فصل ۱۸- بیز ساده[1]

۱۸.۰ مقدمه

قضیه بیز، روشی برتر برای درک احتمال یک رویداد P(A|B)، با توجه به برخی اطلاعات جدید P(B|A)، و یک باور قبلی در مورد احتمال رویداد، P(A) است:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

محبوبیت روش بیزی در دهه گذشته سر به فلک کشیده است، به طوری که هر روز بیشتر با کاربردهای متداول سنتی در دانشگاه، دولت و تجارت رقابت می‌کند. در یادگیری ماشین، یکی از کاربردهای قضیه بیز برای طبقه‌بندی به شکل طبقه‌بندی کننده ساده بیز ارائه می‌شود. طبقه‌بندی‌کننده‌های ساده بیز، تعدادی از ویژگی‌های مطلوب را در یادگیری ماشینی عملی در یک طبقه‌بندی واحد ترکیب می‌کنند. این طبقه‌بندی کننده‌ها شامل موارد زیر هستند:

۱. یک رویکرد شهودی
۲. توانایی کار با داده‌های کوچک
۳. هزینه‌های محاسباتی کم برای آموزش و پیش‌بینی
۴. نتایج محکم در تنظیمات مختلف در اغلب موارد

به طور خاص، یک طبقه‌بندی ساده بیز بر اساس تابع زیر است:

$$P(y|x_1, \dots, x_j) = \frac{P(x_1, \dots, x_j|y)P(y)}{P(x_1, \dots, x_j)}$$

که:

- $P(y|x_1, \dots, x_j)$ پسین[2] نامیده می‌شود و برابر با احتمال مشاهده‌ی کلاس y با توجه به مقادیر مشاهده برای ویژگی‌های j است. $x_j, \dots, x_1$.
- $P(y|x_1, \dots, x_j)$ احتمال[3] نامیده می‌شود و برابر با احتمال مقادیر یک مشاهده برای ویژگی‌های $x_j, \dots, x_1$، با توجه به کلاس آنها، یعنی y است.
- $P(y)$ پیشین[4] نامیده می‌شود و اعتقاد ما به احتمال کلاس y قبل از مشاهده داده‌ها است.
- $P(x_1, \dots, x_j)$ احتمال حاشیه‌ای[5] نامیده می‌شود.

---

[1] - Naive Bayes
[2] - posterior
[3] - likelihood
[4] - prior
[5] - marginal probability

در بیز ساده، ما مقادیر پسین یک مشاهده را برای هر کلاس ممکن مقایسه می‌کنیم. به طور خاص، از آنجایی که احتمال حاشیه ای در این مقایسه‌ها ثابت است، ما اعداد پسین را برای هر کلاس، مقایسه می‌کنیم. برای هر مشاهده، کلاسی که بیشترین عدد پسین را دارد، کلاس پیش‌بینی شده‌ی $\hat{y}$ می‌شود.

دو نکته مهم در مورد طبقه‌بندی کننده‌های ساده‌ی بیز وجود دارد. ابتدا، برای هر ویژگی در داده‌ها، باید توزیع آماری احتمال $P(x_i|y)$ را فرض کنیم. توزیع‌های رایج عبارتند از توزیع‌های نرمال (گاوسی)، چند جمله‌ای و برنولی. توزیع انتخاب شده اغلب بر اساس ماهیت ویژگی‌ها (پیوسته، باینری و غیره) تعیین می‌شود. دوم اینکه، بیز ساده نام خود را به این دلیل گرفته است که ما فرض می‌کنیم هر ویژگی و احتمال نتیجه‌ی آن مستقل است. این فرض "ساده لوحانه[۶]" اغلب اشتباه است، اما در عمل برای جلوگیری از ساخت طبقه‌بندی‌کننده‌های باکیفیت، کار چندانی انجام نمی‌دهد.

در این فصل ما استفاده از scikit-learn را برای آموزش سه نوع طبقه‌بندی کننده ساده بیز با استفاده از سه توزیع احتمال متفاوت پوشش خواهیم داد.

۱۸.۱ آموزش یک طبقه‌بندی کننده برای ویژگی‌های پیوسته

مسئله

شما فقط ویژگی‌های پیوسته دارید و می‌خواهید یک طبقه‌بندی کننده‌ی ساده‌ی بیز را آموزش دهید.

راه‌حل

از یک طبقه‌بندی کننده بیز ساده گوسی در scikit-learn استفاده کنید:

```
# Load libraries
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB


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


# Create Gaussian Naive Bayes object
classifer = GaussianNB()


# Train model
model = classifer.fit(features, target)
```

بحث

[6] - naïve

رایج ترین نوع طبقه‌بندی کننده ساده‌ی بیز، بیز ساده گوسی است. در بیز ساده‌ی گاوسی، فرض می‌کنیم که احتمال مقادیر ویژگی x، با توجه به مشاهده از کلاس y است که از توزیع نرمال پیروی می‌کند:

$$P(x_j|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} e^{-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}}$$

که σ_y^2 و μ_y واریانس و مقادیر میانگین ویژگی x_j برای کلاس y هستند. به دلیل فرض توزیع نرمال، بیز ساده گوسی در مواردی که همه ویژگی‌های ما پیوسته هستند بهترین استفاده را دارد.

در scikit-learn، ما یک بیز ساده‌ی گاوسی را مانند هر مدل دیگری با استفاده از تناسب، آموزش می‌دهیم، و سپس می‌توانیم در مورد کلاس مشاهده، پیش‌بینی کنیم:

```
# Create new observation
new_observation = [[ 4,  4,  4,  0.4]]


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

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

یکی از جنبه‌های جالب طبقه‌بندی‌کننده‌های ساده‌ی بیز این است که به ما اجازه می‌دهند یک باور قبلی[7] را به کلاس‌های هدف اختصاص دهیم. ما می‌توانیم این کار را با استفاده از پارامتر priors در کلاس GaussianNB انجام دهیم، که فهرستی از احتمالات اختصاص داده شده به هر کلاس از بردار هدف را می‌گیرد:

```
# Create Gaussian Naive Bayes object with prior
probabilities of each class
clf = GaussianNB(priors=[0.25, 0.25, 0.5])


# Train model
model = classifer.fit(features, target)
```

اگر هیچ آرگومانی به پارامتر priors اضافه نکنیم، prior بر اساس داده‌ها تنظیم می‌شود.

در نهایت، توجه داشته باشید که احتمالات خام پیش‌بینی شده از بیز ساده‌ی گاوسی (که خروجی دریافت شده با استفاده از predict_proba هستند) کالیبره نشده اند. یعنی نباید آنها را باور کرد. اگر بخواهیم احتمالات پیش‌بینی شده‌ی مفیدی ایجاد کنیم، باید آنها را با استفاده از رگرسیون ایزوتونیک یا یک روش مرتبط کالیبره کنیم.

همچنین ببینید:

- چگونه طبقه‌بندی کننده ساده بیز در یادگیری ماشینی، Dataaspirant کار می‌کند.

[7] - prior belief

۱۸.۲ آموزش یک طبقه‌بندی کننده برای ویژگی‌های گسسته و شمارشی

مسئله

با توجه به داده‌های گسسته یا شمارشی، باید یک طبقه‌بندی کننده ساده بیز را آموزش دهید.

راه‌حل

از یک طبقه‌بندی کننده ساده چند جمله ای Bayes استفاده کنید:

```
# Load libraries
import numpy as np
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer


# Create text
text_data = np.array(['I love Brazil. Brazil!', 'Brazil is best',
'Germany beats both'])


# Create bag of words
count = CountVectorizer()
bag_of_words = count.fit_transform(text_data)


# Create feature matrix
features = bag_of_words.toarray()


# Create target vector
target = np.array([0,0,1])


# Create multinomial naive Bayes object with prior
probabilities of each class
classifer = MultinomialNB(class_prior=[0.25, 0.5])


# Train model
model = classifer.fit(features, target)
```

بحث

بیز ساده چند جمله‌ای مشابه بیز ساده گوسی کار می‌کند، اما فرض بر این است که ویژگی‌ها به صورت چند جمله‌ای توزیع شده اند. در عمل، این بدان معنی است که این طبقه‌بندی معمولاً زمانی استفاده می‌شود که داده‌های مجزا داشته باشیم (به عنوان مثال، رتبه‌بندی فیلم‌ها از ۱ تا ۵). یکی از رایج‌ترین کاربردهای بیز ساده‌ی چند جمله‌ای، طبقه‌بندی متن با استفاده از کیسه‌های کلمات یا رویکردهای tf-idf است (به دستور العمل‌های ۶.۸ و ۶.۹ مراجعه کنید).

در راه حل خود، ما یک مجموعه داده متن اسباب‌بازی[8] از سه مشاهده ایجاد کردیم و رشته‌های متن را به یک ماتریس ویژگی کیسه‌ای از کلمات[9] و یک بردار هدف همراه تبدیل کردیم. سپس از MultinomialNB برای آموزش یک مدل در حالی که احتمالات قبلی را برای دو کلاس (طرفدار برزیل و طرفدار آلمان) تعریف می‌کنیم، استفاده کردیم.

MultinomialNB مشابه GaussianNB عمل می‌کند؛ مدل‌ها با استفاده از fit آموزش داده می‌شوند و مشاهدات را می‌توان با استفاده از predict، پیش‌بینی کرد:

```
# Create new observation
new_observation = [[0, 0, 0, 1, 0, 1, 0]]


# Predict new observation's class
model.predict(new_observation)
```

```
array([0])
```

اگر class_prior مشخص نشده باشد، احتمالات قبلی با استفاده از داده‌ها آموخته می‌شوند. با این حال، اگر بخواهیم یک توزیع یکنواخت به عنوان قبلی استفاده شود، می‌توانیم fit_prior=False را تنظیم کنیم.

در نهایت، MultinomialNB حاوی یک فراپارامتر هموارکننده‌ی افزودنی، به نام آلفا[10] است که باید تنظیم شود. مقدار پیش فرض ۱.۰ است، و همچنین ۰.۰ به این معنی که هیچ هموارسازی صورت نمی‌گیرد.

۱۸.۳ آموزش یک طبقه‌بندی کننده ساده بیز برای ویژگی‌های باینری

مسئله

شما داده‌های ویژگی‌های باینری دارید و باید یک طبقه‌بندی کننده ساده‌ی بیز را آموزش دهید.

راه‌حل

از یک طبقه‌بندی کننده ساده برنولی بیز استفاده کنید:

[8] - toy text
[9] - bag-of-words
[10] - alpha

```
# Load libraries
import numpy as np
from sklearn.naive_bayes import BernoulliNB


# Create three binary features
features = np.random.randint(2, size=(100, 3))


# Create a binary target vector
target = np.random.randint(2, size=(100, 1)).ravel()


# Create Bernoulli Naive Bayes object with prior
probabilities of each class
classifer = BernoulliNB(class_prior=[0.25, 0.5])


# Train model
model = classifer.fit(features, target)
```

بحث

طبقه‌بندی‌کننده بیز ساده برنولی فرض می‌کند که همه ویژگی‌های ما باینری هستند به طوری که فقط دو مقدار را می‌گیرند (به عنوان مثال، یک ویژگی طبقه‌بندی اسمی‌که یک‌بار کدگذاری شده است). مانند پسر عموی چندجمله‌ای[11] خود، بیز ساده‌ی برنولی اغلب در طبقه‌بندی متن استفاده می‌شود، زمانی که ماتریس ویژگی ما صرفاً وجود یا عدم وجود یک کلمه در یک سند است. علاوه بر این، BernoulliNB نیز مانند MultinomialNB، دارای یک هایپرپارامتر هموار کننده‌ی افزودنی، به نام آلفا است که ما می‌خواهیم با استفاده از تکنیک‌های انتخاب مدل، آن را تنظیم کنیم. در نهایت، اگر بخواهیم از priors استفاده کنیم، می‌توانیم از پارامتر class_prior با لیستی حاوی احتمالات قبلی برای هر کلاس استفاده کنیم. اگر بخواهیم یک احتمال قبلی یکنواخت را مشخص کنیم، می‌توانیم fit_prior=False را تنظیم کنیم:

```
model_uniform_prior = BernoulliNB(class_prior=None,
fit_prior=True)
```

۱۸.۴ کالیبره کردن احتمالات پیش‌بینی شده

مسئله

شما می‌خواهید احتمالات پیش‌بینی شده را از طبقه‌بندی کننده‌های ساده‌ی بیز کالیبره کنید تا قابل تفسیر باشند.

راه‌حل

از CalibratedClassifierCV استفاده کنید:

[11] - multinomial cousin

```
# Load libraries
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.calibration import CalibratedClassifierCV

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

# Create Gaussian Naive Bayes object
classifer = GaussianNB()

# Create calibrated cross-validation with sigmoid
calibration
classifer_sigmoid = CalibratedClassifierCV(classifer, cv=2,
method='sigmoid')

# Calibrate probabilities
classifer_sigmoid.fit(features, target)

# Create new observation
new_observation = [[ 2.6, 2.6, 2.6, 0.4]]

# View calibrated probabilities
classifer_sigmoid.predict_proba(new_observation)
```

```
array([[ 0.31859969, 0.63663466, 0.04476565]])
```

بحث

احتمالات کلاس، بخش مشترک و مفیدی از مدل‌های یادگیری ماشین هستند. در scikit-learn، اکثر الگوریتم‌های یادگیری به ما امکان می‌دهند تا احتمالات پیش‌بینی شده عضو یک کلاس را با استفاده از predict_proba ببینیم. این مورد می‌تواند بسیار مفید باشد اگر برای مثال، بخواهیم فقط یک کلاس خاص را پیش‌بینی کنیم در صورتی که مدل احتمال، این را پیش‌بینی کند که آن کلاس بیش از ۹۰٪ است. با این حال، برخی از مدل‌ها، از جمله طبقه‌بندی‌کننده‌های ساده‌ی بیز، احتمالاتی را به دست می‌دهند که بر اساس دنیای واقعی نیستند. یعنی، predict_proba ممکن است پیش‌بینی کند که یک مشاهده ۰.۷۰ شانس دارد که یک کلاس خاص باشد، در حالی که واقعیت این است که احتمال واقعی ۰.۱۰ یا ۰.۹۹ است. به طور خاص در بیز ساده، در حالی که رتبه‌بندی احتمالات پیش‌بینی‌شده برای کلاس‌های هدف مختلف، معتبر است، احتمالات پیش‌بینی‌شده خام تمایل به گرفتن مقادیر شدید نزدیک به ۰ و ۱ دارند.

برای به دست آوردن احتمالات پیش‌بینی شده معنادار، ما نیاز به انجام آنچه کالیبراسیون نامیده می‌شود داریم. در scikit-learn می‌توانیم از کلاس CalibratedClassifierCV برای ایجاد احتمالات پیش‌بینی‌شده با کالیبراسیون خوب با استفاده از اعتبارسنجی متقاطع[۱۲] k-fold استفاده کنیم. در CalibratedClassifierCV از مجموعه‌های آموزشی برای آموزش مدل

[12] - cross-validation

و از مجموعه داده‌های تست برای کالیبره کردن احتمالات پیش‌بینی شده استفاده می‌شود. احتمالات پیش‌بینی‌شده‌ی به دست آمده از این کالیبراسیون، میانگین k-folds هستند.

با استفاده از این راه حل ما می‌توانیم تفاوت بین احتمالات پیش‌بینی شده خام و احتمالات به خوبی کالیبره شده را ببینیم. در راه حل خود، ما یک طبقه‌بندی کننده‌ی ساده‌ی گاوسی بیز ایجاد کردیم. اگر این طبقه‌بندی‌کننده را آموزش دهیم و سپس احتمالات کلاس را برای یک مشاهده جدید پیش‌بینی کنیم، می‌توانیم تخمین‌های احتمال بسیار شدید را ببینیم:

```
# Train a Gaussian naive Bayes then predict class
probabilities
classifer.fit(features,
target).predict_proba(new_observation)
```

```
array([[ 2.58229098e-04, 9.99741447e-01, 3.23523643e-
07]])
```

با این حال، پس از کالیبره کردن احتمالات پیش‌بینی شده (که در راه حل خود انجام دادیم)، نتایج بسیار متفاوتی دریافت می‌کنیم:

```
# View calibrated probabilities
classifer_sigmoid.predict_proba(new_observation)
```

```
array([[ 0.31859969, 0.63663466, 0.04476565]])
```

CalibratedClassifierCV دو روش کالیبراسیون ارائه می‌دهد - مدل سیگموئید پلات[۱۳] و رگرسیون ایزوتونیک[۱۴] - که توسط پارامتر method، تعریف شده اند. در حالی که فضایی برای پرداختن به جزئیات نداریم؛ زیرا رگرسیون ایزوتونیک ناپارامتریک است، با این حال زمانی که اندازه نمونه بسیار کوچک است (به عنوان مثال، ۱۰۰ مشاهده) رگرسیون ایزوتونیک تمایل دارد که بیش از حد برازش[۱۵] کند. در این راه حل، از مجموعه داده عنبیه با ۱۵۰ مشاهدات استفاده کردیم و بنابراین از مدل سیگموئید پلات استفاده کردیم.

[13] - Platt’s sigmoid model
[14] - Isotonic regression
[15] - overfit