دانشگاه تهران

دانشکده علوم مهندسی

تکتم سمیعی
۸۱۰۸۹۶۰۵۴

یادگیری ماشین – دکتر سایه میرزایی

تمرین دوم

بهار ۱۴۰۰

سوال اول

قسمت اول :

در مدل های مولد یا generative model ،برخلاف مدل های discriminative که براساس مدل های $p_\theta(y|x)$ هستند ، به دنبال مدل هایی هستیم که از فضای y به فضای x نگاشت پیدا میکند . در واقع به دنبال مدل هایی هستیم که $p_\theta(x|y)$ را پیدا کند . به عبارتی دیگر به دنبال یادگیری ویژگی های یک کلاس هستند . به طور خلاصه ، generative models ، چکونگی تولید یک ورودی جدید توسط یک کلاس خاص را مدل میکند . وقتی یک نمونه ی مشاهده شده ی جدید داده میشود ، سعی میکند پیش بینی کند کدام کلاس به احتمال زیاد مشاهدات داده شده را ایجاد کرده است .

به زبان ریاضی generative models ، سعی میکنند تابع توزیع احتمال مشترک $p(x,y)$ ورودی های , x و لیبل y را یاد بگیرند و با استفاده از قانون بیز ، احتمال شرطی $p(y|x)$ را محاسبه کنند , و سپس بتوانند مقادیر y را برای نمونه های بعدی پیش بینی کنند و سپس نتیجه با بیشترین احتمال را انتخاب کنند .

روابط ریاضیاتی :

در این مدل ما به دنبال یافتن $p(x|y)$ تا به کمک قانون بیز ، بتوانیم مقدار $p(y|x)$ را محاسبه کنیم :

قانون بیز :

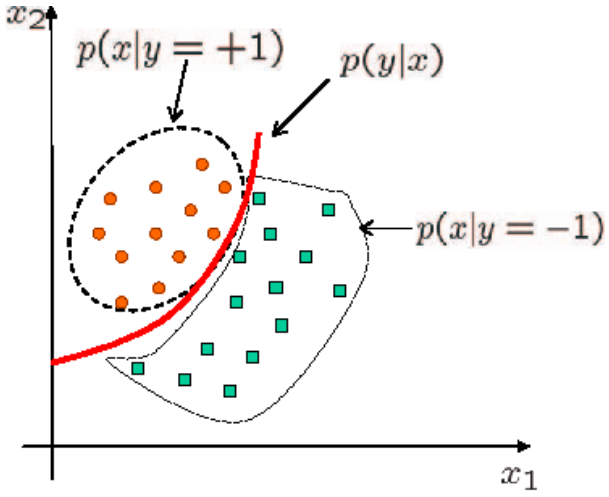
$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

به این صورت که برای هر ویژگی توزیع احتمالاتی آن را پیدا میکنیم و همچنین از داده های موجود ، $p(y)$ را محاسبه میکنیم . سپس برای ورودی جدید مقدار $p(x|y)$ را برای تمام کلاس ها محاسبه کرده و در $p(y)$ ضرب کرده و کلاسی که بیش ترین احتمال را دارد ، به عنوان خروجی y انتخاب میکنیم . از آنجایی که مقدار $p(x)$ برای تمامی کلاس ها یکسان است و ما به دنبال محاسبه ی ماکزیمم مقدار هستیم نه مقدار مطلق احتمال ، میتوانیم از محاسبه ی آن چشم پوشی کنیم .

قسمت دوم :

برای استفاده از generative model ، مقدار متغیر های بیشتری باید محاسبه شود ، به این صورت که برای هر کلاس باید توزیع احتمالاتی آن و همچنین احتمال مشاهده ی هر کلاس محاسبه شود . این توزیع های احتمالاتی برای برای محاسبه ی $p(x,y)$ استفاده میشود و سپس احتمال شرطی $p(x|y)$ محاسبه میشود . در حالیکه discriminative models مسیر کوتاه تری را طی میکند ، به این صورت که به طور مستقیم احتمال شرطی $p(x|y)$ را طی میکند .

generative model برای تولید نمونه های جدید استفاده میشود ولی به داده های بیشتری نیاز دارد . در حالیکه discriminative models اطلاعاتی درباره ی رابطه ی بین ویژگی ها ندارد و در نتیجه نمیتواند داده های جدیدی تولید کن



ustration of generative v.s. discriminative models. Discri

قسمت سوم :

راهکارهایی برای مقابله با دادگانی با مقادیر NaN وجود دارد که البته بستگی به مدلی که استفاده میکنیم دارد . به عنوان مثال میتوان داده های NaN را با میانگین ستون پر کنیم یا از مقادیر همسایه های آن سلول استفاده کنیم یا میتوان از روش رگرسیون خطی برای یافتن مقدار داده های گم شده استفاده کرد .

قسمت چهارم :

Gaussian Discriminant Analysis

زمانی که مسئله ی طبقه بندی با متغیرهایی با توزیع رندوم پیوسته هستند ، از این مدل استفاده میکنیم . GDA یک generative learning algorithm است و در آن فرض میکنیم $p(x\,|\,y)$ یک توزیع multivariate normal distribution و $p(y)$ یک توزیع برنولی است :

$$p(y) = \phi^{y}(1-\phi)^{(1-y)}$$
$$p(x\,|\,y=0) = \frac{1}{(2\pi)^{n/2}|\sum|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu_0)^T\sum^{-1}(x-\mu_0))$$
$$p(x\,|\,y=1) = \frac{1}{(2\pi)^{n/2}|\sum|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu_1)^T\sum^{-1}(x-\mu_1))$$

حالا همان گونه که در رگرسیون لاجیستیک عمل کردیم ، در اینجا هم باید تابع log likelihood تعریف کنیم و آن را نسبت به پارامتر های مسئله ماکزیمم کنیم :

$$\ell(\phi, \mu_0, \mu_1, \sum) = \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \sum)$$

$$= \log \prod_{i=1}^{m} p(x^{(i)} \mid y^{(i)}; \phi, \mu_0, \mu_1, \sum) p(y^{(i)}; \phi)$$

سپس با محاسبه ی مشتقات جزئی از پارامتر ها و مساوی صفر قراردادن آن ، در نهایت به پاسخ های زیر میرسیم :

$$\phi = \frac{1}{m} \sum_{i=1}^{m} 1\{y^{(i)} = 1\}$$

$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$

$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$

$$\sum = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu_k)(x^{(i)} - \mu_k)^T \quad \text{where } k = 1\{y^{(i)} = 1\}$$

Logistic regression

رگرسیون لاجیستیک ، یک discriminative model است و به صورت زیر تعریف میشود :

$$p(y = 1 \mid x) = \frac{1}{1 + exp(-\theta^T x)}$$

و θ تابعی از Σ μ_1 μ_0 ϕ است . اگر رابطه ی بالا را به فرم طبق رابطه ی بیز باز نویسی کنیم ، به رابطه ی زیر میرسیم :

$$\frac{1}{1+\frac{p(x|y=0)\times p(y=0)}{p(x|y=1)\times p(y=1)}} = \frac{1}{1+exp[(\log(\frac{1-\phi}{\phi})-\frac{\mu_0^2+\mu_1^2}{2\sum})\times x_0+\frac{\mu_0-\mu_1}{\sum}\times x]}$$

این رابطه نشان میدهد که نتیجه ی GDA یک رگرسیون لاجیستیک است در حالیکه که عکس آن برقرار نیست . یعنی رگرسیون لاجیستیک بودن $p(y|x)$ ، توزیع گوسی چند متغیره ی $p(x|y)$ را نتیجه نمیدهد .

در واقه میتوانیم این را بیان کنیم که اگر $p(x|y)$ متعلق به خانواده ی نمایی باشد ، $p(y|x)$ آن حتما رگرسیون لاجیستیک است .

اکنون میبینم که رگرسیون لاجیستیک بسیار جامع تر است و برای فرض های زیادی قابل است .

قسمت پنجم :

LDA QDA

دو طبقه بند هستند کلاسیک هستند و از این جهت کاربرد دارند که راه حل های بسته دارند و به راحتی قابل محاسبه هستند ، ذاتا چندطبقه هستند و به خوبی استفاده میشوند و هیچ hyperparameter ی برای تنظیم ندارند .

هردو طبقه بند بر اساس رابطه ی بیزین تعریف میشوند و تفاوت آن ها با لاجیستیک رگرسیون در رویکرد آن ها در طبقه بندی است .

در LAD فرض میشود توزیع های احتمالاتی گوسی هستند :

$p(Y|y)$: احتمال پیشین حاصل از مشاهدات کلاس $y = k$

$\Sigma(Pr(X = x|y = p) * Pr(y = p))$: مجموع احتمالات مشاهدات برای هر کلاس به صورت جدا

LDA زمانی استفاده میشود که برای جدا سازی کلاس ها نیاز به linear boundry است و QDA برای جداسازی کلاس ها به صورت non-linear boundary است . هردو این روش ها زمانی که کلاس های پاسخ کاملا جدا هستند و و توزیع $X=x$ برای همه ی کلاس ها نرمال است ، بهتر کار میکنند .

سوال دوم

در ابتدا داده ها را read میکنیم :

```
data = pd.read_csv("wdbc.data")
```

	842302	M	17.99	10.38	122.8	1001	0.1184	0.2776	0.3001	0.1471	...	25.38	17.33	184.6	2019	0.1622	0.6656	0.7119	0.2654	0.4601
0	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	24.990	23.41	158.80	1956.0	0.12380	0.18660	0.2416	0.1860	0.2750
1	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	23.570	25.53	152.50	1709.0	0.14440	0.42450	0.4504	0.2430	0.3613
2	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	14.910	26.50	98.87	567.7	0.20980	0.86630	0.6869	0.2575	0.6638
3	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	22.540	16.67	152.20	1575.0	0.13740	0.20500	0.4000	0.1625	0.2364
4	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	...	15.470	23.75	103.40	741.6	0.17910	0.52490	0.5355	0.1741	0.3985
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
563	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	25.450	26.40	166.10	2027.0	0.14100	0.21130	0.4107	0.2216	0.2060
564	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	23.690	38.25	155.00	1731.0	0.11660	0.19220	0.3215	0.1628	0.2572
565	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	18.980	34.12	126.70	1124.0	0.11390	0.30940	0.3403	0.1418	0.2218
566	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	25.740	39.42	184.60	1821.0	0.16500	0.86810	0.9387	0.2650	0.4087
567	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	9.456	30.37	59.16	268.6	0.08996	0.06444	0.0000	0.0000	0.2871

568 rows × 32 columns

سپس ویژگی ها (X) و ستون هدف (Y) را در دو دیتا ست جدا نگه میداریم . ستون اول که شناسه است را به ماتریس X اضافه نمیکنیم :

```
Data = data
y = Data.iloc[:,1]
x = Data.iloc[:,2:32]
```

سپس با sample rate = 0.2 داده های test را از دادهای train جدا میکنیم :

```
sample_rate = 0.2

Y_val = y[:math.floor(len(data)*sample_rate)]
Y_train = y[math.floor(len(data)*sample_rate):]

X_val = x[:math.floor(len(data)*sample_rate)]
X_train = x[math.floor(len(data)*sample_rate):]
```

با دستور groupby ، میانگین و واریانس داده های هر ستون ماتریس X را بر اساس کلاس مختلف آن ها جدا میکنیم . در اینجا چون y دو مقدار M و B دارد ، دو کلاس داریم . در نتیجه ماتریس mean و var ، دارای دو سطر هستند که سطر اول مربوط به کلاس B ، و سطر دوم مربوط به کلاس M است . هر سطر شامل میانگین هر ستون ویژگی داده های X است .

همچنین میانگین داده های Y را با توجه به کلاس هر کدام به همین ترتیب محاسبه میکنیم . یعنی تعداد نمونه های B تقسیم بر کل نمونه ها و تعداد نمونه های M تقسیم بر کل نمونه ها :

```
mean = X_train.groupby(Y_train).apply(np.mean).to_numpy()
var = X_train.groupby(Y_train).apply(np.var).to_numpy()
P_y = (X_train.groupby(Y_train).apply(lambda x: len(x))/len(Y_train)).to_numpy()
```

حالا باید تابع Gaussian_Distribution را طراحی کنیم تا برای هر داده تست بتوانیم احتمال آن را محاسبه کنیم :

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

```
def Gaussian_Distribution(x , mean , var):
    return (np.exp(-((x - mean) ** 2) / (2 * var)))/(np.sqrt(2 * np.pi * var))
```

حال به توصیف الگوریتم میپردازیم :

Naive Bayse

یک نوع طبقه بندی بر مبنای قانون بیز است ، با فرض استقلال ویژگی ها از یکدیگر . به این معنی که فرض میکند حضور یک ویژگی در کلاس ، کاملا مستقل از حضور هر ویژگی دیگر در این کلاس است . در این الگوریتم به این صورت عمل میکنیم که برای هر کلاس ، میانیگین و واریانس هر ستون (هر ویژگی) را محاسبه میکنیم و همچنین برای هر کلاس میانگین ستون هدف را محاسبه میکنیم . به این ترتیب مدل بیزین را به دست می آوریم . سپس برای داده ی تست ، برای هر کلاس به صورت جداگانه ، مجموع لگاریتم احتمال هر ویژگی را با لگاریتم احتمال هر کلاس با هم جمع میکنیم ، سپس ماکزیمم عبارت های به دست آمده برای هر کلاس را به عنوان خروجی پیدا کرده ، و کلاس مروبط به آن را انتخاب میکنیم .

$$P(y|x_1,...,x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

$$P(y|x_1,...,x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

در این تابع به محاسبه احتمال های هر کلاس برای داده های تست میپردازیم و در نهایت یک لیست برمیگردانیم که نشان دهنده ی حدس های ما برای هر داده ی تست است .

```
def assess_func(X_val,mean,var,P_y):
    assessment =[]
    for r in range(len(X_val)):

        B_value = 0
        for c in range(len(X_val.iloc[0,:])):
            B_value += np.log(Gaussian_Distribution(X_val.iloc[r,c],mean[0,c],var[0,c]))
        B_value += np.log(P_y[0])

        M_value = 0
        for c in range(len(X_val.iloc[0,:])):
            M_value += np.log(Gaussian_Distribution(X_val.iloc[r,c],mean[1,c],var[1,c]))
        M_value += np.log(P_y[1])

        if B_value > M_value :
            assessment.append('B')
        elif B_value < M_value :
            assessment.append('M')
        del M_value
        del B_value
    return  assessment
```

در نهایت نتایج داده های تست را با لیست هدف Y مقایسه میکنیم و و نتایج درست را بر تعدا کل نتایج تقسیم میکنیم :

```
assessment = assess_func(X_val,mean,var,P_y)
counter =0
for i in range(len(Y_val)):
    if assessment[i]==Y_val[i]:
        counter+=1
print(counter/len(Y_val))
```

```
0.8761061946902655
```

نتیجه برابر است با : 0.87610

این الگوریتم به میزان 87.61 درصد جواب درست را برمیگرداند .

سوال سوم

قسمت اول :

هردو روش MLE. و MAP برای تخمین پارامترهای توزیع های احتمالاتی استفاده میشوند و از قانون بیز که چندی پیش به آن اشاره شد استفاده میکنیم .

Maximum Likelihood Estimation

در این روش ما به دنبال max کردن تابع log likelihood هستیم :

$$\begin{aligned}\theta_{ML} = \underset{\theta}{\operatorname{argmax}}\, L(\theta; X) &= \underset{\theta}{\operatorname{argmax}}\, p_{model}(X;\theta) \\ &= \underset{\theta}{\operatorname{argmax}}\, \Pi_{i=1}^{m} p_{model}(x^{(i)};\theta),\ \textit{where } m \textit{ denotes the dataset size} \\ &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \log p_{model}(x^{(i)};\theta)\end{aligned}$$

Maximum A Posteriori

در این روش به دنبال ماکزیمم کردن توزیع پیشین تتا $p(\theta)$ هستیم .

$$\begin{aligned}\theta_{MAP} &= \underset{\theta}{\operatorname{argmax}}\, p(\theta|X) \\ &= \underset{\theta}{\operatorname{argmax}}\, p(X|\theta) \cdot p(\theta),\ (\textit{by applying bayes rule}) \\ &= \underset{\theta}{\operatorname{argmax}}\, \log p(X|\theta) + \log p(\theta),\ (\textit{by applying logarithm}) \\ &= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{m} \{\log p(x^{(i)}|\theta)\} + \log p(\theta),\ (i.i.d)\end{aligned}$$

یعنی همانند MLE عمل میکنیم و تنها یک ترم $log^{p(\theta)}$ به آن اضافه میشود که بستگی به توزیع تتا دارد . به عنوان مثال اگر تتا توزیع یکنواخت داشته باشد در طی محاسبه ماکزیمم حذف میشود و MAP و MLE یکسان خواهد بود .

با مشاهده ی روابط بالا میبینیم که MLE. یک مورد از MAP. است . هردو روش یک مقدار ثابت ، به عنوان نقطه ی تخمین برمیگردانند .

ما زمانی از MLE استفاده میکنیم که اطلاعات قبلی راجب توزیع احتمالاتی نداشته باشیم ، در نتیجه نیازمند حجم زیادی دیتا هستیم تا بتوانیم پارامترهارا با دقت خوبی محاسبه کنیم . ولی در MAP نا اطلاعاتی راجب توزیع احتمالاتی داریم و باعث میشود مدل ما بر روی داده ها overfitt نشود . برای دیتا ست های کوچک ، بهتر است که از MAP استفاده کنیم تا بتوانیم تخمین بهتری بزنیم . در نهایت ،به این نکته اشاره میکنیم که افزودن توزیع احتمال پیشین ، وابستگی بیش از حد به داده های مشاهده شده برای تخمین پارامتر را کاهش میدهد .

قسمت دوم :

بخش ۱ :

ابتدا داده هارا لود میکنیم و سپس میانگین داده های هدف را پیدا کرده و داده ها با میانگین بزرگ تر را مساوی ۱ و داده های کوچک تر از میانگین را برابر 0 قرار میدهیم :

```
mean1 = data_["medv"].mean();
print(mean1)
```

```
22.532806324110677
```

```
data_.loc[(data_.medv >= mean1),'medv']= 1
data_.loc[(data_.medv != 1),'medv']= 0
```

سپس داده های ستون هدف را در ماتریس Y و ویژگی های ستون اول تا ۱۳ را در ماتریس X میریزیم . در این قسمت هم از ستون 0 م صرف نظر میکنیم چون ایدی هر ستون است .

```
Data = data_
y = Data.iloc[:,-1]
x = Data.iloc[:,1:14]
```

در اینجا هم مانند سوال قبل داده هارا با یک سمپل ریت مشخص به داده های test و train تقسیم میکنیم:

```
sample_rate = 0.2

Y_val_ = y[:math.floor(len(data_)*sample_rate)]
Y_train_ = y[math.floor(len(data_)*sample_rate):]

X_val_ = x[:math.floor(len(data_)*sample_rate)]
X_train_ = x[math.floor(len(data_)*sample_rate):]
```

و همچنین میانگین و واریانس را به برای هر ویژگی و کلاس به طور جداگانه و همچینین احتمال $p(Y = y)$ محاسبه میکنیم . و در نهایت با تابع assessment ، نتایج پیش بینی با مدل ایجاد شده را به دست میاوریم و با محاسبه ی میزان حدس های درست به مقدار55 درصد میرسیم :

```
mean_ = X_train_.groupby(Y_train_).apply(np.mean).to_numpy()
var_ = X_train_.groupby(Y_train_).apply(np.var).to_numpy()
P_y_ = (X_train_.groupby(Y_train_).apply(lambda x: len(x))/len(Y_train_)).to_numpy()
```

```
assessment_ = assess_func(X_val_,mean_,var_,P_y_)
for i in range (len(assessment_)):
    if assessment_[i] =='M':
        assessment_[i] = 1
    elif assessment_[i] =='B':
        assessment_[i] = 0
count =0
for i in range(len(Y_val_)):
    if assessment_[i]==Y_val_[i]:
        count+=1
print(count/len(Y_val_))
```

```
0.5544554455445545
```

سپس داده هارا با sample rate $= 0.6$ جدا میکنیم . عملیات را مجددا بر روی داده ها انجام میدهیم :

```
mean_ = X_train_.groupby(Y_train_).apply(np.mean).to_numpy()
var_ = X_train_.groupby(Y_train_).apply(np.var).to_numpy()
P_y_ = (X_train_.groupby(Y_train_).apply(lambda x: len(x))/len(Y_train_)).to_numpy()
```

```
assessment_ = assess_func(X_val_,mean_,var_,P_y_)
for i in range (len(assessment_)):
    if assessment_[i] =='M':
        assessment_[i] = 1
    elif assessment_[i] =='B':
        assessment_[i] = 0
count =0
for i in range(len(Y_val_)):
    if assessment_[i]==Y_val_[i]:
        count+=1
print(count/len(Y_val_))
```

```
0.6831683168316832
```

تعداد حدس های درست افزایش یافته است .

- https://medium.com/@akankshamalhotra٢۴/generative-classifiers-v-s-discriminative-classifiers-١٠۴۵f۴٩٩d٨cc#:~:text=Generative%٢٠Classifiers%٢٠tries%٢٠to%٢٠model,likely%٢٠generated%٢٠the%٢٠given%٢٠observation.
- https://developers.google.com/machine-learning/gan/generative
- https://www.quora.com/What-are-some-benefits-and-drawbacks-of-discriminative-and-generative-models
- https://towardsdatascience.com/generative-vs-٢۵٢٨de۴۳a٨۳۶
- https://towardsdatascience.com/gaussian-discriminant-analysis-an-example-of-generative-learning-algorithms-٢e٣٣۶ba٧aa۵c
- https://duphan.wordpress.com/٢٠١۶/١٠/٢٧/gaussian-discriminant-analysis-and-logistic-regression/
- https://datascienceplus.com/how-to-perform-logistic-regression-lda-qda-in-r/#:~:text=LDA%٢٠(Linear%٢٠Discriminant%٢٠Analysis)%٢٠is,for%٢٠all%٢٠class%٢٠is%٢٠normal.
- https://automaticaddison.com/difference-between-maximum-likelihood-and-maximum-a-posteriori-estimation/