فصل ۱۶- رگرسیون لجستیک

۱۶.۰ مقدمه

علیرغم اینکه رگرسیون لجستیک، رگرسیون نامیده می‌شود، در واقع یک تکنیک طبقه‌بندی نظارت شده پرکاربرد است. رگرسیون لجستیک و بسط‌های آن، مانند رگرسیون لجستیک چند جمله ای، به ما این امکان را می‌دهند که با استفاده از یک رویکرد ساده و کاملاً قابل درک، احتمال اینکه یک مشاهده از یک کلاس خاص است را پیش‌بینی کنیم. در این فصل، آموزش انواع طبقه‌بندی کننده‌ها را با استفاده از scikit-learn پوشش خواهیم داد.

۱۶.۱ تطبیق یک خط

مسئله

شما باید یک مدل طبقه‌بندی کننده ساده را آموزش دهید.

راه‌حل

آموزش یک رگرسیون لجستیک در scikit-learn با استفاده از LogisticRegression:

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

# Load data with only two classes
iris = datasets.load_iris()
features = iris.data[:100,:]
target = iris.target[:100]

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

# Create logistic regression object
logistic_regression = LogisticRegression(random_state=0)

# Train model
model = logistic_regression.fit(features_standardized,
target)
```

بحث

علیرغم وجود "رگرسیون" در نام خود، رگرسیون لجستیک در واقع یک طبقه‌بندی کننده‌ی دودویی پرکاربرد است (به عنوان مثال، بردار هدف فقط می‌تواند دو مقدار بگیرد). در یک رگرسیون لجستیک، یک مدل خطی (به عنوان مثال، $(\beta_0 + \beta_1 x$ در یک تابع لجستیک (که سیگموئید نیز نامیده می‌شود)، $\frac{1}{1+e^{-z}}$ گنجانده می‌شود، به طوری که:

$$P(y_i = 1|X) = \frac{1}{1 + e^{-(\beta_0+\beta_1 x)}}$$

که در آن $P(y_i = 1|X)$ احتمال کلاس ۱ بودن مقدار هدف (y_i) در مشاهده i‌ام است، X بردار داده آموزشی است، β_0 و β_1 پارامترهایی هستند که باید گرفته شوند و e عدد اویلر است. اثر تابع لجستیک این است که مقدار خروجی تابع را بین ۰ و ۱ محدود می‌کند تا بتوان آن را به عنوان یک احتمال تفسیر کرد. اگر $P(y_i = 1|X)$ بزرگتر از 0.5 باشد، کلاس 1 پیش بینی می‌شود. در غیر این صورت کلاس ۰ پیش بینی می‌شود.

در scikit-learn، ما می‌توانیم یک مدل رگرسیون لجستیک را با استفاده از LogisticRegression یاد بگیریم. پس از آموزش، می‌توانیم از مدل برای پیش بینی کلاس مشاهدات جدید استفاده کنیم:

```
# Create new observation
new_observation = [[.5, .5, .5, .5]]

# Predict class
model.predict(new_observation)
```

```
array([1])
```

در این مثال، مشاهده ما کلاس ۱ پیش‌بینی شده است. علاوه بر این، می‌توانیم این احتمال را ببینیم که یک مشاهده عضوی از هر کلاس است:

```
# View predicted probabilities
model.predict_proba(new_observation)
```

```
array([[ 0.18823041, 0.81176959]])
```

مشاهدات ما ۱۸.۸ درصد شانس کلاس ۰ بودن و ۸۱.۱ درصد احتمال کلاس ۱ بودن داشت.

۱۶.۲ آموزش یک طبقه‌بندی کننده چند کلاسه

مسئله

با توجه به بیش از دو کلاس، شما باید یک مدل طبقه‌بندی کننده را آموزش دهید.

راه‌حل

یک رگرسیون لجستیک را در scikit_learn با LogisticRegression با استفاده از روش یکی در مقابل بقیه[1] یا روش‌های چند جمله‌ای آموزش دهید:

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


# Load data
iris = datasets.load_iris()
features = iris.data
target = iris.target


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


# Create one-vs-rest logistic regression object
logistic_regression = LogisticRegression(random_state=0,
multi_class="ovr")


# Train model
model = logistic_regression.fit(features_standardized,
target)
```

بحث

رگرسیون‌های لجستیک به خودی خود فقط طبقه‌بندی کننده‌های باینری هستند، به این معنی که نمی‌توانند بردارهای هدف با بیش از دو کلاس را مدیریت کنند. با این حال، دو افزونه هوشمندانه برای رگرسیون لجستیک این کار را انجام می‌دهند. در روش اول، در رگرسیون لجستیکِ یکی در مقابل بقیه (OVR) یک مدل جداگانه برای هر کلاس آموزش داده می‌شود که پیش‌بینی می‌کند آیا یک مشاهده، از آن کلاس است یا خیر (در نتیجه آن را به یک مسئله طبقه‌بندی باینری تبدیل می‌کند). فرض می‌کند که هر مسئله طبقه‌بندی (به عنوان مثال، کلاس ۰ یا نه) مستقل است.

روش دیگر، در رگرسیون لجستیک چند جمله‌ای (MLR) تابع لجستیکی که در دستور العمل ۱۵.۱ دیدیم با یک تابع softmax جایگزین می‌شود:

$$P(y_i = 1|X) = \frac{e^{\beta_k x_i}}{\sum_{j=1}^{K} e^{\beta_j x_i}}$$

که در آن $P(y_i = 1|X)$ احتمال کلاس k بودن مقدار هدف (y_i) در مشاهده iام است و K تعداد کل کلاس‌ها است. یکی از مزیت‌های عملی MLR این است که احتمالات پیش بینی شده آن با استفاده از روش predict_proba قابل اطمینان تر هستند (یعنی کالیبره بودنِ بهتر).

[1] . one-vs-rest

هنگام استفاده از LogisticRegression، می‌توانیم با تنظیم مقدار multi_class انتخاب کنیم که کدام یک از دو تکنیک را می‌خواهیم که البته پارامتر ovr به صورت پیش فرض استفاده می‌شود. ما می‌توانیم با تنظیم آرگومان بر روی چند جمله‌ای به یک MNL سوئیچ کنیم.

۱۶.۳ کاهش واریانس از طریق منظم سازی

مسئله

شما باید واریانس مدل رگرسیون لجستیک خود را کاهش دهید.

راه‌حل

تنظیم فراپارامتر قدرت تنظیم، C:

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

# Load data
iris = datasets.load_iris()
features = iris.data
target = iris.target

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

# Create decision tree classifier object
logistic_regression = LogisticRegressionCV(
penalty='l2', Cs=10, random_state=0, n_jobs=-1)

# Train model
model = logistic_regression.fit(features_standardized, target)
```

بحث

منظم سازی، روشی برای جریمه کردن مدل‌های پیچیده برای کاهش واریانس آن‌ها است. به طور خاص، یک عبارتِ جریمه به تابع ضرر اضافه می‌شود که ما در تلاش برای به حداقل رساندن آن هستیم، معمولاً جریمه‌های L1 و L2. در پنالتی L1:

$$a\sum_{j=1}^{p}\hat{\beta}_j$$

که در آن $\hat{\beta}_j$ پارامترهای j ام از ویژگی‌های p است آموخته می‌شوند و α یک فراپارامتر است که قدرت منظم سازی را نشان می‌دهد. با پنالتی L2:

$$a\sum_{j=1}^{p}\hat{\beta}_j^2$$

مقادیر بالاتر α باعث افزایش جریمه برای مقادیر پارامترهای بزرگتر (به عنوان مثال، مدل‌های پیچیده‌تر) می‌شود. scikit-learn از روش رایج استفاده از C به جای α پیروی می‌کند که در آن C معکوس قدرت منظم سازی است: $C = \frac{1}{\alpha}$. برای کاهش واریانس در حین استفاده از رگرسیون لجستیک، می‌توانیم C را به عنوان یک فراپارامتر در نظر بگیریم تا مقدار C را که بهترین مدل را ایجاد می‌کند، تنظیم کنیم. در scikit-learn می‌توانیم از کلاس LogisticRegressionCV برای تنظیم مؤثر C استفاده کنیم. پارامتر Cs، LogisticRegressionCV، می‌تواند محدوده‌ای از مقادیر را برای جستجوی C بپذیرد (اگر لیستی از اعداد اعشاری به عنوان یک آرگومان ارائه شود) یا اگر یک عدد صحیح ارائه شود، فهرستی از تعداد زیادی از مقادیر کاندید را که از مقیاس لگاریتمی‌بین ۱۰،۰۰۰- تا ۱۰،۰۰۰ ترسیم شده اند ایجاد می‌کند.

متأسفانه، LogisticRegressionCV به ما اجازه نمی‌دهد تا در مورد عبارات مختلف جریمه‌ها جستجو کنیم. برای انجام این کار، باید از تکنیک‌های انتخاب مدل کمتر کارآمدی که در فصل ۱۲ بحث شده است استفاده کنیم.

۱۶.۴ آموزش یک طبقه‌بندی کننده بر روی داده‌های بسیار بزرگ

مسئله

شما باید یک مدل طبقه‌بندی کننده ساده را روی مجموعه بسیار بزرگی از داده‌ها آموزش دهید.

راه‌حل

با استفاده از حل کننده‌ی گرادیان میانگین تصادفی (SAG) یک رگرسیون لجستیک را در scikit-learn با LogisticRegression آموزش دهید:

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


# Load data
iris = datasets.load_iris()
features = iris.data
target = iris.target


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


# Create logistic regression object
logistic_regression = LogisticRegression(random_state=0, solver="sag")


# Train model
model = logistic_regression.fit(features_standardized, target)
```

بحث

رگرسیون لجستیک scikit-learn تعدادی تکنیک را برای آموزش یک رگرسیون لجستیک ارائه می‌دهد که حل کننده[۲] نامیده می‌شوند. اکثر اوقات scikit-learn بهترین حل کننده را به طور خودکار برای ما انتخاب می‌کند یا به ما هشدار می‌دهد که نمی‌توانیم کاری با آن حل کننده انجام دهیم. با این حال، یک مورد خاص وجود دارد که باید از آن آگاه باشیم.

در حالی که توضیح دقیق، فراتر از محدوده این کتاب است (برای اطلاعات بیشتر به اسلایدهای مارک اشمیت در «همچنین ببینید» مراجعه کنید)، نزول گرادیان میانگین تصادفی به ما امکان می‌دهد که زمانی که داده‌های ما بسیار بزرگ هستند، یک مدل را بسیار سریع‌تر از حل‌کننده‌های دیگر آموزش دهیم. با این حال، به مقیاس بندی ویژگی‌ها نیز بسیار حساس است. بنابراین استاندارد کردن ویژگی‌های ما اهمیت ویژه ای دارد. ما می‌توانیم الگوریتم یادگیری خود را برای استفاده از این حل کننده با تنظیم solver='sag' تنظیم کنیم.

همچنین ببینید:

- به حداقل رساندن مجموع محدود با الگوریتم گرادیان میانگین تصادفی، مارک اشمیت

۱۶.۵ رسیدگی به کلاس‌های نامتعادل

مسئله

شما باید یک مدل طبقه‌بندی کننده ساده را آموزش دهید.

راه‌حل

[۲] - solver

آموزش یک رگرسیون لجستیک در scikit-learn با استفاده از LogisticRegression:

```
# Load libraries
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler


# Load data
iris = datasets.load_iris()
features = iris.data
target = iris.target


# Make class highly imbalanced by removing first 40
observations
features = features[40:,:]
target = target[40:]


# Create target vector indicating if class 0, otherwise 1
target = np.where((target == 0), 0, 1)


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


# Create decision tree classifier object
logistic_regression = LogisticRegression(random_state=0,
class_weight="balanced")


# Train model
model = logistic_regression.fit(features_standardized, target)
```

بحث

مانند بسیاری از الگوریتم‌های یادگیری دیگر در Sicit-Learn، LogisticRegression با یک روش داخلی برای مدیریت کلاس‌های نامتعادل همراه است. اگر کلاس‌های بسیار نامتعادل داریم و در طول پیش‌پردازش به آن توجه نکرده‌ایم، می‌توانیم از پارامتر class_weight برای وزن کردن کلاس‌ها استفاده کنیم تا مطمئن شویم که ترکیب متعادلی از هر کلاس داریم. به طور خاص، آرگومان متوازن، به طور خودکار، کلاس‌ها را به طور معکوس و متناسب با فرکانس آنها وزن می‌کند:

$$w_j = \frac{n}{kn_j}$$

که در آن w_j وزن کلاس j، n تعداد مشاهدات، n_j تعداد مشاهدات در کلاس j و k تعداد کل کلاس‌ها است.