصی هر ویژگی داده‌های ورودی را به دست آوریم که

این ویژگی مختص کرنل خطی است:

SVCClf.coef\_

که خروجی به صورت زیر است:

array([[0.5, 0.5]])

می‌توانیم به کمک کد زیر مقدار برچسب برای ورودی دلخواه را به دست آوریم:

SVCClf.predict([[-0.5,-0.8]])

خروجی کد به شکل زیر است:

array([1])

بردارهای پشتیبان را نیز می‌توان به شکل زیر به دست آورد:

SVCClf.support\_vectors\_

خروجی بردارهای پشتیبان به شکل زیر است:

array(

[

[-1., -1.],

[ 1., 1.]

]

)

نتیجه گیری: برای کار کردن با این کتابخانه لازم است تا با کتابخانه‌های Numpy، Pandas و matplotlib آشنایی

داشته باشید و بتوانید با ساختار داده‌های آن‌ها کار کنید. این کتابخانه مدل‌های زیادی را با تخمین‌گرهای خود فیت

(fit) می‌کند و با استفاده از API امکان تست آن را روی داده‌های تست فراهم می‌کند.

مفاهیم Scikit-Learn در پایتون

مهدی باغگلی

منبع:

<https://7learn.com/blog/scikit-learn-library-tutorail>

16 صفحه