Scikit-learn & Preprocessing Data

Scikit-learn (Import : sklearn) توابع و کلاس های مختلفی را برای پیش پردازش داده ها و مدل سازی (یادگیری ماشین) ارائه می دهد.

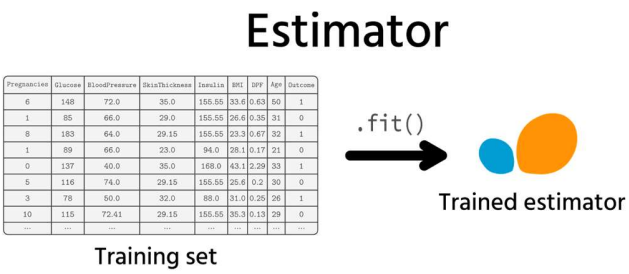
اشیاء اصلی sklearn ، برآوردگر(estimator) ، تبدیل کننده (transformer) ، پیش بینی کننده (predictor)و مدل هستند.

Estimator

هر کلاس **sklearn** با متد **() fit.** یک **Estimator** در نظر گرفته می شود.

متد **()fit** به یک شی اجازه می دهد تا از داده ها یاد بگیرد.

Fit method args : پارامترهای **X** و **y** را می گیرد (**y** برای کارهای یادگیری بدون نظارت اختیاری است).

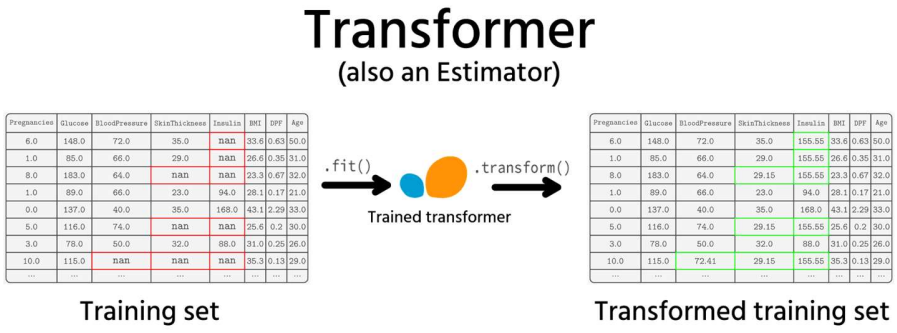


Transformer

یک تبدیل کننده دارای روش **() fit.** و متد **transform.** است که داده ها را به نوعی تبدیل می کند.

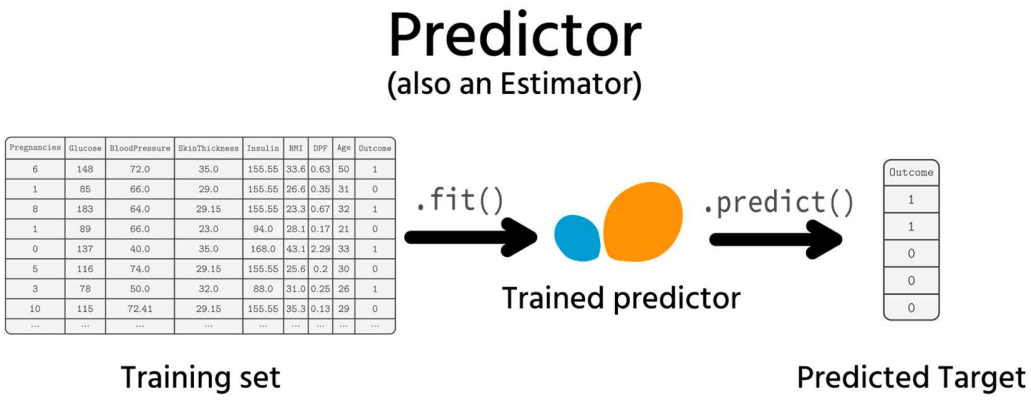
معمولاً، تبدیل کننده ها قبل از تبدیل داده ها باید چیزی را از داده یاد بگیرند ، بنابراین باید از **()fit.** و سپس **(transform.)** استفاده کنید. برای جلوگیری از آن، ترانسفورماتورها متد **fit_transform()** را نیز دارند.

نکته : ترانسفورماتورها معمولاً برای تبدیل آرایه **X** استفاده می شوند. با این حال، همانطور که در مثال **LabelEncoder** خواهیم دید، برخی از ترانسفورماتورها برای آرایه **y** ساخته شده اند.



Predictor

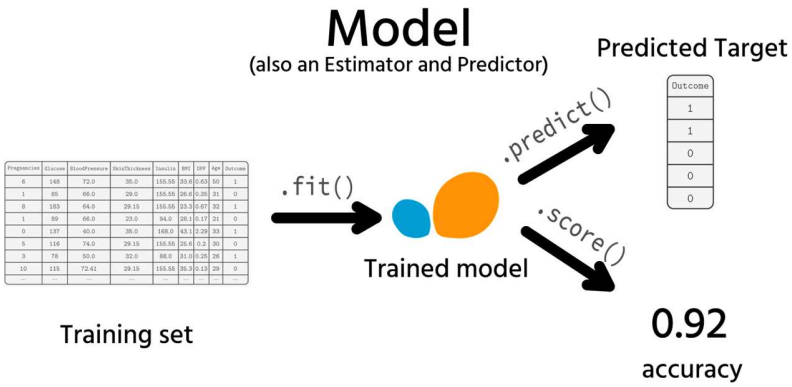
پیش‌بینی‌کننده ، تخمین‌گر (**estimator**) است که دارای **()fit.** و متد **predict()** است. از متد **()predict.** برای پیش بینی استفاده می شود.



Model

یک مدل پیش بینی کننده ای است که متد score() را نیز دارد.

روش score() یک امتیاز (متریک) را برای اندازه گیری عملکرد پیش بینی محاسبه می کند.



Type of class	Required methods
Estimator	`.fit()`
Transformer	`.fit()`, `.transform()`, `.fit_transform()`
Predictor	`.fit()`, `.predict()`
Model	`.fit()`, `.predict()`, `.score()`

مرحله پیش پردازش شامل کار با تبدیل کننده ها است و ما در مرحله مدل سازی با پیش بینی کننده ها (به طور خاص با مدل ها) کار می کنیم.

آشنایی با Dataset

داده ها در یک فایل .csv یا xlsx.موجود است.

ما فایل را از یک ادرس با استفاده از تابع pd.read_csv() بارگذاری می کنیم.

خلاصه ای از تفاوت های کلیدی بین xlsx و csv آمده است:

ویژگی	xlsx	Csv
فرمت	اختصاصی مایکروسافت	مبتنی بر متن
پیچیدگی	می تواند شامل قالب بندی سلول، فرمول ها، نمودارها و سایر داده های پیچیده باشد.	ساده تر، معمولاً فقط شامل داده های خام است.
سازگاری	به طور پیش فرض توسط مایکروسافت اکسل باز می شود، اما می تواند توسط سایر برنامه های صفحه گسترده نیز باز شود.	می تواند توسط طیف گسترده ای از برنامه ها، از جمله ویرایشگرهای متن ساده، باز شود.
اندازه فایل	معمولاً بزرگتر از فایل هایcsv	معمولاً کوچکتر از فایل هایxlsx

برخی از مشکلاتی را که باید برای دیتا ست ها قبل مدلسازی حل کنیم :

- **داده های از دست رفته** - Missing data
- **متغیرهای طبقه بندی شده** - Categorical variables
- **مقیاس های مختلف** – Different scales

Missing data

بیشتر الگوریتم‌های ML نمی‌توانند مقادیر از دست رفته را به طور خودکار مدیریت کنند، بنابراین قبل از اینکه مجموعه آموزشی را به یک مدل برسانیم، باید آن‌ها را حذف کنیم (یا با مقادیری جایگزین کنیم که به آن imputing می‌گویند).

کتابخانه pandas ، سلول های خالی جدول را با NaN پر می کنند .

اگر حداقل یک NaN در داده ها وجود داشته باشد، اکثر مدل های ML با خطا مواجه می شوند.

species	island	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
Adelie	Torgersen	18.7	181.0	3750.0	MALE
Adelie	Torgersen	17.4	186.0	3800.0	FEMALE
Adelie	Torgersen	18.0	195.0	3250.0	FEMALE
Adelie	Torgersen	nan	nan	nan	nan
Adelie	Torgersen	19.3	193.0	3450.0	FEMALE
Adelie	Torgersen	20.6	190.0	3650.0	MALE
Adelie	Torgersen	17.8	181.0	3625.0	FEMALE
Adelie	Torgersen	19.6	195.0	4675.0	MALE
Adelie	Torgersen	18.1	193.0	3475.0	nan
Adelie	Torgersen	20.2	190.0	4250.0	nan
...	...	...	...	...	...

Categorical data

داده‌ها حاوی داده‌های category **هستند که قبلاً می‌دانیم مدل‌های یادگیری ماشین نمی‌توانند آن‌ها را مدیریت کنند.**

بنابراین باید داده های طبقه بندی شده را به عددی رمزگذاری کنیم.

species	island	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
Adelie	Torgersen	18.7	181.0	3750.0	MALE
Adelie	Torgersen	17.4	186.0	3800.0	FEMALE
Adelie	Torgersen	18.0	195.0	3250.0	FEMALE
Adelie	Torgersen	nan	nan	nan	nan
Adelie	Torgersen	19.3	193.0	3450.0	FEMALE
Adelie	Torgersen	20.6	190.0	3650.0	MALE
Adelie	Torgersen	17.8	181.0	3625.0	FEMALE
Adelie	Torgersen	19.6	195.0	4675.0	MALE
Adelie	Torgersen	18.1	193.0	3475.0	nan
Adelie	Torgersen	20.2	190.0	4250.0	nan
...	...	...	...	...	...

Different scales

مقادیر «culmen_depth_mm» **از** 13.1 **تا** 21.5 **متغیر است در حالی که مقادیر** «body_mass_g» **از** 2700 **تا** 6300 **متغیر است. به همین دلیل، برخی از مدل‌های** ML **ویژگی** «body_mass_g» **را بسیار مهم‌تر از** «culmen_depth_mm» **می‌دانند.**

Scaling solves this problem

Dealing with Missing Values

فقط تعداد کمی از مدل‌های یادگیری ماشین داده‌های با مقادیر گمشده را تحمل می‌کنند. بنابراین باید اطمینان حاصل کنیم که داده‌های ما حاوی مقادیر گمشده نیستند. اگر این کار را کرد، می توانیم:

- **ردیف حاوی مقادیر از دست رفته را حذف کنید.**
- **سلول های خالی را با مقداری پر کنید. به آن انتساب (imputing) نیز می گویند.**

برای بررسی اینکه آیا مجموعه داده شما مقادیر nan **ندارد ، می توانید از متد** DataFrame.info() **استفاده کنید. (** df.isnull() – df.isna() **)**

Null is another Name for missing values.(NAN)

حذف ردیف (نمونه) : ما می‌توانیم با خیال راحت ردیف‌هایی را حذف کنیم که اطلاعات بسیار کمی را به ما میدهند .

نسبت دادن به ردیف : ردیف‌های دیگر که حاوی اطلاعات بسیار مفیدتری هستند و فقط حاوی NaN هستند، به جای حذف کامل آنها ما فقط می توانیم مقادیری را برای سلول های NaN در نظر بگیریم. اغلب با استفاده از تبدیل کننده SimpleImputer به دست می آید.

Imputing Challenge

SimpleImputer مقادیر از دست رفته را با یک مقدار منفرد خاص جایگزین می کند (برای هر ستون مقدار واحد خودش).

اما باید مقداری را انتخاب کنیم که به آن نسبت داده شود.

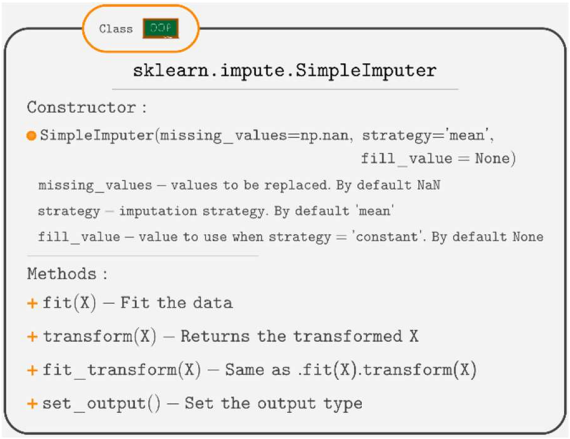
انتخاب رایج این است که میانگین را برای مقادیر عددی و حالت (متداول ترین مقدار) را برای مقادیر طبقه بندی کنیم .

mean for numerical values and *mode* (most frequent value) for categorical values

می توان آن را با استفاده از پارامترهای استراتژی کنترل کرد :

- **strategy='mean' – impute with mean along each column**
- **strategy='median' – impute with median along each column**
- **strategy='most_frequent' – impute with mode along each column (مقدار mode)**
- **strategy='constant' – impute with constant number specified in fill_value parameter**

پارامتر missing_values کنترل می کند که چه مقادیری گم شده در نظر گرفته می شوند. به طور پیش فرض، NaN است، اما در مجموعه داده های مختلف، می تواند یک رشته خالی '' یا هر چیز دیگری باشد.



SimpleImputerو بسیاری از ترانسفورماتورهای دیگر با series پاندا کار نمی کنند، فقط با DataFrame کار می کنند.

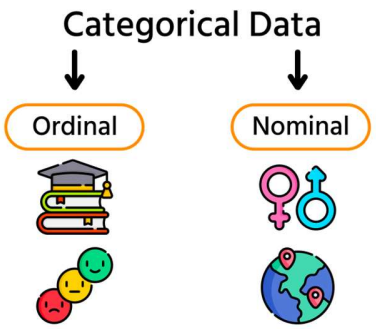
اما انتخاب یک ستون از یک DataFrame (df['column']) یک سری را برمی‌گرداند. برای جلوگیری از آن، می‌توانید از df[['column']] (به دو براکت توجه کنید) مانند این استفاده کنید

هنگامی که از متد SimpleImputer ()fit_transform. استفاده می کنید، یک آرایه دوبعدی برمی گرداند، اما pandas هنگام اختصاص دادن به ستون DataFrameانتظار یک آرایه (یا یک سری) یک بعدی را دارند.

برای حل این مشکل، می‌توانید از متد ravel() برای تبدیل آرایه دو بعدی به یک آرایه ۱ بعدی استفاده کنید

عالی! ما با مشکل مقادیر از دست رفته در مجموعه داده خود برخورد کردیم. ما ردیف‌هایی را که بیش از یک عدد تهی داشتند حذف کردیم و ستون «جنس» را با بیشترین مقدار MALE - نسبت دادیم.

OrdinalEncoder(Categorical Data Type)



داده های ترتیبی (ordinal) از نظم طبیعی پیروی می کنند، در حالی که اسمی (nomial) این گونه نیست.

از آنجایی که یک نظم طبیعی وجود دارد، می‌توانیم دسته‌ها را به اعداد در آن ترتیب رمزگذاری کنیم.

برای مثال، ستون «نرخ» حاوی مقادیر «وحشتناک»، «بد » خوب» و «عالی» را رمزگذاری می‌کنیم:

'Bad' – 1
'OK' – 2
'Good' – 3
'Great' – 4

برای رمزگذاری داده های ترتیبی از OrdinalEncoder استفاده می شود. این فقط دسته ها را به 0، 1، 2، ... رمزگذاری می کند.

Class OOP

sklearn.preprocessing.OrdinalEncoder

Constructor :

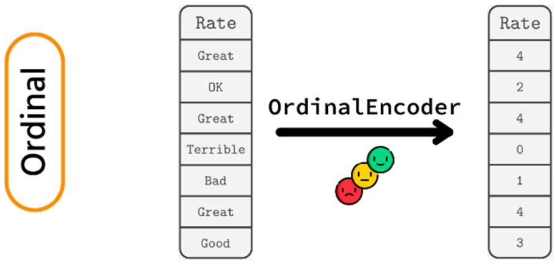
OrdinalEncoder(categories='auto')

categories – ordered categories for each column.
Default value is 'auto' to determine automatically

Methods :

+ fit(X) – Fit the data
+ transform(X) – Returns the transformed X
+ fit_transform(X) – Same as .fit(X).transform(X)
+ set_output() – Set the output type
+ inverse_transform(X) – Convert back to the original representation

استفاده از OrdinalEncoder مانند هر ترانسفورماتور دیگری آسان است. تنها مشکل این است که آرگومان دسته ها را به درستی مشخص کنید.



OrdinalEncoder بیشتر برای تبدیل ویژگی ها (متغیر X) استفاده می شود. و متغیر X معمولا یک DataFrame حاوی بیش از 1 ستون است.

به همین دلیل، آرگومان دسته‌ها اجازه می‌دهد تا لیست دسته‌ها را برای هر ستون مشخص کنید،

اگر می‌خواهید فقط 1 ستون را تغییر دهید، همچنان باید فهرستی حاوی فهرست دیگری ارسال کنید،

همچنین به همین دلیل است که متد .fit_transform() انتظار DataFrame را دارد و با Series کار نمی کند، بنابراین شما باید df[['column']] را برای تبدیل تنها یک ستون ارسال کنید.

One-Hot Encoder(nomial type)

داستان با مقادیر اسمی کمی پیچیده تر است.

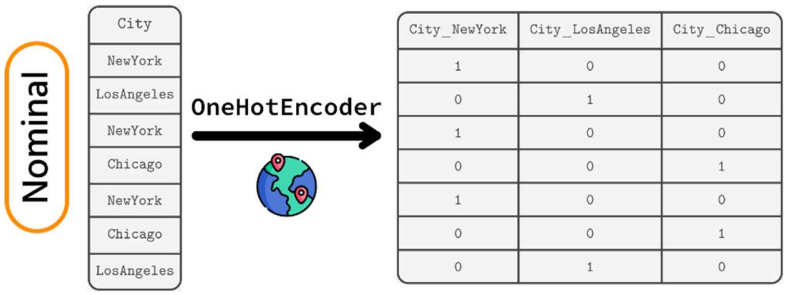
فرض کنید این ویژگی حاوی داده های ترتیبی است، به عنوان مثال، نرخ های کاربر. مقادیر آن وحشتناک، بد، خوب، خیلی خوب و عالی است. رمزگذاری آن مقادیر به صورت 0 تا 4 منطقی به نظر می رسد.

مدل ML ترتیب را در نظر می گیرد. حال، ویژگی «شهر» را تصور کنید که شامل پنج شهر مختلف است. اگر آنها را از 0 تا 4 رمزگذاری کنیم، حالت ML نیز فکر می کند که یک نظم منطقی وجود دارد، اما وجود ندارد.

برای رمزگذاری داده های اسمی، از ترانسفورماتور OneHotEncoder استفاده می شود.

برای هر مقدار منحصر به فرد یک ستون ایجاد می کند.

سپس برای هر سطر، 1 را به ستون مقدار این سطر و 0 را برای ستون های دیگر تنظیم می کند.



آنچه در ابتدا "NewYork" بود، اکنون دارای 1 در ستون City_NewYork و 0 در سایر ستون‌های City_ است.

برای استفاده ازOneHotEncoder ، فقط باید یک شی را مقداردهی اولیه کنید و مانند هر ترانسفورماتور دیگری، ستون ها را به fit_transform() ارسال کنید.

OneHotEncoderیک ماتریس پراکنده را برمی گرداند، که می تواند با استفاده از روش toarray() به آرایه NumPy تبدیل شود.

LabelEncoder

OrdinalEncoder و OneHotEncoder معمولا برای رمزگذاری ویژگی ها (متغیر X) استفاده می شوند.

اما هدف (متغیر y) نیز می تواند طبقه بندی شود.

LabelEncoder برای رمزگذاری هدف، صرف نظر از اینکه اسمی یا ترتیبی باشد، استفاده می شود.

مدل‌های ML ترتیب هدف را در نظر نمی‌گیرند و به آن اجازه می‌دهند به عنوان هر مقدار عددی کدگذاری شود.

LabelEncoder هدف را روی اعداد 0، 1، ... رمزگذاری می کند.

هدف را با استفاده از LabelEncoder کد می کند و سپس از متد ()inverse_transform برای تبدیل آن به نمایش اصلی استفاده می کند.

از آنجایی که LabelEncoder برای تبدیل هدف(y) ، که معمولاً یک ستون است، استفاده می شود، برخلافOrdinalEncoder ، با pandas series به خوبی کار می کند.

بنابراین ما فقط می توانیم متغیر y را به متد ()fit_transform. پاس کنیم.

Why Scale the Data?

اکنون که مقادیر از دست رفته و ویژگی‌های طبقه‌بندی کدگذاری شده را مدیریت کرده‌ایم، با تمام مشکلاتی که هنگام وارد کردن به مدل خطا ایجاد می‌کنند، برخورد کرده‌ایم.

اما یک مشکل دیگر وجود دارد که به آن اشاره کردیم، مقیاس های مختلف.

اگر داده های وضعیت فعلی را به مدل وارد کنید، این مشکل باعث خطا نمی شود.

اما می تواند به طور قابل ملاحظه ای برخی از مدل های ML را از لحاظ پرفورمنس بدتر کند.

مثالی را در نظر بگیرید که در آن یک ویژگی «سن» است و ویژگی دوم «درآمد» است. ویژگی اول از 18 تا 50 و ویژگی دوم از 25000 تا 500000 متغیر است.

culmen_depth_mm	body_mass_g
18.7	3750

می توان گفت که اختلاف ده سال از اختلاف ده دلاری مهم تر است.

اما برخی از مدل ها (مانند k-NN) این تفاوت را دارای اهمیت یکسانی می دانند.

در نتیجه، ستون "درآمد" تاثیر بسیار مهمتری بر مدل خواهد گذاشت.

بنابراین ما نیاز داریم که ویژگی‌ها تقریباً همان محدوده را داشته باشند تا k-NN به درستی کار کند.

مدل‌های دیگر کمتر تحت تأثیر مقیاس‌های مختلف قرار می‌گیرند، اما برخی زمانی که داده‌ها مقیاس‌بندی می‌شوند، بسیار سریع‌تر عمل می‌کنند.

بنابراین مقیاس بندی داده ها معمولاً به عنوان آخرین مرحله در پیش پردازش گنجانده می شود.

همانطور که در بالا ذکر شد، مقیاس بندی داده ها معمولا آخرین مرحله از مرحله پیش پردازش است. این به این دلیل است که تغییرات در ویژگی‌های ایجاد شده پس از مقیاس‌بندی می‌تواند داده‌ها را مجدداً غیرمقیاس کند.

StandardScaler, MinMaxScaler, MaxAbsScaler

سه روش رایج برای مقیاس بندی داده ها وجود دارد:

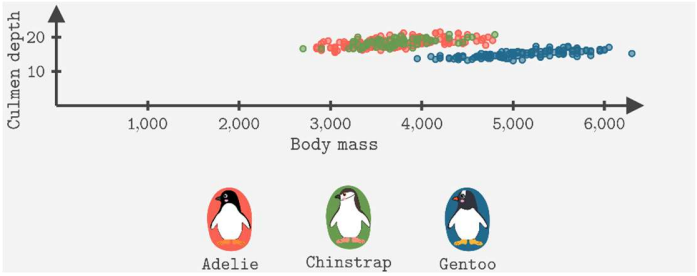
MinMaxScaler - ویژگی ها را در محدوده [0، 1] مقیاس می کند.

MaxAbsScaler - ویژگی هایی مانند حداکثر مقدار مطلق 1 را مقیاس می کند

(بنابراین تضمین می شود که داده ها در محدوده [-1، 1] قرار دارند).

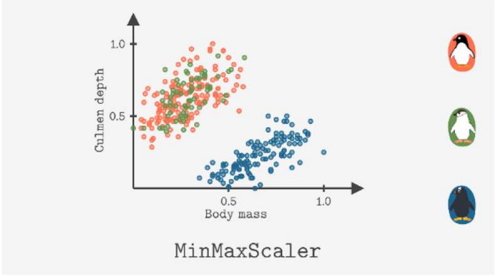
StandardScaler - استاندارد کردن ویژگی‌ها که میانگین آن برابر 0 و واریانس برابر با 1 باشد.

برای نشان دادن نحوه عملکرد مقیاس‌کننده‌ها، از ویژگی‌های «culmen_depth_mm» و «body_mass_g» مجموعه داده‌های پنگوئن‌ها استفاده می‌کنیم. بیایید آنها را طرح ریزی کنیم.



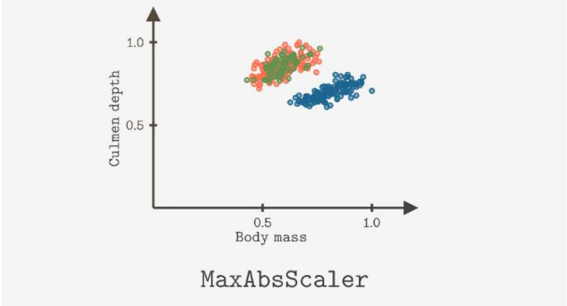
MinMaxScaler با کم کردن مقدار حداقل (برای شروع مقادیر از صفر) و سپس تقسیم بر (x_max - x_min) کار می کند تا آن را کمتر یا مساوی 1 کند.

$$\text{MinMaxScaler}: \; x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

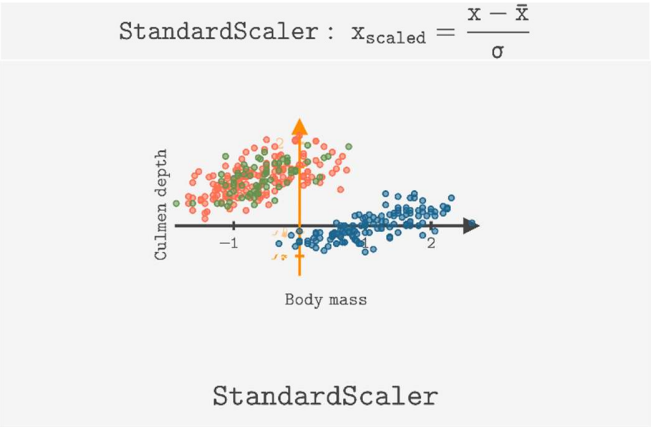


MaxAbsScaler با یافتن حداکثر مقدار مطلق و تقسیم هر مقدار بر آن کار می کند. این تضمین می کند که حداکثر مقدار مطلق 1 است.

$$\text{MaxAbsScaler}: \; x_{scaled} = \frac{x}{\max|x|}$$



ایده StandardScaler از آمار ناشی می شود. با تفریق میانگین (به مرکز حول صفر) و تقسیم بر انحراف استاندارد (برای ایجاد واریانس برابر با 1) کار می کند.



خروجی زیبایی نیست زیرا مقیاس‌کننده‌ها داده‌ها را به آرایه NumPy تبدیل می‌کنند، اما با piplines ، مشکلی نخواهد بود.

شما فقط باید ستون های ویژگی (متغیر X) را مقیاس کنید.

نیازی به اندازه گیری هدف نیست. فقط به دست آوردن تبدیل معکوس سخت تر میشود.

از کدام مقیاس کننده استفاده کنیم؟

StandardScaler نسبت به مقادیر پرت حساسیت کمتری دارد، بنابراین مقیاس‌کننده پیش‌فرض خوبی است.

اگر از StandardScaler خوشتان نمی‌آید، بین MinMaxScaler وMaxAbsScaler ، به اولویت‌های شخصی مربوط می‌شود، مقیاس دادن داده‌ها به محدوده [0،1] یا به[1،1-]

Summery :

اینها رایج ترین مشکلاتی هستند که مجموعه داده ها با آن روبرو هستند، بنابراینImputation ، Encodingو Scaling تقریباً در هر pipline گنجانده شده است.

Imputers (Dealing with missing values)

Imputer	What for
`SimpleImputer(strategy='most_frequent')`	Impute categorical data
`SimpleImputer(strategy='mean'/'median')`	Impute numerical data

Encoders (Dealing with categorical values)

Encoder	What for
`OrdinalEncoder`	Encode ordinal features
`OneHotEncoder`	Encode nominal features
`LabelEncoder`	Encode target

Scalers (Dealing with different scales)

Scaler	What for
`MinMaxScaler`	Scale the features to a [0,1] range
`MaxAbsScaler`	Scale the features to a [-1,1] range
`StandardScaler`	Scale the features so that the mean is 0 and the variance is 1

Pipline

مشکل دیگر این است که برای انجام یک پیش‌بینی نمونه‌های جدید باید همان مراحل پیش‌پردازش را طی کنند، بنابراین باید همه آن تبدیل‌ها را دوباره انجام دهیم.

خوشبختانه، Scikit-learn یک کلاس Pipeline ارائه می‌کند - یک راه ساده برای جمع‌آوری همه آن تبدیل‌ها با هم، بنابراین تبدیل داده‌های آموزشی و نمونه‌های جدید آسان‌تر است.

Pipeline محفظه ای برای تمام ترانسفورماتورها (و برآوردگر نهایی) است. با فراخوانی متد fit_transform(). یک شیPipeline ، به صورت متوالی هر ترانسفورماتور fit_transform. را فراخوانی می کند.

به این ترتیب، شما فقط باید یک بار fit_transform(). را فراخوانی کنید تا یک مجموعه آموزشی را تغییر دهید و سپس متد transform(). را برای تبدیل نمونه های جدید فراخوانی کنید.

ColumnTransformer

هنگامی که متد fit_transform(X). را در شی Pipeline فراخوانی می کنیم، هر تبدیل کننده روی X اعمال می شود .

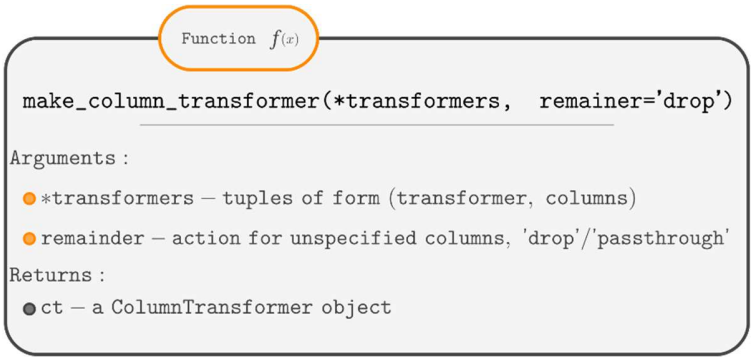
اما این رفتاری نیست که ما می خواهیم.

ما نمی‌خواهیم مقادیر عددی از قبل را رمزگذاری کنیم، یا ممکن است بخواهیم ترانسفورماتورهای مختلفی را در ستون‌های مختلف اعمال کنیم (به عنوان مثال، OrdinalEncoder برای ویژگی‌های ترتیبی و OneHotEncoder برای اسمی).

ترانسفورماتور ColumnTransformer این مشکل را برطرف می کند.

به ما این امکان را می دهد که هر ستون را جداگانه بررسی کنیم.

برای ایجاد یک ColumnTransformer ، می توانید از یک تابع خاص make_column_transformer از ماژول sklearn.compose استفاده کنید.



این تابع به عنوان آرگومان ، تاپلی که شامل transformer و لیست ستون ها و تنظیمات دریافت میکند .

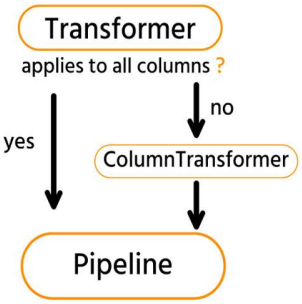
مثال :

```
ct = make_column_transformer(
        (OrdinalEncoder(), ['education']) ,
        (OneHotEncoder(), ['gender']),
        remainder='passthrough
)
```

در پایان به آرگومان remainder توجه کنید. مشخص می‌کند که با ستون‌هایی که در make_column_transformer ذکر نشده‌اند چه باید کرد (در اینجا فقط «جنسیت» و «آموزش» ذکر شده است).

به‌طور پیش‌فرض، روی «drop» تنظیم شده است، به این معنی که حذف خواهند شد.

باید remainder='passthrough' را تنظیم کنید تا سایر ستون‌ها دست نخورده منتقل شوند.



برای استفاده از make_column_transformer، باید مراحل زیر را انجام دهید:

1. یک لیست از پیش پردازنده ها را ایجاد کنید.
2. یک استراتژی انتخاب را انتخاب کنید.
3. یک شیء make_column_transformerرا با لیست پیش پردازنده ها و استراتژی انتخاب ایجاد کنید.
4. شیء make_column_transformerرا روی مجموعه داده خود مناسب کنید.

برای ایجاد خط لوله با استفاده از Scikit-learn، می توانید از سازنده کلاس Pipeline یا تابع make_pipeline، هر دو از ماژول sklearn.pipeline استفاده کنید.

شما فقط باید تمام ترانسفورماتورها را به عنوان آرگومان به یک تابع منتقل کنید. ایجاد خطوط لوله به همین سادگی است.

با این حال، هنگامی که شما متد .fit_transform(X) را در شی Pipeline فراخوانی می کنید، .fit_transform(X) را برای هر ترانسفورماتور داخل خط لوله اعمال می کند.

بنابراین اگر می‌خواهید با برخی از ستون‌ها متفاوت رفتار کنید، باید از ColumnTransformer استفاده کنید و آن را به .make_pipeline() بدهید.

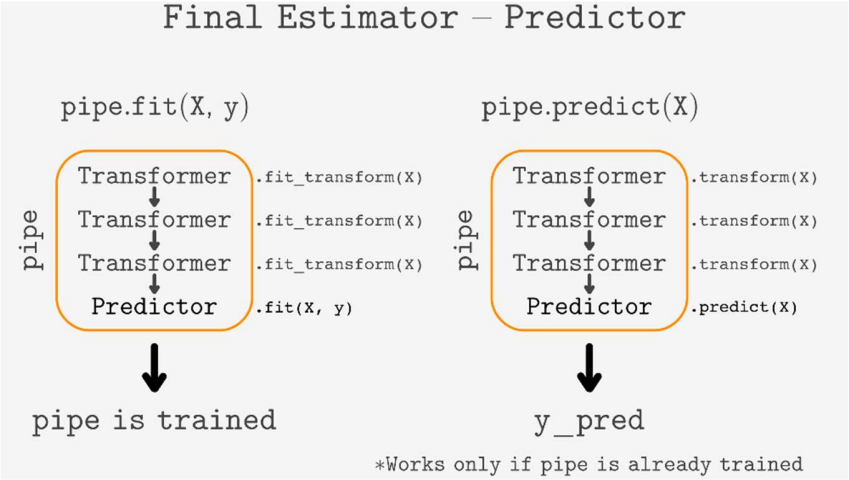
Final Estimator

در حال حاضر، ما فقط از یک خط لوله برای پیش پردازش استفاده کردیم .

با این حال ، بیشتر اوقات، این نقطه پایانی نیست. معمولاً پس از پیش پردازش، می خواهیم از این داده های تبدیل شده در یک Predictor (Model) استفاده کنیم.

به همین دلیل است که کلاس Pipeline اجازه می دهد تا مرحله نهایی هر تخمینگر باشد، معمولاً یک پیش بینی.

تصویر زیر نشان می دهد که خط لوله چگونه کار می کند زمانی که آخرین مرحله آن یک پیش بینی است.



هنگامی که متد .fit() یک خط لوله را فراخوانی می کنیم، fit_transform()روی هر ترانسفورماتور فراخوانی می شود.

اما وقتی متد predict. () را فراخوانی می کنیم، متد transform() فراخوانی می شود.

متد predict () بیشتر برای پیش‌بینی نمونه‌های جدید استفاده می‌شود، که باید دقیقاً به همان روشی که مجموعه آموزشی در طول .fit) تغییر شکل داده شود.

اگر از متد .fit_transform() به جای .transform() برای تبدیل نمونه‌های جدید استفاده کنیم، OneHotEncoder می‌تواند ستون‌های جدیدی را با ترتیب متفاوتی ایجاد کند و Scalers به احتمال زیاد داده‌ها را کمی متفاوت مقیاس می‌دهد.

در نتیجه، نمونه‌های جدید متفاوت از مجموعه آموزشی تغییر می‌کنند و پیش‌بینی غیرقابل اعتماد خواهد بود.

زمانی که از خطوط لوله استفاده نمی کنید باید از آن آگاه باشید.

یکی دیگر از مزایای خطوط لوله این است که آنها فقط این مراحل را به طور خودکار انجام می دهند.

برای استفاده از برآوردگر نهایی، فقط باید آن را به عنوان آخرین مرحله خط لوله اضافه کنید .

Models

بیایید مرور کنیم که مدل چیست. در Scikit-learn، این یک برآوردگر است که دارای هر دو متد ()predict و () .score است (و از آنجایی که یک تخمینگر است، متد ()fit نیز وجود دارد).

هنگامی که داده ها از قبل پردازش شدند و برای رفتن به الگوریتم آماده شدند، اولین مرحله ساخت مدل آموزش یک مدل است.

.fit() : فرآیند یادگیری با استفاده از fit (X, y). انجام می شود.

برای آموزش مدلی که وظیفه یادگیری نظارت شده را انجام می دهد (به عنوان مثال، رگرسیون، طبقه بندی)، باید X و y را به متد () fit. منتقل کنید.
اگر با الگوریتم یادگیری بدون نظارت (مثلاً خوشه‌بندی) سر و کار دارید، این کار به داده‌های برچسب‌دار نیاز ندارد، بنابراین فقط می‌توانید متغیرX را ارسال کنید. با این حال، استفاده از fit (X, y). خطایی ایجاد نمی کند. مدل فقط متغیر y را نادیده می گیرد.

در طول فرآیند آموزش، یک مدل هر آنچه را که برای انجام پیش‌بینی نیاز دارد، یاد می‌گیرد.
اینکه یک مدل چه می آموزد و چه مدت تمرین می کند به الگوریتمی که انتخاب می کنید بستگی دارد.
برای هر کار، مدل های زیادی بر اساس الگوریتم های مختلف وجود دارد. برخی از آنها کندتر تمرین می کنند، برخی سریع تر.

اما به طور کلی، آموزش معمولاً وقت‌گیرترین چیز در یادگیری ماشینی است و اگر مجموعه آموزشی بزرگ باشد، یک مدل می‌تواند برای دقیقه‌ها، ساعت‌ها یا حتی روزها تمرین کند.

.predict() : هنگامی که مدل با استفاده از متد .fit () آموزش داده می شود، می تواند پیش بینی ها را انجام دهد.

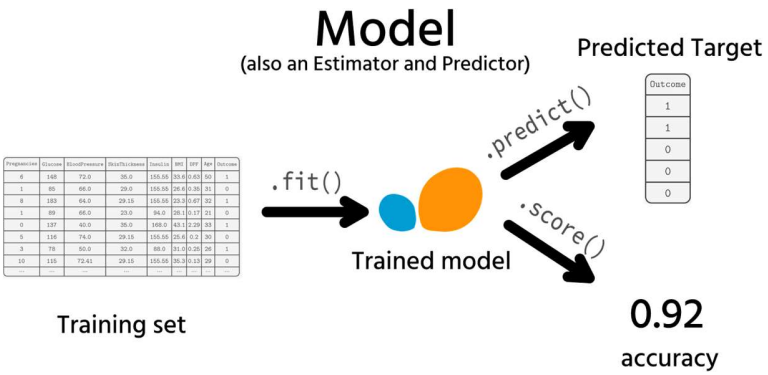
معمولاً می خواهید یک هدف را برای نمونه های جدید پیش بینی کنید، .X_new

```
model.predict(X_new)
```

.score() : روش score (X, y). برای اندازه گیری عملکرد یک مدل آموزش دیده استفاده می شود. معمولاً بر روی مجموعه آزمایشی محاسبه میشود .
متد ()score. به مقادیر هدف واقعی نیاز دارد . این پیش بینی برای نمونه های X_test را محاسبه می کند و این پیش بینی را با هدف واقعی (y_test) با استفاده از برخی متریک مقایسه می کند.

به طور پیش‌فرض، این معیار دقت طبقه‌بندی است. (Accuracy)

$$\text{accuracy} = \frac{\text{predicted correctly}}{\text{predicted correctly} + \text{predicted incorrectly}}$$



Implement Algorithm(K-NN)

k-Nearest Neighbors یک الگوریتم ML است که مبتنی بر یافتن مشابه ترین نمونه ها در مجموعه آموزشی برای پیش بینی است.

KNeighborsClassifier اجرای الگوریتم k-Nearest Neighbors برای classification task است .

نحوه کارکرد :

1. **برای یک نمونه جدید، k نزدیکترین (بر اساس ویژگی ها) نمونه های مجموعه آموزشی را پیدا میکند . آن k نمونه ها همسایه نامیده می شوند.**
2. **بیشترین کلاس را در میان k همسایه پیدا کنید. آن کلاس یک پیش بینی برای نمونه جدید خواهد بود.**

k تعداد همسایه هایی است که می خواهید در نظر بگیرید.

شما باید این عدد را هنگام مقداردهی اولیه مدل مشخص کنید.

با مقادیر مختلف k مدل پیش بینی های متفاوتی به دست می آید.

به آن هایپرپارامتر (hyperparameter) گفته می شود – پارامتری که باید از قبل مشخص کنید و می تواند پیش بینی های مدل را تغییر دهد.

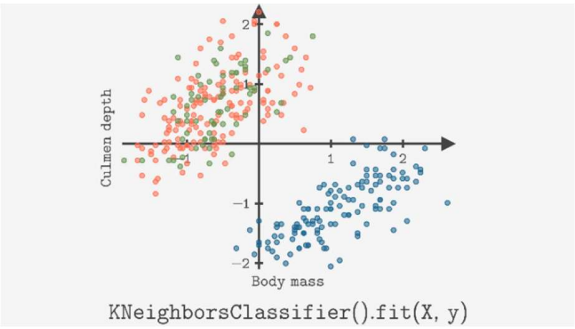
می توانید مقادیر k مختلف را تنظیم کنید و بهترین را برای کار خود بیابید.

این فرآیند تنظیم هایپرپارامترها به عنوان hyperparameter tuning شناخته می شود و می تواند به شما در بهینه سازی عملکرد مدل خود کمک کند.

KNN .Fit :

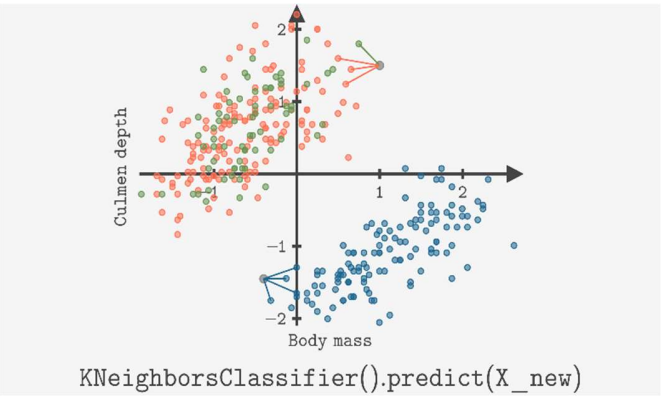
برخلاف اکثر مدل‌های ML، در حین آموزش، KNeighborsClassifier کاری جز ذخیره مجموعه آموزشی انجام نمی‌دهد.

اما حتی اگر آموزش زمان نمی برد، فراخوانی .fit(X, y) برای به خاطر سپردن مجموعه آموزشی الزامی است.



KNN .predict :

در طول پیش‌بینی، KNeighborsClassifier به طور حریصانه k نزدیک‌ترین همسایه‌ها را برای هر نمونه جدید پیدا می‌کند.



ویژگی های اضافی احتمالاً به مدل کمک می کند تا نقاط داده سبز و قرمز را بهتر از هم جدا کند، بنابراین KNeighborsClassifier پیش بینی های بهتری انجام می دهد.

برای سادگی، داده های موجود در یک فایل csv. در حال حاضر به طور کامل از قبل پردازش شده است.

برای تعیینk ، از آرگومان n_neighbors سازنده KNeighborsClassifier استفاده کنید. ما مقادیر 5 (مقدار پیش فرض) و 1 را امتحان خواهیم کرد.

```python
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier

df = pd.read_csv('https://codefinity-content-media.s3.eu-west-1.amazonaws.com/a65
# Assign X, y variables (X is already preprocessed and y is already encoded)
X, y = df.drop('species', axis=1), df['species']
# Initialize and train a model
knn5 = KNeighborsClassifier().fit(X, y) # Trained 5 neighbors model
knn1 = KNeighborsClassifier(n_neighbors=1).fit(X, y) # Trained 1 neighbor model
# Print the scores of both models
print('5 Neighbors score:',knn5.score(X, y))
print('1 Neighbor score:',knn1.score(X, y))
```

ما دقت بسیار خوبی دریافت می کنیم! برای 1–نزدیک ترین همسایه، حتی دقت کامل.

اما نکته ای که می باشد ، این است که ما مدل را روی دیتا های آموزشی تست کردیم . مدل بر روی این مقادیر آموزش داده شده است،

بنابراین به طور طبیعی، نمونه هایی را که قبلاً دیده است به خوبی پیش بینی می کند.

ما باید مدل را بر روی نمونه هایی ارزیابی کنیم که مدل هرگز ندیده است تا بفهمیم چقدر خوب عمل می کند.

Evaluating a Model. Train-Test split.

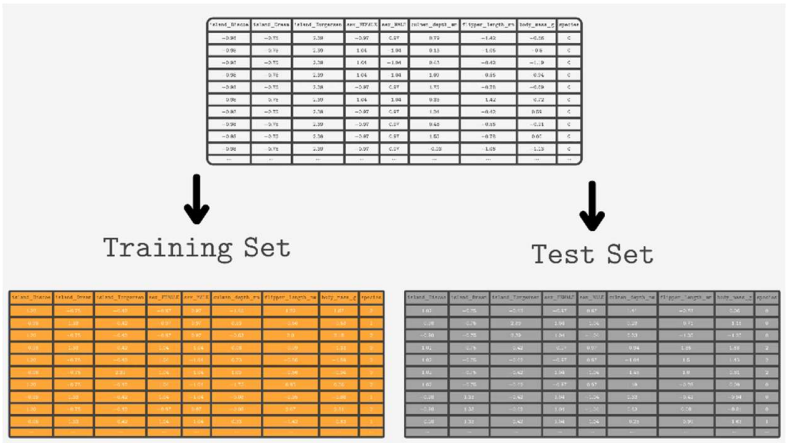
زمانی که ما یک مدل برای پیش‌بینی‌ها می‌سازیم، قبل از اینکه واقعاً چیزی را پیش‌بینی کنیم، لازم است بدانیم مدل چقدر خوب عمل می‌کند.

ارزیابی یک مدل به فرآیند ارزیابی عملکرد آن در پیش بینی ها اشاره دارد.

به همین دلیل متد score(). مورد نیاز است.

اما اگر مدل را با استفاده از داده‌های مجموعه آموزشی ارزیابی کنیم، نتایج غیرقابل اعتماد هستند، زیرا یک مدل احتمالاً بر روی داده‌هایی که روی آن آموزش داده شده است، بهتر عمل می‌کند تا روی داده‌هایی که هرگز ندیده است.

ما می توانیم این کار را با تقسیم تصادفی داده ها به یک مجموعه آموزشی و یک مجموعه آزمایشی انجام دهیم.



معمولاً برای یک مجموعه آزمایشی، زمانی که مجموعه داده کوچک است از 25 تا 40 درصد داده ها، برای مجموعه داده های متوسط از 10 تا 30 درصد و برای مجموعه داده های بزرگ کمتر از 10 درصد استفاده می کنیم.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33) = **0.33 یعنی 33 درصد**

Cross-Validation

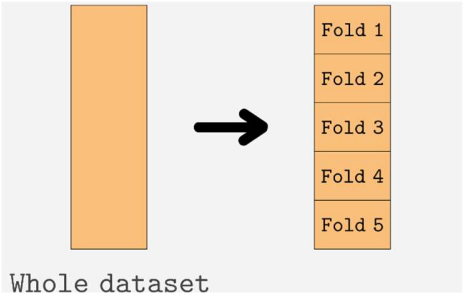
نکات منفی روش train_test_split :

1. **ما فقط از بخشی از مجموعه داده برای آموزش استفاده می کنیم. طبیعتاً هرچه داده های بیشتری به مدل بدهیم، باید از آن بیشتر آموزش ببینیم و عملکرد مدل بهتر خواهد بود.**
2. **یک نتیجه می تواند به شدت به تقسیم بستگی داشته باشد. از آنجایی که مجموعه داده به طور تصادفی تقسیم می شود، اجرای چندین بار کد می تواند نتایج متفاوتی داشته باشد.**

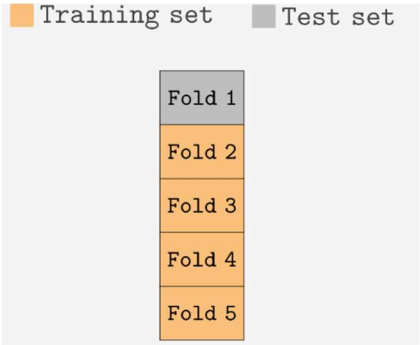
بنابراین رویکرد متفاوتی برای ارزیابی یک مدل به نام cross-validation **(اعتبار متقابل) وجود دارد.**

نحوه کارکرد (براساس k-fold **) :**

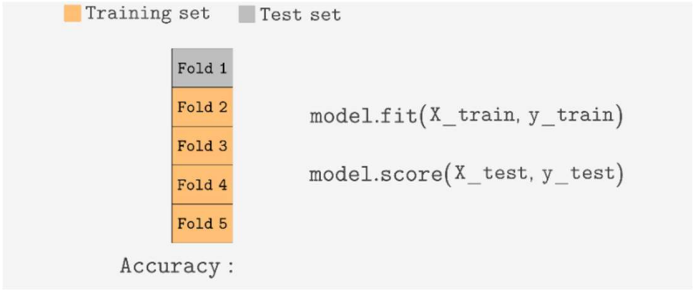
1. **ابتدا یک مجموعه داده کامل را به 5 قسمت مساوی تقسیم می کنیم که folds نامیده می شود.**



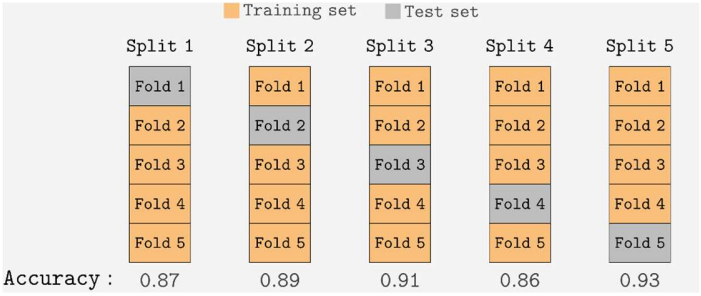
2. **یک بخش را به عنوان مجموعه تست و دیگری را به عنوان مجموعه آموزشی درنظر میگیریم.**



3. مثل همیشه از مجموعه آموزشی برای آموزش مدل و مجموعه تست برای ارزیابی مدل استفاده می کنیم.



4. اکنون، این روند را برای هر فولد تکرار کنید تا یک مجموعه آزمایشی باشد.



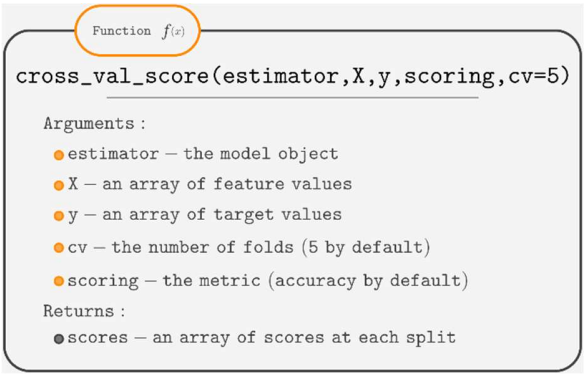
در نتیجه برای هر تقسیم 5 امتیاز دقت می گیریم.

اکنون می‌توانیم میانگین آن 5 امتیاز را برای اندازه‌گیری عملکرد متوسط مدل بگیریم.

برای محاسبه امتیاز cross-validation در پایتون، می‌توانیم از **cross_val_score()** از ماژول **sklearn.model_selection** استفاده کنیم.

اگرچه مثال با 5 فولد نشان داده شده است، شما می توانید از هر تعداد فولد برای اعتبارسنجی متقاطع استفاده کنید.

به عنوان مثال، شما می توانید از 10 تا، 9 برای یک مجموعه آموزشی و 1 برای یک مجموعه تست استفاده کنید. این با استفاده از آرگومان **cv** **cross_val_score**() کنترل می شود.



نتایج پایدارتر و قابل اعتمادتری نسبت به روش split test train نشان می دهد، اما به طور قابل توجهی کندتر است زیرا نیاز به آموزش و ارزیابی مدل 5 بار دارد (n بار)، در حالی که split test train این کار را یکبار انجام می دهد.

scores = cross_val_score(KNeighborsClassifier(), X, y)

اعتبارسنجی متقاطع معمولاً برای تعیین بهترین هایپرپارامترها (به عنوان مثال، بهترین تعداد همسایگان) استفاده می شود.

GridSearchCV

اکنون زمان تلاش برای بهبود عملکرد مدل است!

این کار با یافتن بهترین هایپرپارامترهای متناسب با وظیفه ما انجام می شود.

این فرآیند تنظیم Hyperparameter نامیده می شود.

روش پیش‌فرض این است که مقادیر مختلف فراپارامتر را امتحان کنید و امتیاز cross-validation را برای آنها محاسبه کنید.

سپس فقط مقداری را انتخاب کنید که بهترین امتیاز را به همراه دارد.

Model	Score
KNeighborsClassifier(n_neighbors = 1)	0.83
KNeighborsClassifier(n_neighbors = 3)	0.84
KNeighborsClassifier(n_neighbors = 5)	0.86
KNeighborsClassifier(n_neighbors = 7)	0.88

این فرآیند را می توان با استفاده از کلاس GridSearchCV ماژول sklearn.model_selection انجام داد.

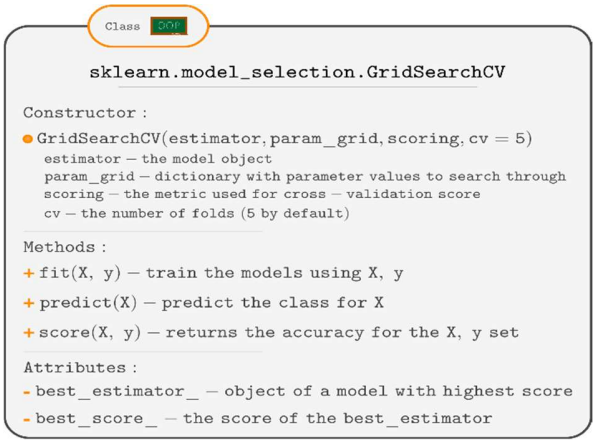
در حین ساختن یک شیGridSearchCV ، باید مدل و پارامترها (به صورت اختیاری امتیازدهی و تعداد foldها) را پاس کنیم.

پارامترها (param_grid) یک dictionary است که شامل تمام پیکربندی هایپرپارامترهایی است که می خواهیم امتحان کنیم.

برای مثال : {param_grid={'n_neighbors': [1, 3, 5, 7] , مقادیر 1، 3، 5 و 7 را به عنوان تعداد همسایگان امتحان می کند.

سپس باید آن را با استفاده از fit (X, y). آموزش دهیم.

پس از آن، با استفاده از ویژگی best_estimator_. و امتیاز cross-validation آن با استفاده از ویژگی best_score_. می توانیم به مدلی با بهترین پارامترها دسترسی پیدا کنیم.



پس از آن، با استفاده از ویژگی best_estimator_. و امتیاز cross-validation آن با استفاده از ویژگی best_score_. می توانیم به مدلی با بهترین پارامترها دسترسی پیدا کنیم.

```
param_grid = {'n_neighbors[1,3,5,7,9]:'}
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid)

grid_search.fit(X, y)


print(grid_search.best_estimator_)
print(grid_search.best_score_)
```

گام بعدی انتخاب best_estimator_ و آموزش آن بر روی کل مجموعه داده است،

زیرا از قبل می‌دانیم که بهترین پارامترها را دارد (از آنهایی که امتحان کردیم)، و امتیاز آن را می‌دانیم.

این مرحله آنقدر واضح است که GridSearchCV آن را به طور پیش فرض انجام می دهد.

بنابراین شی (grid_search در مثال ما) به یک مدل آموزش دیده با بهترین پارامترها تبدیل می شود.

اکنون می توانیم از این شی برای پیش بینی یا ارزیابی استفاده کنیم.

The Flaw of GridSearchCV

لازم به ذکر است که KNeighborsClassifier بیش از 1 هایپرپارامتر برای تغییر دادن دارد.

در حال حاضر، ما فقط از n_neighbors استفاده می کردیم. اجازه دهید به طور کوتاه دو ابرپارامتر دیگر، weights و p را مورد بحث قرار دهیم.

Weights

همانطور که میدانیم ، KNeighborsClassifier با یافتن k نزدیکترین همسایه کار می کند.

سپس پرتکرارترین کلاس را در بین آن همسایه ها اختصاص می دهد، صرف نظر از اینکه هر کدام چقدر نزدیک هستند.

رویکرد دیگر این است که فاصله تا همسایه را نیز در نظر بگیرید تا کلاس های همسایه های نزدیک وزن بیشتری داشته باشند.

این را می توان با تنظیم 'weights='distance انجام داد.

به‌طور پیش‌فرض، رویکرد اول استفاده می‌شود که با استفاده از 'weights='uniform تنظیم می‌شود.

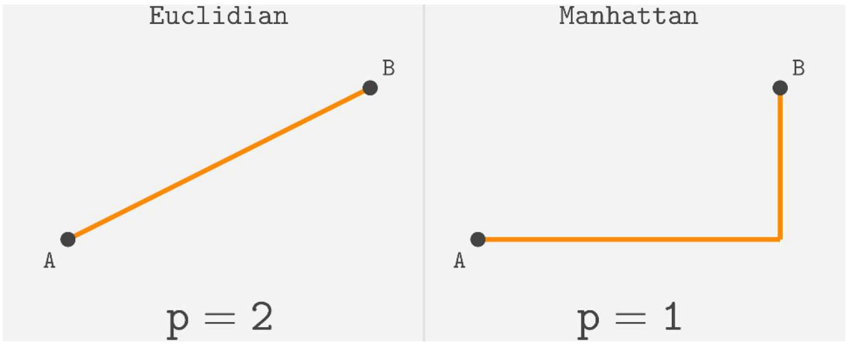
P

همچنین روش های مختلفی برای محاسبه فاصله وجود دارد. هایپرپارامتر p آن را کنترل می کند.

p=1 فاصله منهتن است.

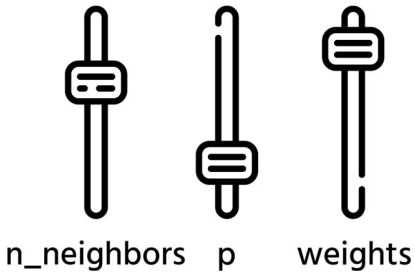
p=2 فاصله اقلیدسی است که در مدرسه یاد گرفتید.

یک پارامتر p می تواند هر عدد صحیح مثبتی را بگیرد. فواصل مختلف زیادی وجود دارد، اما تجسم آنها از p=1 یا p=2 دشوارتر است.



پارامتر p می تواند هر عدد صحیح مثبت را بگیرد .

فواصل مختلف زیادی وجود دارد، اما تجسم آنها از p=1 یا p=2 دشوارتر است.



GridSearchCV تمام ترکیب های ممکن را برای یافتن بهترین ها امتحان می کند.

```
( n_neighbors=1, p=1, weights='distance' )    ( n_neighbors=5, p=1, weights='distance' )
( n_neighbors=1, p=1, weights='uniform' )     ( n_neighbors=5, p=1, weights='uniform' )
( n_neighbors=1, p=2, weights='distance' )    ( n_neighbors=5, p=2, weights='distance' )
( n_neighbors=1, p=2, weights='uniform' )     ( n_neighbors=5, p=2, weights='uniform' )
( n_neighbors=3, p=1, weights='distance' )    ( n_neighbors=7, p=1, weights='distance' )
( n_neighbors=3, p=1, weights='uniform' )     ( n_neighbors=7, p=1, weights='uniform' )
( n_neighbors=3, p=2, weights='distance' )    ( n_neighbors=7, p=2, weights='distance' )
( n_neighbors=3, p=2, weights='uniform' )     ( n_neighbors=7, p=2, weights='uniform' )
```

این در حال حاضر کار زیادی است. اما اگر بخواهیم مقادیر بیشتری را امتحان کنیم چه؟
برای مجموعه داده های کوچک ما مشکلی نیست، اما معمولاً مجموعه داده ها بسیار بزرگتر هستند و آموزش ممکن است زمان زیادی را ببرد.

ایده RandomizedSearchCV این است که مانند GridSearchCV کار می کند، اما به جای اینکه همه ترکیب ها را امتحان کند، یک زیرمجموعه نمونه تصادفی را امتحان می کند. معمولاً منجر به نتیجه کمی بدتر می شود، اما بسیار سریعتر است.
با استفاده از آرگومان n_iter (به طور پیش فرض روی 10 تنظیم شده است) می توانید تعداد ترکیب های مورد آزمایش را کنترل کنید. جدای از آن، کار با آن مانند GridSearchCV است.
گاهی اوقات به دلیل وجود بهترین پارامترها در بین ترکیبات نمونه برداری شده توسط RandomizedSearchCV ، امتیازها می توانند یکسان باشند.

Your task is to build GridSearchCV and RandomizedSearchCV with 20 combinations and compare the results :

1. **Initialize the RandomizedSearchCV object. Pass the parameters grid and set the number of combinations to 20.**
2. **Initialize the GridSearchCV object.**
3. **Train both GridSearchCV and RandomizedSearchCV objects.**
4. **Print the best estimator of grid.**
5. **Print the best score of randomized.**

پس از محاسبات با الگوریتم ها ، ما در نهایت به مدل تنظیم شده ای می رسیم که در این مجموعه خاص از نمونه ها (مجموعه آموزشی) بهترین عملکرد را دارد. ما همچنین امتیاز اعتبارسنجی متقابل را دریافت می کنیم زیرا GridSearchCV.best_score محاسبه شده را در حین یافتن بهترین هایپرپارامترها فراهم می کند.

مشکل این است که بهترین هایپرپارامترها در یک مجموعه داده خاص تضمین نمی شود که به طور کلی برای مشکل شما بهترین باشند. اگر داده های جدیدی به مجموعه داده اضافه شود – بهترین هایپرپارامترها ممکن است تغییر کنند.
بنابراین .best_score_ ممکن است بهتر از امتیاز داده‌های کاملاً دیده نشده باشد، زیرا فراپارامترهایی که در یک مجموعه داده بهترین هستند ممکن است خوب باشند اما برای داده‌های جدید بهترین نیستند.

Let's sum it all up. We need:

1. Preprocess the data.
2. Do a train-test split.
3. Find the model with the best cross-validation score on the training set.
   This includes trying several algorithms and finding the best hyperparameters for them.
   To simplify, we only used one algorithm in this course.
4. Evaluate the best model on the test set.