# فصل ۱۳- رگرسیون خطی

## ۱۳.۰ مقدمه

رگرسیون خطی یکی از ساده‌ترین الگوریتم‌های یادگیری تحت نظارت در جعبه ابزار ما است. اگر تا به حال یک دوره مقدماتی آمار را در کالج گذرانده باشید، احتمالاً آخرین موضوعی که به آن پرداخته‌اید رگرسیون خطی بوده است. در واقع، آنقدر ساده است که گاهی اوقات اصلاً به عنوان یادگیری ماشین در نظر گرفته نمی‌شود! باور داشته باشید یا خیر، واقعیت این است که رگرسیون خطی - و بسط‌های آن - زمانی که بردار هدف، مقدار کمی‌است (به عنوان مثال، قیمت خانه، سن) یک روش رایج و مفید برای پیش‌بینی است. در این فصل ما انواع روش‌های رگرسیون خطی (و برخی پسوندها) را برای ایجاد مدل‌های پیش‌بینی با عملکرد خوب بررسی خواهیم کرد.

## ۱۳.۱ تطبیق یک خط

### مسئله

می‌خواهید مدلی را آموزش دهید که نشان دهنده رابطه خطی بین بردار ویژگی و بردار هدف باشد.

### راه‌حل

از LinearRegression در scikit-learn استفاده کنید:

```
# Load libraries
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston


# Load data with only two features
boston = load_boston()
features = boston.data[:,0:2]
target = boston.target


# Create linear regression
regression = LinearRegression()


# Fit the linear regression
model = regression.fit(features, target)
```

### بحث

رگرسیون خطی فرض می‌کند که رابطه بین ویژگی‌ها و بردار هدف تقریباً خطی است. یعنی اثر (که ضریب، وزن یا پارامتر نیز نامیده می‌شود) ویژگی‌ها بر بردار هدف ثابت است. در راه حل ما، برای توضیح، مدل خود را تنها با استفاده از دو ویژگی آموزش داده‌ایم. این بدان معناست که مدل خطی ما به صورت زیر خواهد بود:

$$\hat{y} = \widehat{\beta_0} + \widehat{\beta_1} x_1 + \widehat{\beta_2} x_2 + \in$$

که در آن $\hat{y}$ هدف ما است، $x$ داده برای یک ویژگی واحد، $\hat{\beta}_1$ و $\hat{\beta}_2$ ضرایبی هستند که با برازش[1] مدل شناسایی می‌شوند، و $\in$ خطا است.

پس از اینکه مدل خود را متناسب کردیم، می‌توانیم مقدار هر پارامتر را مشاهده کنیم. به عنوان مثال، β^0، که به آن سوگیری(bias) یا رهگیری(intercept) نیز گفته می‌شود، می‌تواند با استفاده از intercept_ مشاهده شود:

```
# View the intercept
model.intercept
```

```
22.46681692105723
```

و $\hat{\beta}_1$ و $\hat{\beta}_2$ به عنوان coef_ نشان داده می‌شوند:

```
# View the feature coefficients
model.coef
```

```
array([-0.34977589, 0.11642402])
```

در مجموعه داده (دیتاست) ما، مقدار هدف، ارزش متوسط یک خانه در بوستون (در دهه ۱۹۷۰) به هزار دلار است. بنابراین قیمت خانه اول در مجموعه داده عبارت است از:

```
# First value in the target vector multiplied by 1000
target[0]*1000
```

```
24000.0
```

که با استفاده از روش پیش‌بینی، می‌توانیم مقداری را برای این خانه پیش‌بینی کنیم:

```
# Predict the target value of the first observation,
multiplied by 1000
model.predict(features)[0]*1000
```

```
24560.23872370844
```

بد نیست! مدل ما فقط ۵۶۰.۲۴ دلار تخفیف(تفاوت) داشت!

مزیت اصلی رگرسیون خطی تفسیرپذیری آن است، تا حد زیادی به این دلیل که ضرایب مدل، اثر یک تغییر یک واحدی بر بردار هدف هستند. به عنوان مثال، اولین ویژگی در راه حل ما تعداد جرایم به ازای هر ساکن است. ضریب مدل ما برای این

[1] - fitting

ویژگی 0.35 – ~ بود، به این معنی که اگر این ضریب را در ۱۰۰۰ ضرب کنیم (از آنجایی که بردار هدف قیمت خانه به هزار دلار است)، تغییر قیمت خانه را برای هر جرم سرانه اضافی خواهیم داشت:

```
# First coefficient multiplied by 1000
model.coef_[0]*1000
```

```
-349.77588707748947
```

این عبارت می‌گوید که سرانه هر جنایت، قیمت خانه را تقریباً ۳۵۰ دلار کاهش می‌دهد!

## ۱۳.۲ مدیریت تاثیرات تعاملی

### مسئله

شما یک ویژگی دارید که تأثیر آن بر متغیر هدف، به ویژگی دیگری بستگی دارد.

### راه‌حل

با استفاده از ویژگی‌های چند جمله‌ای scikit-learn، یک اصطلاح تعاملی[۲] ایجاد کنید تا این وابستگی را نشان دهد:

```
# Load libraries
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston
from sklearn.preprocessing import PolynomialFeatures

# Load data with only two features
boston = load_boston()
features = boston.data[:,0:2]
target = boston.target

# Create interaction term
interaction = PolynomialFeatures(
degree=3, include_bias=False, interaction_only=True)
features_interaction = interaction.fit_transform(features)

# Create linear regression
regression = LinearRegression()

# Fit the linear regression
model = regression.fit(features_interaction, target)
```

### بحث

[2] - Interaction term

گاهی اوقات تأثیر یک ویژگی بر متغیر هدف ما حداقل تا حدی به ویژگی دیگری وابسته است. به عنوان مثال، یک مثال ساده مبتنی بر قهوه را تصور کنید که در آن ما دو ویژگی دوتایی داریم - وجود شکر (قند) و اینکه آیا هم زده‌ایم (هم‌زده) یا نه - و می‌خواهیم پیش‌بینی کنیم که آیا قهوه طعم شیرینی دارد یا خیر. فقط گذاشتن شکر در قهوه (شکر=۱، هم‌زدن=۰) طعم قهوه را شیرین نمی‌کند (همه شکر تَهِ لیوان است!) و فقط هم‌زدن قهوه بدون افزودن شکر (شکر=۰، هم زده=۱) هم آن را شیرین نمی‌کند در عوض، این تعامل بین قرار دادن شکر در قهوه و هم‌زدن قهوه (شکر=۱، هم‌زدن=۱) است که طعم قهوه را شیرین می‌کند. تاثیر شکر و هم زدن بر شیرینی به یکدیگر بستگی دارد. در این مورد می‌گوییم اثر متقابلی بین ویژگی‌های شکر و هم‌زدن وجود دارد.

ما می‌توانیم اثرات متقابل را با گنجاندن یک ویژگی جدید که حاصل ارزش‌های متناظر از ویژگی‌های متقابل را تشکیل می‌دهد، توضیح دهیم:

$$\hat{y} = \widehat{\beta_0} + \widehat{\beta_1} x_1 + \widehat{\beta_2} x_2 + \widehat{\beta_3} x_1 x_2 + \epsilon$$

که در آن $x_1$ و $x_2$ به ترتیب مقادیر شکر و هم‌زدن هستند و $x_1 x_2$ نشان دهنده تعامل بین این دو است.

در این راه حل، ما از مجموعه داده ای استفاده کردیم که فقط شامل دو ویژگی بود. در اینجا مقادیر اولین مشاهده برای هر یک از آن ویژگی‌ها آمده است:

```
# View the feature values for first observation
features[0]
```

```
array([ 6.32000000e-03, 1.80000000e+01])
```

برای ایجاد یک عبارت تعاملی، ما به سادگی آن دو مقدار را برای هر مشاهده ضرب می‌کنیم:

```
# Import library
import numpy as np

# For each observation, multiply the values of the first and
second feature
interaction_term = np.multiply(features[:, 0], features[:,1])
```

سپس می‌توانیم عبارت تعامل برای اولین مشاهده را مشاهده کنیم:

```
# View interaction term for first observation
interaction_term[0]
```

```
0.11376
```

با این حال، در حالی که اغلب ما دلیلی اساسی برای باور وجود تعامل بین دو ویژگی خواهیم داشت، گاهی اوقات چنین نیست. در این موارد استفاده از ویژگی‌های چند جمله ای (PolynomialFeatures) در scikit-learn برای ایجاد اصطلاحات تعاملی برای همه‌ی ترکیب‌های ویژگی‌ها می‌تواند مفید باشد. سپس می‌توانیم از استراتژی‌های انتخاب مدل برای شناسایی ترکیبی از ویژگی‌ها و اصطلاحات تعاملی که بهترین مدل را تولید می‌کنند، استفاده کنیم.

برای ایجاد اصطلاحات تعاملی با استفاده از PolynomialFeatures، سه پارامتر مهم وجود دارد که باید تنظیم کنیم. مهم‌تر از همه، interaction_only=True به PolynomialFeatures می‌گوید که فقط عبارات تعامل را برگرداند (و نه ویژگی‌های چند جمله‌ای، که در دستور العمل ۱۳.۳ در مورد آن صحبت خواهیم کرد). به طور پیش‌فرض، PolynomialFeatures ویژگی‌هایی به نام سوگیری را اضافه می‌کند. ما می‌توانیم با include_bias=False از آن جلوگیری کنیم. در نهایت، پارامترِ درجه، حداکثر تعداد ویژگی‌ها را برای ایجاد عبارت‌های تعاملی تعیین می‌کند (این مورد در صورتی استفاده می‌شود که می‌خواهیم یک عبارت تعاملی ایجاد کنیم که آن عبارت تعاملی ترکیبی از سه ویژگی باشد). ما می‌توانیم خروجی PolynomialFeatures از راه‌حل خود را با بررسی اینکه آیا مقادیر ویژگی اولین مشاهده و مقدار مدت تعامل با نسخه محاسبه‌شده دستی ما مطابقت دارد یا خیر، ببینیم:

```
# View the values of the first observation
features_interaction[0]
```

```
array([ 6.32000000e-03, 1.80000000e+01, 1.13760000e-01])
```

## ۱۳.۳ برازش یک رابطه غیرخطی

### مسئله

شما می‌خواهید یک رابطه غیر خطی را مدل کنید.

### راه‌حل

یک رگرسیون چند جمله‌ای با گنجاندن ویژگی‌های چند جمله‌ای در مدل رگرسیون خطی ایجاد کنید:

```
# Load library
from sklearn.linear_model import LinearRegression
from sklearn.datasets import load_boston
from sklearn.preprocessing import PolynomialFeatures

# Load data with one feature
boston = load_boston()
features = boston.data[:,0:1]
target = boston.target

# Create polynomial features x^2 and x^3
polynomial = PolynomialFeatures(degree=3,
include_bias=False)
features_polynomial = polynomial.fit_transform(features)

# Create linear regression
regression = LinearRegression()

# Fit the linear regression
model = regression.fit(features_polynomial, target)
```

**بحث**

تا اینجا ما فقط در مورد مدل‌سازی روابط خطی بحث کردیم. نمونه ای از یک رابطه خطی می‌تواند تعداد طبقات یک ساختمان و ارتفاع ساختمان باشد. در رگرسیون خطی، ما تأثیر تعداد طبقات و ارتفاع ساختمان را تقریباً ثابت فرض می‌کنیم، به این معنی که ارتفاع یک ساختمان ۲۰ طبقه تقریباً دو برابر یک ساختمان ۱۰ طبقه است که تقریباً دو برابر بلندتر از یک ساختمان ۵ طبقه است. با این حال، بسیاری از روابط واقعی کاملاً خطی نیستند.

اغلب ما می‌خواهیم یک رابطه غیرخطی را مدل کنیم - برای مثال، رابطه بین تعداد ساعات مطالعه یک دانش‌آموز و نمره‌ای که در آزمون می‌گیرد. به طور شهودی، می‌توانیم تصور کنیم که تفاوت زیادی در نمرات آزمون بین دانش‌آموزانی که یک ساعت مطالعه می‌کنند در مقایسه با دانش‌آموزانی که اصلا مطالعه نکرده‌اند وجود دارد. با این حال، بین دانش‌آموزی که ۹۹ ساعت درس خوانده و دانش‌آموزی که ۱۰۰ ساعت مطالعه کرده است، تفاوت بسیار کمتری در نمرات آزمون وجود دارد. تأثیر یک ساعت مطالعه بر نمره آزمون دانش‌آموز با افزایش تعداد ساعات کاهش می‌یابد.

رگرسیون چند جمله‌ای توسعه‌ای از رگرسیون خطی است تا به ما امکان مدل‌سازی روابط غیرخطی را بدهد. برای ایجاد یک رگرسیون چند جمله‌ای، تابع خطی را که در دستور العمل ۱۳.۱ استفاده کردیم تبدیل کنید:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \epsilon$$

که با افزودن ویژگی‌های چند جمله‌ای به یک تابع چند جمله‌ای می‌رسیم:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_1 x_1^2 + \cdots + \hat{\beta}_d x_i^d + \epsilon$$

که در آن d درجه چند جمله‌ای است. چگونه می‌توانیم از رگرسیون خطی برای یک تابع غیرخطی استفاده کنیم؟ پاسخ این است که ما نحوه تناسب رگرسیون خطی با مدل را تغییر نمی‌دهیم، بلکه فقط ویژگی‌های چند جمله ای را به آن اضافه می‌کنیم. یعنی رگرسیون خطی «نمی‌داند» که $x^2$ تبدیل درجه دوم $x$ است. فقط آن را یک متغیر دیگر در نظر می‌گیرد.

ممکن است توضیح کاربردی تری لازم باشد. برای مدل‌سازی روابط غیرخطی، می‌توانیم ویژگی‌های جدیدی ایجاد کنیم که یک ویژگی موجود، $x$ را تا یک مقدار توانی افزایش می‌دهد: $x^2$، $x^3$، و غیره. هر چه بیشتر از این ویژگی‌های جدید اضافه کنیم، «خط» ایجاد شده توسط مدل ما انعطاف پذیرتر است. برای واضح تر شدن این موضوع، تصور کنید که می‌خواهیم یک چند جمله‌ای تا درجه سوم ایجاد کنیم. به منظور سادگی، ما فقط بر روی یک مشاهده (اولین مشاهده در مجموعه داده) تمرکز خواهیم کرد، $x_0$:

```
# View first observation
features[0]
```

```
array([ 0.00632])
```

برای ایجاد یک ویژگی چند جمله ای، مقدار مشاهده اول را به درجه دوم یعنی $x_1^2$ افزایش می‌دهیم:

```
# View first observation raised to the second power, x^2
features[0]**2
```

```
array([ 3.99424000e-05])
```

این ویژگی جدید ما خواهد بود. سپس مقدار مشاهده اول را به درجه سوم یعنی $x_1^3$ نیز افزایش می‌دهیم:

```
# View first observation raised to the third power, x^3
features[0]**3
```

```
array([ 2.52435968e-07])
```

با گنجاندن هر سه ویژگی ($x$، $x^2$ و $x^3$) در ماتریس ویژگی‌ها و سپس اجرای یک رگرسیون خطی، یک رگرسیون چند جمله‌ای انجام داده‌ایم:

```
# View the first observation's values for x, x^2, and x^3
features_polynomial[0]
```

```
array([ 6.32000000e-03, 3.99424000e-05, 2.52435968e-07])
```

PolynomialFeatures دو پارامتر مهم دارد. ابتدا، درجه حداکثر تعداد درجه را برای ویژگی‌های چند جمله ای تعیین می‌کند. برای مثال degree = 3، $x^2$ و $x^3$ را ایجاد می‌کند. در نهایت، به طور پیش‌فرض، PolynomialFeatures شامل یک ویژگی حاوی تنها یک‌ها[3] می‌شود (به نام سوگیری). ما می‌توانیم آن را با تنظیم include_bias=False حذف کنیم.

## ۱۳.۴ کاهش واریانس با منظم سازی

### مسئله

شما می‌خواهید واریانس مدل رگرسیون خطی خود را کاهش دهید.

### راه‌حل

از یک الگوریتم یادگیری استفاده کنید که شامل جریمه انقباض[4] (همچنین منظم سازی) مانند رگرسیون رج ( ridge regression) و رگرسیون کمند (lasso regression) است:

[3] - ones
[4] - shrinkage penalty

```
# Load libraries
from sklearn.linear_model import Ridge
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler


# Load data
boston = load_boston()
features = boston.data
target = boston.target


# Standardize features
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)


# Create ridge regression with an alpha value
regression = Ridge(alpha=0.5)



# Fit the linear regression
model = regression.fit(features_standardized, target)
```

**بحث**

در رگرسیون خطی استاندارد، مدل برای به حداقل رساندن مجموع مجذور خطا بین مقادیر واقعی ($y_i$) و پیش‌بینی، ($\hat{y}_i$) مقادیر هدف، یا مجموع باقیمانده مربع‌ها (RSS) آموزش می‌بینند:

$$RSS = \sum_{i=1}^{n}(y_i + \hat{y}_i)^2$$

یادگیرندگان رگرسیون منظم مشابه هستند، با این تفاوت که سعی می‌کنند RSS و مقدار جریمه برای اندازه کل مقادیر ضرایب را به حداقل برسانند، که جریمه انقباض نامیده می‌شود؛ زیرا سعی می‌کند مدل را "کوچک" کند. دو نوع متداول از یادگیرندگان منظم برای رگرسیون خطی وجود دارد: رگرسیون ridge و lasso. تنها تفاوت رسمی‌در نوع جریمه انقباض مورد استفاده است. در رگرسیون ridge، جریمه انقباض یک فراپارامتر تنظیم ضرب در مجذور همه ضرایب است:

$$RSS + \alpha\sum_{j=1}^{p}\hat{\beta}_j^2$$

که در آن $\beta_j$ ضریب $j$ ام ویژگی p و α یک فراپارامتر است (در ادامه بحث می‌شود). Lasso نیز مشابه همین است، با این تفاوت که جریمه انقباض یک فراپارامتر تنظیم ضرب در مجموع قدر مطلق همه ضرایب است:

$$\frac{1}{2n}RSS + \alpha\sum_{j=1}^{p}\beta_j$$

که در آن n تعداد مشاهدات است. پس از کدام یک استفاده کنیم؟ به عنوان یک قاعده کلی ، رگرسیون ridge اغلب پیش‌بینی‌های کمی‌بهتری نسبت به lasso ایجاد می‌کند، اما lasso (به دلایلی که در دستور العمل ۱۳.۵ در مورد آن صحبت

خواهیم‌کرد) مدل‌های قابل تفسیرتری تولید می‌کند. اگر بخواهیم بین توابع پنالتی ridge و lasso تعادل برقرار کنیم، می‌توانیم از شبکه الاستیک (elastic net) استفاده کنیم که به سادگی یک مدل رگرسیونی است که هر دو پنالتی را شامل می‌شود. صرف نظر از اینکه از کدام یک استفاده می‌کنیم، هر دو رگرسیون ridge و lasso می‌توانند مدل‌های بزرگ یا پیچیده را با گنجاندن مقادیر ضرایب در تابع ضرری که در تلاش برای به حداقل رساندن آن هستیم، جریمه کنند.

فراپارامتر $\alpha$، به ما اجازه می‌دهد تا میزان جریمه کردن ضرایب را با استفاده از مقادیر بالای ساخت مدل‌های ساده کنترل کنیم. مقدار ایده آل $\alpha$ باید مانند هرهایپرپارامتر دیگری تنظیم شود. در scikit-learn، $\alpha$ با استفاده از پارامتر آلفا تنظیم می‌شود.

scikit-learn شامل یک روش RidgeCV است که به ما امکان می‌دهد مقدار ایده آل $\alpha$ را انتخاب کنیم:

```
# Load library
from sklearn.linear_model import RidgeCV

# Create ridge regression with three alpha values
regr_cv = RidgeCV(alphas=[0.1, 1.0, 10.0])

# Fit the linear regression
model_cv = regr_cv.fit(features_standardized, target)

# View coefficients
model_cv.coef_
```

```
array([-0.91215884, 1.0658758 , 0.11942614, 0.68558782, -2.03231631,
2.67922108, 0.01477326, -3.0777265 , 2.58814315, -2.00973173,
-2.05390717, 0.85614763, -3.73565106])
```

سپس می‌توانیم به راحتی مقدار $\alpha$ برای بهترین مدل را مشاهده کنیم:

```
# View alpha
model_cv.alpha_
```

```
1.0
```

نکته پایانی: چون در رگرسیون خطی مقدار ضرایب تا حدی توسط مقیاس ویژگی تعیین می‌شود و در مدل‌های منظم شده همه ضرایب با هم جمع می‌شوند، باید قبل از آموزش از استانداردسازی ویژگی اطمینان حاصل کنیم.

## ۱۳.۵ کاهش ویژگی‌ها با رگرسیون lasso

### مسئله

شما می‌خواهید مدل رگرسیون خطی خود را با کاهش تعداد ویژگی‌ها ساده کنید.

### راه‌حل

از رگرسیون lasso استفاده کنید:

```
# Load library
from sklearn.linear_model import Lasso
from sklearn.datasets import load_boston
from sklearn.preprocessing import StandardScaler

# Load data
boston = load_boston()
features = boston.data
target = boston.target

# Standardize features
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)

# Create lasso regression with alpha value
regression = Lasso(alpha=0.5)

# Fit the linear regression
model = regression.fit(features_standardized, target)
```

**بحث**

یکی از ویژگی‌های جالب جریمه رگرسیون lasso این است که می‌تواند ضرایب یک مدل را به صفر کاهش دهد و به طور موثر تعداد ویژگی‌های مدل را کاهش دهد. به عنوان مثال، در حل ما $\alpha$ را روی 0.5 قرار می‌دهیم و می‌بینیم که بسیاری از ضرایب 0 هستند، به این معنی که ویژگی‌های مربوط به آنها در مدل استفاده نمی‌شوند:

```
# View coefficients
model.coef_
```

```
array([-0.10697735, 0. , -0. , 0.39739898, -0. , 2.97332316, -0. ,
-0.16937793, -0. , -0. , -1.59957374, 0.54571511, -3.66888402])
```

با این حال، اگر $\alpha$ را به مقدار بسیار بالاتری افزایش دهیم، می‌بینیم که به معنای واقعی کلمه از هیچ یک از ویژگی‌ها استفاده نمی‌شود:

```
# Create lasso regression with a high alpha
regression_a10 = Lasso(alpha=10)
model_a10 = regression_a10.fit(features_standardized,
target)
model_a10.coef_
```

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

مزیت عملی این اثر این است که ما می‌توانیم ۱۰۰ ویژگی را در ماتریس ویژگی‌های خود بگنجانیم و سپس با تنظیم فراپارامتر $\alpha$ برای lasso، مدلی تولید کنیم که تنها از ۱۰ (مثلا) ویژگی از مهمترین ویژگی‌ها استفاده می‌کند. این روش به ما

امکان می‌دهد واریانس را کاهش دهیم و در عین حال تفسیرپذیری مدل خود را بهبود ببخشیم (زیرا ویژگی‌های کمتری برای توضیح آسان‌تر است).