**فصل 10. کاهش ابعاد با استفاده از انتخاب ویژگی**

**10.0 مقدمه**

در فصل 9، ما در مورد چگونگی کاهش ابعاد ماتریس ویژگی خود با ایجاد ویژگی‌های جدید (ایده‌آل) با توانایی مشابه برای آموزش مدل‌های با کیفیت، اما با تعداد ابعاد قابل توجه کمتر، صحبت کردیم. این به عنوان استخراج ویژگی شناخته می‌شود. در این فصل، ما رویکرد جایگزینی را پوشش می‌دهیم: انتخاب ویژگی‌های با کیفیت و اطلاعاتی بالا و حذف ویژگی‌های کم‌اهمیت، که به عنوان انتخاب ویژگی شناخته می‌شود.

سه نوع اصلی از روش‌های انتخاب ویژگی وجود دارند: روش‌های فیلتر، بسته‌بندی و تعبیه‌ شده. روش‌های‌ فیلتری، بهترین ویژگی‌ها را با بررسی ویژگی‌ها بر اساس خصوصیات آماری آن‌ها انتخاب می‌کنند. روش‌های بسته‌بندی از آزمون و خطا استفاده می‌کنند تا زیرمجموعه‌ای از ویژگی‌ها را که مدل‌های با دقت بالاتری را پیش‌بینی می‌کنند، پیدا کنند. در نهایت، روش‌های تعبیه‌شده به عنوان بخشی یا به عنوان گسترشی از فرآیند آموزش الگوریتم یادگیری، زیرمجموعه بهترین ویژگی‌ها را انتخاب می‌کنند.

در حالت ایده آل، ما هر سه روش را در این فصل شرح می دهیم. با این حال، از آنجایی که روش‌های تعبیه‌شده با الگوریتم‌های یادگیری خاص در هم تنیده هستند، توضیح آن‌ها قبل از بررسی عمیق‌تر خود الگوریتم‌ها دشوار است. بنابراین، در این فصل ما فقط روش‌های انتخاب ویژگی فیلتر و بسته‌بندی را پوشش می‌دهیم، و در مورد روش‌های تعبیه‌شده خاص تا فصل‌هایی که در آن الگوریتم‌های یادگیری به طور عمیق‌تر مورد بررسی قرار می‌گیرند، بحث نمی‌کنیم.

**10.1 آستانه‌گذاری بر اساس واریانس ویژگی‌های عددی**

**مسئله**

شما یک مجموعه از ویژگی‌های عددی دارید و می‌خواهید ویژگی‌های با واریانس کم را حذف کنید. (به عبارت دیگر حذف ویژگی‌هایی که کمترین اطلاعات را شامل می‌شوند)

**راه‌حل:**

برای انتخاب زیرمجموعه‌ای از ویژگی‌ها که واریانس آن‌ها بالاتر از آستانه‌ای مشخص باشد، می‌توانید این روش را دنبال کنید:

# Load libraries

**from** **sklearn** **import** datasets

**from** **sklearn.feature\_selection** **import** VarianceThreshold

# import some data to play with

iris = datasets.load\_iris()

# Create features and target

features = iris.data

target = iris.target

# Create thresholder

thresholder = VarianceThreshold(threshold=.5)

# Create high variance feature matrix

features\_high\_variance = thresholder.fit\_transform(features)

# View high variance feature matrix

features\_high\_variance[0:3]

array([[ 5.1, 1.4, 0.2],

[ 4.9, 1.4, 0.2],

[ 4.7, 1.3, 0.2]])

**بحث:**

آستانه واریانس (VT) یکی از اساسی‌ترین رویکردها برای انتخاب ویژگی است. ایده آن این است که ویژگی‌های با واریانس کم نسبت به ویژگی‌های با واریانس بالا کمتر مفید هستند. VT ابتدا واریانس هر ویژگی را محاسبه می کند:

در رابطه بالا بردار ویژگی است، مقدار هر ویژگی به تنهایی است و مقدار میانگین آن ویژگی است. در مرحله بعد، تمام ویژگی‌هایی را که واریانس آن‌ها به مقدار آستانه نمی‌رسد حذف می‌کند.

هنگام استفاده از VT باید به دو نکته توجه داشت. اول، واریانس در مرکز نیست. یعنی در واحد مجذور خود ویژگی است. بنابراین، زمانی که مجموعه ویژگی‌ها دارای واحدهای مختلف باشند، VT کار نخواهد کرد (به عنوان مثال، واحد یک ویژگی سال است در حالی که واحد ویژگی متفاوت دیگری به دلار است). دوم، مقدار آستانه واریانس به صورت دستی انتخاب می‌شود، بنابراین باید از نظرات خود برای انتخاب یک مقدار خوب استفاده کنیم (یا از تکنیک انتخاب مدل که در فصل 12 توضیح داده شده است استفاده کنیم). ما می توانیم واریانس هر ویژگی را با استفاده از variances\_ ببینیم:

# View variances

thresholder.fit(features).variances\_

array([ 0.68112222, 0.18675067, 3.09242489, 0.57853156])

در نهایت، اگر ویژگی‌ها استاندارد شده باشند (به معنای میانگین صفر و واریانس یک)، به دلایل واضح آستانه واریانس به درستی کار نخواهد کرد:

# Load library

**from** **sklearn.preprocessing** **import** StandardScaler

# Standardize feature matrix

scaler = StandardScaler()

features\_std = scaler.fit\_transform(features)

# Caculate variance of each feature

selector = VarianceThreshold()

selector.fit(features\_std).variances\_

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

**10.2 آستانه‌گذاری بر اساس واریانس ویژگی‌های دودویی**

**مسئله**

شما مجموعه‌ای از ویژگی‌های رسته‌ای[[1]](#footnote-1) دودویی دارید و می‌خواهید ویژگی‌هایی که واریانس کم دارند (به عبارت دیگر، احتمالاً حاوی اطلاعات کمی هستند) را حذف کنید.

**راه‌حل:**

انتخاب زیرمجموعه‌ای از ویژگی‌ها با واریانس متغیر تصادفی برنولی بالاتر از یک آستانه داده‌ شده:

# Load library

**from** **sklearn.feature\_selection** **import** VarianceThreshold

# Create feature matrix with:

# Feature 0: 80% class 0

# Feature 1: 80% class 1

# Feature 2: 60% class 0, 40% class 1

features = [[0, 1, 0],

[0, 1, 1],

[0, 1, 0],

[0, 1, 1],

[1, 0, 0]]

# Run threshold by variance

thresholder = VarianceThreshold(threshold=(.75 \* (1 - .75)))

thresholder.fit\_transform(features)

array([[0],

[1],

[0],

[1],

[0]])

مشابه ویژگی‌های عددی، یک استراتژی برای انتخاب ویژگی‌های رسته‌ای با اطلاعات زیاد این است که واریانس آن‌ها را مورد بررسی قرار دهیم. در ویژگی‌های دودویی (یعنی متغیرهای تصادفی برنولی)، واریانس به صورت زیر محاسبه می‌شود:

که در آن نسبت تعداد مشاهدات کلاس ۱ را نشان می‌دهد. بنابراین با تنظیم مقدار ، می‌توانیم ویژگی‌هایی که در آن بیشترین تعداد مشاهدات به یک کلاس تعلق دارند را حذف کنیم، به عبارت دیگر ویژگی‌هایی که تنوع مقدار در آن‌ها کم است و عملا حاوی اطلاعات کمی هستند را حذف می‌کنیم.

**10.3 مدیریت ویژگی‌های با همبستگی بالا**

**مسئله**

شما یک ماتریس ویژگی دارید و شک دارید که برخی از ویژگی‌ها همبستگی بالایی دارند.

**راه‌حل:**

برای بررسی ویژگی‌های با همبستگی بالا، از یک ماتریس همبستگی استفاده کنید. اگر ویژگی‌های با همبستگی بالا وجود داشته باشند، در نظر داشته باشید یکی از ویژگی‌های مرتبط را حذف نمایید:

# Load libraries

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

# Create feature matrix with two highly correlated features

features = np.array([[1, 1, 1],

[2, 2, 0],

[3, 3, 1],

[4, 4, 0],

[5, 5, 1],

[6, 6, 0],

[7, 7, 1],

[8, 7, 0],

[9, 7, 1]])

# Convert feature matrix into DataFrame

dataframe = pd.DataFrame(features)

# Create correlation matrix

corr\_matrix = dataframe.corr().abs()

# Select upper triangle of correlation matrix

upper = corr\_matrix.where(np.triu(np.ones(corr\_matrix.shape),

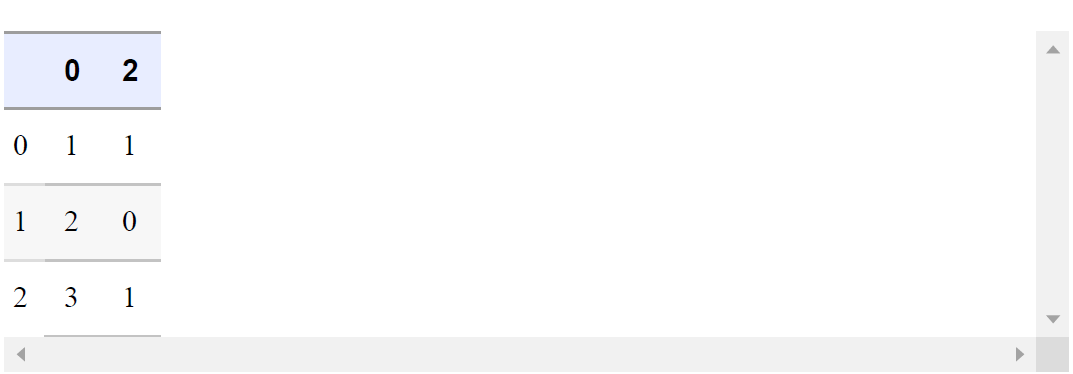
k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.95

to\_drop = [column **for** column **in** upper.columns **if** any(upper[column] > 0.95)]

# Drop features

dataframe.drop(dataframe.columns[to\_drop], axis=1).head(3)



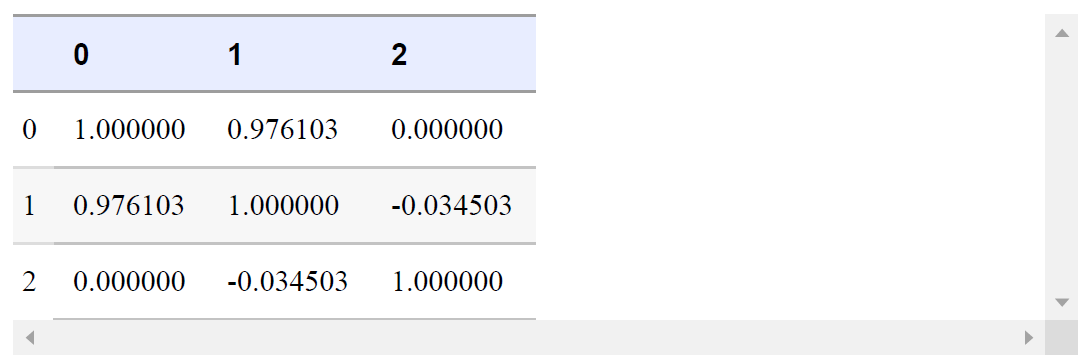
**بحث:**

یکی از مشکلاتی که اغلب در یادگیری ماشین به آن برخورد می‌کنیم، ویژگی‌های با همبستگی بالاست. اگر دو ویژگی با همبستگی بالا باشند، اطلاعاتی که در آنها وجود دارد بسیار شبیه به هم هستند و احتمالاً یکی از این دو ویژگی اضافه است. برای ویژگی‌های با همبستگی بالا راه‌حل بسیار ساده است: یکی از آن‌ها را از مجموعه ویژگی‌ها حذف کنید.

در راه‌حل پیشنهادی ما، ابتدا یک ماتریس همبستگی برای تمام ویژگی‌ها ایجاد می‌کنیم:

# Correlation matrix

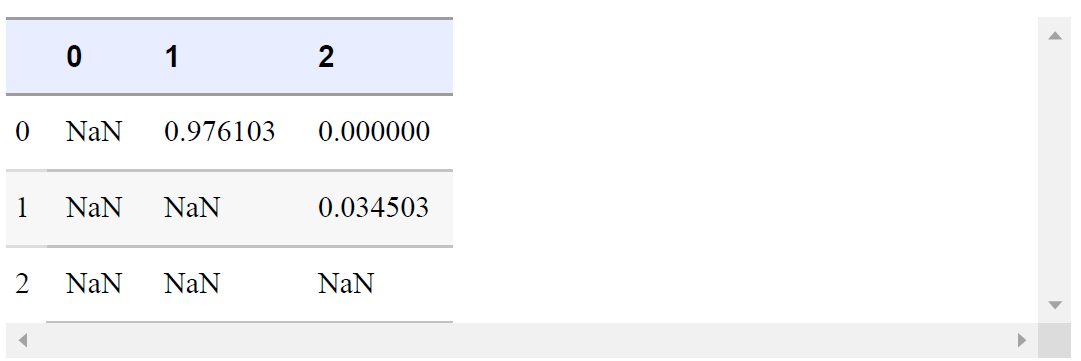
dataframe.corr()



در مرحله دوم، به بخش بالایی ماتریس همبستگی نگاه می‌کنیم تا جفت ویژگی‌های با همبستگی بالا را شناسایی کنیم:

# Upper triangle of correlation matrix

Upper



در مرحله سوم، یکی از ویژگی‌ها را از هر جفت از این ویژگی‌ها از مجموعه ویژگی‌ها حذف می‌کنیم.

**10.4 حذف ویژگی‌های بی‌اهمیت برای دسته‌بندی**

**مسئله**

شما یک بردار هدف رسته‌ای دارید و می‌خواهید ویژگی‌های بدون اطلاعات را حذف کنید.

**راه‌حل:**

اگر ویژگی‌ها رسته‌ای هستند، توزیع کای مربع آماری ( ) بین هر ویژگی و بردار هدف را محاسبه کنید:

# Load libraries

**from** **sklearn.datasets** **import** load\_iris

**from** **sklearn.feature\_selection** **import** SelectKBest

**from** **sklearn.feature\_selection** **import** chi2, f\_classif

# Load data

iris = load\_iris()

features = iris.data

target = iris.target

# Convert to categorical data by converting data to integers

features = features.astype(int)

# Select two features with highest chi-squared statistics

chi2\_selector = SelectKBest(chi2, k=2)

features\_kbest = chi2\_selector.fit\_transform(features, target)

# Show results

**print**("Original number of features:", features.shape[1])

**print**("Reduced number of features:", features\_kbest.shape[1])

Original number of features: 4

Reduced number of features: 2

اگر ویژگی‌ها ترتیبی باشند، مقدار ANOVA F-Values بین هر ویژگی و بردار هدف را محاسبه کنید:

# Select two features with highest F-values

fvalue\_selector = SelectKBest(f\_classif, k=2)

features\_kbest = fvalue\_selector.fit\_transform(features, target)

# Show results

**print**("Original number of features:", features.shape[1])

**print**("Reduced number of features:", features\_kbest.shape[1])

Original number of features: 4

Reduced number of features: 2

به جای انتخاب تعداد خاصی از ویژگی‌ها، می‌توانیم از تابع SelectPercentile (انتخاب ویژگی‌ها بر اساس درصدی از بالاترین امتیازها) استفاده کنیم تا n درصد بالایی از ویژگی‌ها را انتخاب کنیم:

# Load library

**from** **sklearn.feature\_selection** **import** SelectPercentile

# Select top 75% of features with highest F-values

fvalue\_selector = SelectPercentile(f\_classif, percentile=75)

features\_kbest = fvalue\_selector.fit\_transform(features, target)

# Show results

**print**("Original number of features:", features.shape[1])

**print**("Reduced number of features:", features\_kbest.shape[1])

Original number of features: 4

Reduced number of features: 3

**بحث:**

توزیع کای مربع آماری (Chi-square) استقلال دو بردار ویژگی از نوع رسته‌ای را بررسی می‌کند. به عبارت دیگر، آمار بین تفاوت تعداد نمونه‌های مشاهده شده از هر کلاس از یک ویژگی رسته‌ای و چیزی که ما انتظار داشتیم که در صورت مستقل بودن ویژگی از ویژگی هدف مشاهده کنیم:

در این رابطه تعداد مشاهدات در کلاس و تعداد مشاهدات مورد انتظار در کلاس است در صورتی که هیچ ارتباطی بین ویژگی و بردار هدف وجود نداشته باشد.

توزیع کای مربع آماری یک عدد تکی است که به شما می‌گوید در صورتی که هیچ ارتباطی به طور کامل در نمونه‌ها وجود نداشته باشد، چه تفاوتی بین تعداد‌ مشاهدات و تعدادی که انتظار دارید وجود دارد. با محاسبه توزیع کای مربع آماری بین یک ویژگی و بردار هدف، یک اندازه‌گیری از استقلال بین دو متغیر به دست می‌آوریم. اگر هدف مستقل از ویژگی باشد، آن ویژگی برای ما بی‌اهمیت است، زیرا هیچ اطلاعاتی که می‌توانیم برای دسته‌بندی استفاده کنیم، ندارد. از سوی دیگر، اگر دو ویژگی به طور قابل مشاهده‌ای وابسته باشند، احتمالاً برای آموزش مدل ما اطلاعات زیادی دارند و برای آموزش مدل بسیار مفید هستند.

برای استفاده از توزیع کای مربع آماری در انتخاب ویژگی، کای-مربع را بین هر ویژگی و بردار هدف محاسبه کرده و سپس ویژگی‌هایی را با بهترین توزیع کای مربع آماری انتخاب می‌کنیم. با استفاده از کتابخانه scikit-learn در پایتون، می‌توانیم از تابع SelectKBest برای انتخاب ویژگی‌های با بهترین توزیع کای مربع آماری استفاده کنیم. پارامتر k تعداد ویژگی‌هایی که می‌خواهیم نگه داریم را تعیین می‌کند.

مهم است به یاد داشته باشید که توزیع کای مربع آماری تنها بین دو بردار رسته‌ای قابل محاسبه است. به همین دلیل، استفاده از توزیع کای مربع آماری برای انتخاب ویژگی نیازمند این است که هر دو بردار هدف و ویژگی‌ها از نوع رسته‌ای باشند. با این حال، اگر ویژگی عددی داریم، با تبدیل ویژگی کمی به یک ویژگی رسته‌ای می‌توانیم از تکنیک توزیع کای مربع آماری استفاده کنیم. در نهایت، برای استفاده از رویکرد توزیع کای مربع، تمام مقادیر باید غیرمنفی باشند.

به عنوان جایگزین، اگر ویژگی‌ها عددی باشند، می‌توانیم از تابع f\_classif استفاده کنیم تا مقدار ANOVA F-value را بین هر ویژگی و بردار هدف محاسبه کنیم. امتیازهای F-value بررسی می‌کنند که آیا وقتی که ویژگی عددی را بر اساس بردار هدف گروه‌بندی می‌کنیم، میانگین هر گروه به طور معناداری متفاوت است یا خیر. به عنوان مثال، اگر بردار هدف باینری (مانند جنسیت) و ویژگی کمی (مثل نمرات آزمون) داشته باشیم، امتیاز F-value به ما می‌گوید که آیا میانگین نمرات آزمون مردان با میانگین نمرات آزمون زنان متفاوت است یا خیر. اگر تفاوت وجود نداشته باشد، نمره آزمون به ما در پیش‌بینی جنسیت کمک نمی‌کند و بنابراین این ویژگی بی‌اهمیت است.

**10.5 حذف ویژگی‌ها به صورت بازگشتی**

**مسئله**

شما می‌خواهید به طور خودکار بهترین ویژگی‌ها را برای نگه‌داری انتخاب کنید.

**راه‌حل:**

با استفاده از تابع RFECV از کتابخانه Scikit-learn برای انجام حذف ویژگی به صورت بازگشتی (RFE) با استفاده از اعتبارسنجی متقابل (CV) استفاده کنید. به این ترتیب، به صورت مکرر یک مدل را آموزش دهید، هر بار یک ویژگی را حذف کنید تا زمانی که عملکرد مدل (به عنوان مثال، دقت) بدتر شود. ویژگی‌های باقی‌مانده بهترین‌ها هستند:

# Load libraries

**import** **warnings**

**from** **sklearn.datasets** **import** make\_regression

**from** **sklearn.feature\_selection** **import** RFECV

**from** **sklearn** **import** datasets, linear\_model

# Suppress an annoying but harmless warning

warnings.filterwarnings(action="ignore", module="scipy",

message="^internal gelsd")

# Generate features matrix, target vector, and the true coefficients

features, target = make\_regression(n\_samples = 10000,

n\_features = 100,

n\_informative = 2,

random\_state = 1)

# Create a linear regression

ols = linear\_model.LinearRegression()

# Recursively eliminate features

rfecv = RFECV(estimator=ols, step=1, scoring="neg\_mean\_squared\_error")

rfecv.fit(features, target)

rfecv.transform(features)

array([[ 0.00850799, 0.7031277 , -1.2416911 , -0.25651883, -0.10738769],

[-1.07500204, 2.56148527, 0.5540926 , -0.72602474, -0.91773159],

[ 1.37940721, -1.77039484, -0.59609275, 0.51485979, -1.17442094],

...,

[-0.80331656, -1.60648007, 0.37195763, 0.78006511, -0.20756972],

[ 0.39508844, -1.34564911, -0.9639982 , 1.7983361 , -0.61308782],

[-0.55383035, 0.82880112, 0.24597833, -1.71411248, 0.3816852 ]])

پس از اعمال RFE، می‌توانیم تعداد ویژگی‌هایی که باید نگه داریم را مشاهده کنیم:

# Number of best features

rfecv.n\_features\_

5

همچنین می‌توانیم ببینیم که کدام یک از این ویژگی‌ها را باید نگه داریم:

# Which categories are best

rfecv.support\_

array([False, False, False, False, False, True, False, False, False,

False, False, False, False, False, False, False, False, False,

False, False, False, False, False, False, False, False, False,

False, False, False, False, False, False, False, False, False,

False, False, False, True, False, False, False, True, False,

False, False, False, False, False, False, False, False, False,

False, False, False, False, False, False, True, False, False,

False, False, False, False, False, True, False, False, False,

False, False, False, False, False, False, False, False, False,

False, False, False, False, False, False, False, False, False,

False, False, False, False, False, False, False, False, False,

False], dtype=bool)

حتی می‌توانیم رتبه‌بندی ویژگی‌ها را مشاهده کنیم:

# Rank features best (1) to worst

rfecv.ranking\_

array([11, 92, 96, 87, 46, 1, 48, 23, 16, 2, 66, 83, 33, 27, 70, 75, 29,

84, 54, 88, 37, 42, 85, 62, 74, 50, 80, 10, 38, 59, 79, 57, 44, 8,

82, 45, 89, 69, 94, 1, 35, 47, 39, 1, 34, 72, 19, 4, 17, 91, 90,

24, 32, 13, 49, 26, 12, 71, 68, 40, 1, 43, 63, 28, 73, 58, 21, 67,

1, 95, 77, 93, 22, 52, 30, 60, 81, 14, 86, 18, 15, 41, 7, 53, 65,

51, 64, 6, 9, 20, 5, 55, 56, 25, 36, 61, 78, 31, 3, 76])

**بحث:**

احتمالاً این روش پیچیده‌ترین دستورالعمل در این کتاب تا این نقطه است که ترکیبی از تعدادی موضوع است که هنوز به طور دقیق به آنها پرداخته نشده است. با این حال، درک مستقیم آن به اندازه کافی ساده است که ما می‌توانیم در اینجا به آن بپردازیم، نه اینکه تا فصل بعدی صبر کنیم. ایده پشت RFE این است که یک مدل با برخی پارامترها (همچنین به آن وزن‌ها یا ضرایب نیز گفته می‌شود) مانند رگرسیون خطی یا ماشین‌های بردار پشتیبان به صورت تکراری آموزش داده شود. اولین بار که ما مدل را آموزش می‌دهیم، تمام ویژگی‌ها را در نظر می‌گیریم. سپس، ویژگی با کوچک‌ترین ضریب را پیدا می‌کنیم (توجه کنید که این فرض را در نظر می‌گیریم که ویژگی‌ها تغییر مقیاس داده شده و یا استاندارد شده باشند)، به این معنی که اهمیت کمتری دارد، و این ویژگی را از مجموعه ویژگی‌ها حذف می‌کنیم.

سوال واضح این است: چند ویژگی را باید حفظ کنیم؟ می‌توانیم (به صورت فرضی) این حلقه را تا زمانی که فقط یک ویژگی باقی بماند، تکرار کنیم. یک رویکرد بهتر نیازمند این است که یک مفهوم جدید به نام اعتبارسنجی متقابل (CV) را در نظر بگیریم. ما در فصل بعد به تفصیل در مورد اعتبارسنجی متقابل (CV) صحبت خواهیم کرد، اما در اینجا ایده کلی را تشریح می‌کنیم.

داده‌هایی داریم که حاوی 1) مقدار هدفی است که می‌خواهیم پیش‌بینی کنیم و 2) ماتریس ویژگی است. ابتدا داده‌ها را به دو گروه مجموعه آموزشی و مجموعه آزمون تقسیم می‌کنیم. در مرحله دوم، مدل خود را با استفاده از مجموعه آموزشی آموزش می‌دهیم. در مرحله سوم، وانمود می‌کنیم که مقدار هدف مجموعه آزمون را نمی‌دانیم (به عبارتی مقداری که باید پیش‌بینی شود را کنار می‌گذاریم) و مدل خود را بر روی ویژگی‌های مجموعه آزمون اعمال می‌کنیم تا مقادیر مجموعه آزمون را پیش‌بینی کنیم. در نهایت، مقادیر هدف پیش‌بینی شده خود را با مقادیر هدف واقعی مقایسه کرده و مدل خود را ارزیابی می‌کنیم.

می‌توانیم از اعتبارسنجی متقابل (CV) برای یافتن تعداد بهینه ویژگی‌هایی که باید در طول RFE نگه داریم استفاده کنیم. به طور خاص، در RFE پس از هر تکرار، از اعتبارسنجی متقابل برای ارزیابی مدل خود استفاده می‌کنیم. اگر اعتبارسنجی متقابل نشان دهد که مدل ما پس از حذف یک ویژگی بهبود یافته است، تکرار کار خود را ادامه می‌دهیم. اما اگر اعتبارسنجی متقابل نشان دهد که مدل ما پس از حذف یک ویژگی بدتر شده است، آن ویژگی را دوباره به مجموعه ویژگی‌ها برمی‌گردانیم و آن را به عنوان بهترین ویژگی‌ها انتخاب می‌کنیم.

در کتابخانه scikit-learn، RFE همراه با اعتبار سنجی متقابل با استفاده از تابع RFECV پیاده‌سازی شده است و شامل تعدادی پارامتر مهم است. پارامتر estimator، نوع مدلی را که می‌خواهیم آموزش دهیم (به عنوان مثال رگرسیون خطی) تعیین می‌کند. پارامتر step تعداد یا نسبت ویژگی‌هایی را که در هر حلقه حذف می‌شوند، تنظیم می‌کند. پارامتر scoring معیار ارزیابی کیفیتی که برای ارزیابی مدل خود در طول اعتبارسنجی متقابل استفاده می‌شود را تعیین می‌کند.

**برای مشاهده موضوعات یا دستورالعمل‌های مشابه به منبع زیر مراجعه نمایید:**

* [Recursive feature elimination with cross-validation](http://bit.ly/2Ftuffz)

1. Categorical [↑](#footnote-ref-1)