یادگیری ماشین(تمرین ۲)

استاد:
دکتر میرزایی

سپیده فاطمی خوراسگانی
شماره دانشجویی: 810897059

سؤال اول:

۱) طبقه بندی با استفاده از مدل های مولد:

مسأله های طبقه بندی را به دو stage می‌توان تقسیم کرد:
stage استنباط یا inference که در این مرحله ما با استفاده از داده‌های آموزشی مسأله $P(y|x)$ را مدل می کنیم. و در مرحله بعد که stage تصمیم گیری یا decision است از آن احتمالات به دست آمده برای طبقه بندی بهینه استفاده کنیم.
در مدل مولد، در مرحله inference برای هر کلاس به طور جداگانه مقدار $P(x|y)$ یعنی توزیع مشاهدات ما در هر کلاس(class-conditional densities) و همچنین مقدار احتمال پیشین $p(y)$ را برای هر کلاس محاسبه می کنیم و سپس با استفاده از تئوری بیز برای محاسبه $P(y|x)$ داریم:

$$P(y|x)=\frac{P(x|y)*P(y)}{P(x)}$$

به عبارت دیگر با استفاده از likelihood احتمال prior یا پیشین را به احتمال posterior یا پسین تبدیل می کنیم.

$$posterior=\frac{likelihood*prior}{evidence}$$

به این طریق احتمال پسین کلاس‌ها را به دست می آوریم.
Likelihood به این معنی است که اگر ما مدل را y در نظر بگیریم احتمال مشاهده ما به چه صورت است.

برای محاسبه $p(x)$ در فضای تک بعدی داریم:

$$P(x)=\sum P(x|y)P(y)$$

بنابراین در تصمیم گیری ها کلاسی را می‌توانیم انتخاب کنیم که احتمال پسین یاposterior یا $P(y|x)$ بیشتری داشته باشد. یعنی برای دو کلاس داریم:

$$P(y_1|x)>P(y_2|x)$$

و در حالت کلی ماکسیمم $P(y|x)$ از بین همه کلاس‌ها، انتخاب ما می باشد :

$$max(P(y|x_{test}))$$

به دلیل اینکه مخرج $p(x)$ در همه کلاس ها یکسان و مثبت است در ماکسیمم گیری تأثیری ندارند:

$$max(P(y|x_{test}))=max\,P(x|y)P(y)$$

باید توجه شود که در فضایposterior مجموع احتمالات برابر با یک می باشد:

$$\sum_i P(y_i|x)=1$$

برای دو کلاس این رابطه به صورت زیر در می آید:

$$y_1 \rightarrow if \quad \frac{P(x|y_1)}{P(x|y_2)}>\frac{P(y2)}{P(y_1)}$$
$$y_2 \rightarrow otherwise$$

به کسر $\frac{P(x|y_1)}{P(x|y_2)}$ نسبت likelihood یا (likelihood ratio) می گوییم.
به عبارتی اگر نسبت likelihood کلاس ۱ به کلاس ۲ از نسبت prior کلاس ۲ به کلاس ۱ بیشتر باشد ما کلاس ۱ را انتخاب می کنیم.
به نحوی دیگر اگر برای $P(y|x)$ روابط زیر را در نظر بگیریم:

$$P(y_1|x)=\frac{P(x|y_1)*P(y_1)}{P(x)}=\frac{1}{1+e^{-a}}=\sigma(a)$$

تابع سیگموییدهمانند آنچه قبلاً داشتیم، کل مقادیر را به مقادیر محدود بین صفر و یک نگاشت می کند.

که در آنa برابر است با:

$$a=\ln\left(\frac{P(x|y_1)P(y_1)}{P(x|y_2)P(y_2)}\right)=\ln\left(\frac{\sigma}{1-\sigma}\right)$$

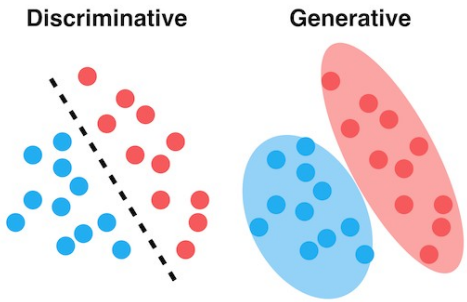
a به عنوان تابع logit هم شناخته می‌شود و log نسبت احتمالات را به ما می دهد.

این یادگیری برای کاربرد طبقه بندی، نظارتی است در generative model برای یادگیری و آموزش مدل به جای اینکه از مقادیر x به y برسیم از مقادیر y به x می‌رسیم و درواقع احتمال مشاهدات را با وجود دانستن کلاس آن بررسی می کنیم. ولی همچنان هدف، طبقه بندی داده‌های مشاهده شده می باشد.
کاربرد مدل های مولد در مسائل طبقه بندی به این صورت است که برای هر کلاس مدلی ایجاد می‌کند که با احتمالی می‌داند که داده‌های مشاهده شده آن کلاس چگونه بوجود آمده اند. و زمانی که داده جدید به آن داده می‌شود تلاش می‌کند که پیش‌بینی کند کدام کلاس احتمال بیشتری دارد که آن داده را ایجاد کرده باشد. این روش بیشتر سعی در شناخت محیط دارد.
در این مدل با استفاده از یک مدل احتمالاتی توضیح می‌دهد که داده‌های داده شده چگونه generateشده اند و همچنین این مدل توانایی generate کردن داده جدید بر اساس پارامتر های به دست آمده را دارد.
باید توجه شود که این مدل باید حتماً احتمالاتی یا probabilistic باشد و نه به صورت قطعی یا deterministic.
مثال‌هایی ازکاربرد generative model:

- Naivie Bayes
- Hidden Markov model
- LDA linear discriminant analysis

۲) مزایا و معایب طبقه بند مولد و کاربرد:

برای بررسی مزایا و معایب روش مولد آن را با روش discriminative مقایسه می کنیم:



همان‌طور که گفته شد در هر دو روش می‌خواهیم $P(y|x)$ را محاسبه کنیم. در روش discriminative مانند logistic regression مقدار آن را مستقیماً از داده‌ها به دست می آوریم. و هدف آن تشخیص مرز بین داده‌ها است.
مدل های مولد برای generate کردن داده جدید بسیار مناسب است اما برای ساختن مدلی که با استفاده از آن توزیع زیرین احتمال را به دست آورد، بسیار دشوار است. با این وجود در بعضی شرایط تخمین زدن پارامتر های مدل مولد ساده‌تر از یادگیری توسط مدل discriminative است.
روش مدل های مولد دارای فرضیات زیادی می‌باشد و دقت آن هم به اندازه روش discriminative نمی باشد.
از طرفی روش discriminative از لحاظ محاسباتی هزینه کمتری دارند. و برای داده‌های بزرگ روش بهتری می باشد.
البته با وجود اینکه یادگیری discriminative دقیق‌تر است و خطای زمان اجرای کمتری دارد اما روش generative سریع‌تر به خطا می رسد. البته گفته شده فقط در صورتی که داده‌های ورودی مناسب باشند به این شکل عمل خواهد کرد و در غیر این صورت هر دو الگوریتم عمل‌کرد یکسانی دارند.
بنابراین برای مسأله های طبقه بندی بیشتر از روش discriminative استفاده می شود.
روش مدل های مولد معمولاً برای یادگیری غیر نظارتی یا unsupervised learning استفاده می شود.

روش discriminative نسبت به داده‌های پرت مقاوم‌تر (more robust) می باشد. بنابر این برای تشخیص outlier ها مدل generative عمل‌کرد بهتری دارد.
روش مولد هنگامی که داده گم شده داشته باشیم به نسبت خوب عمل می کند.در ادامه بیشتر توضیح داده می شود.

۳) بدون داشتن همه ویژگی‌های داده‌های تست:

در روش مولد اگر داده گم شده یا NaN داشته باشیم روش‌های اصولی در مواجهه با آن وجود دارد.مثلاً اگر مدل های ما گوسی یا Naive Bayes باشد می‌توان از بین بقیه متغیر های باقی‌مانده بهترین گزینه را انتخاب کرد و متغیر های از دست رفته را با استفاده از آن مقدار دهی کرد.
اما برای مدل discriminative این کار به دقت بالایی نیاز دارد و می‌تواند در نتیجه تأثیر گذار باشد.
در مقاله GAIN: Missing Data Imputation using Generative Adversarial Nets توضیح داده شده که با استفاده از فریم ورک GAIN از داده‌های واقعی در دسترس استفاده می‌کند و مقادیر گم شده را با احتمال یا تخمینی generate می کند. البته در این متد مدل discriminator هم در نظر گرفته شده که وظیفه آن تشخیص داده‌های واقعی از آن داده‌های generate شده می باشد.

۴) مدل GDA و مقایسه با logistic regression

مدل GDA یکی از الگوریتم های یادگیری generative می باشد.اگر فرض ما برای توزیع مولد داده‌های x یا $P(x|y)$ گوسی چند متغیره باشد از روش GDA یا Gaussian Discriminant Analysis استفاده می کنیم.
برای توزیع گوسی چند متغیره داریم:

$$P(x;\mu,\Sigma)=\frac{1}{(2\pi)^{(\frac{n}{2})}|\Sigma|^{(\frac{1}{2})}}\exp\left(\left(\frac{-1}{2}\right)(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

بردار x، بردار میانگین μ با ابعاد n ، ماتریس کوواریانس Σ با ابعاد n*n می باشد.

$$\mu\in\mathbb{R}^n$$
$$\Sigma_{(n*n)}\in\mathbb{R}^{(n*n)}\geqslant 0$$

ماتریس کوواریانس PSD یا positive semi definite می‌باشد یعنی برای هر بردار z دلخواه داریم:

$$(Z^*)^T\Sigma Z\geqslant 0$$

تحلیل تابع:
در تابع سیگما هر چه مقادیر قطر بزرگ‌تر باشند، پخش شدگی بیشتری در آن بعد خواهیم داشت.
تأثیر عناصر غیر قطری به این صورت است که کوررلیشن مقادیر x ها را نشان می‌دهد و باعث می‌شود محور به سمتی انحراف پیدا کند.
احتمال پیش‌بینی در نزدیکی مرکز کانتور مربوط به یک کلاس بیشتر است. و هر چه از مرکز دور می‌شویم احتمال آن کاهش میابد.
مدل گوسی را می‌توان برای مسأله طبقه بندی باینری هم تعریف کرد.
مقایسه:
با توجه به توضیحات ما با استفاده از $P(x|y)$ می‌توانیم به $P(y|x)$ در مدل سازی logistic برسیم.
ولی عکس این قضیه برقرار نیست یعنی $P(y|x)$ در logistic لزوماً به $P(x|y)$ نمی‌رسد و می‌تواند هر توزیع دیگری هم باشد و لزوماً توزیع گوسی ندارد.
بنابر این در GDA فرضیات قوی تری برای داده ها در نظر گرفتیم و داده‌ها حتماً باید توزیع گوسی داشته باشند تا مدل GDA خوب عمل کند.
در صورتی که این شرایط را داشته باشد و بر داده‌ها منطبق باشد GDA روش بهتری نسبت به logistic regression است ولی logistic regression مقاوم‌تر و کم حساس‌تر به فرضیات خطا می باشد.
بنابر این اگر توزیع نرمال نیست روش logistic regression بهتر عمل می کند.
روش GDA برای آموزش به داده‌های کمی نیاز دارد. ولی اگر از توزیع داده‌ها مطمئن نیستیم بهتر است که از روش logistic regression استفاده کنیم.
بنابر این معمولاً روش logistic regression نسبت به GDA پرکاربرد تر است.

۵) مدل LDA , QDA

هر دو مدل برای یادگیری داده‌های آموزشی برای مسأله های طبقه بندی استفاده می شوند.
در هر دو روش فرض می‌شود که توزیع مشاهدات به صورت گوسی یا نرمال می باشد.
تفاوت اساسی این دو مدل در این است که در روش LDA فرض می‌شود که ماتریس کواریانس برای همه ویژگی‌ها با هر کلاسی یکسان می باشد. که در نتیجه آن باعث می‌شود که مرز خطی برای داده‌ها پیش‌بینی کند.
ولی در QDA متناسب با هر کلاس ماتریس کوواریانس را تشکیل می دهد. که باعث ایجاد مرز quadratic برای داده‌های می شود.

LDA:
در این روش discriminant scores را برای هر کلاس محاسبه می‌کنیم و کلاسی را انتخاب می‌کنیم که عدد آن بزرگ‌تر باشد.

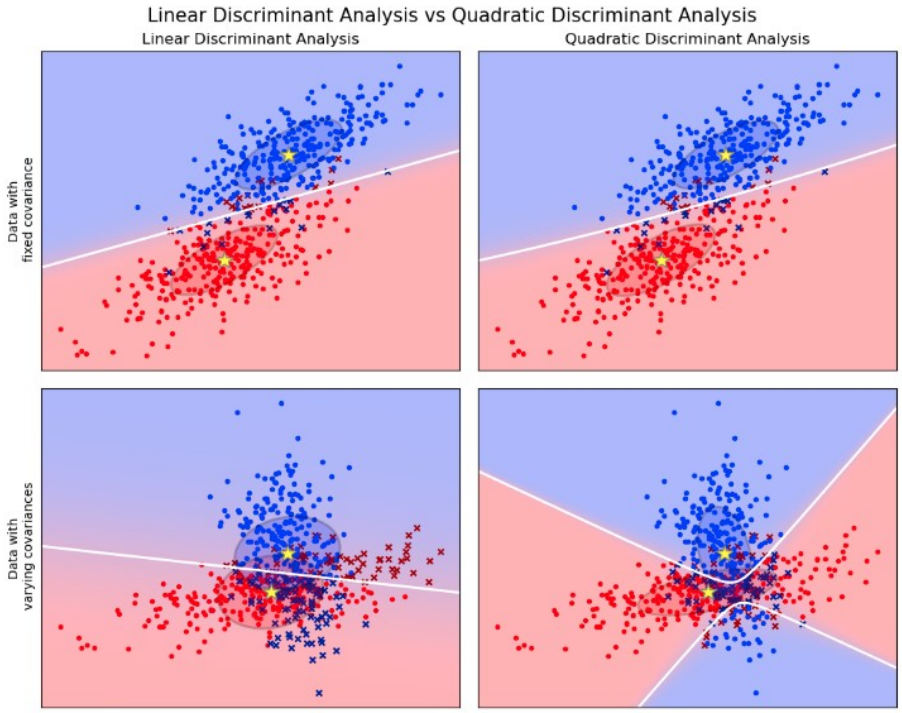
$$\hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k)$$

$$\hat{\delta}_k(x) = x^T \Sigma^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^T \Sigma^{-1} - \hat{\mu}_k + \log(\hat{\pi}_k)$$

QDA:

$$\hat{\delta}_k(x) = \frac{-1}{2} x^T \Sigma_k^{-1} x + x^T \Sigma_k^{-1} \hat{\mu}_k - \frac{1}{2} \mu_k^T \Sigma_k^{-1} \hat{\mu}_k - \frac{1}{2} \log|\Sigma_k| + \log(\hat{\pi}_k)$$

هنگامی که تعداد کلاس‌هازیاد است و در نتیجه تعداد ماتریس های کواریانس زیاد می شود، روش LDA دارای بایاس زیادی می‌شود و روش QDA مناسب‌تر است.
اما از طرفی زمانی که تعداد پارامتر ها زیاد باشند، استفاده از QDA به دلیل اینکه برای هر کلاس به طور جداگانه باید ماتریس کواریانس را محاسبه کنیم، دشوار است.
بنابر این یک trade_off بین بایاس و هزینه محاسبه واریانس می باشد.
این دو روش زمانی به logistic regression ترجیح داده می‌شوند که تعداد کلاس‌ها زیاد باشند.



سؤال دوم:

۱) Naive Bayes

فرض ما در روش Naive Bayes این است که با شرط داشتن y ویژگی‌ها از هم مستقل هستند. با استفاده از تخمین زن ML در Naive Bayes داریم:

$$ML: argmax\, P(Y|y)P(X=x|Y=y) = argmax\, P(Y|y)\prod P(X_i=x_i|Y_i=y_i)$$

با log گرفتن از عبارت بالا و max کردن log likelihood راحت‌تر به جواب می رسیم.
ابتدا داده‌های تست و آموزش را جدا می کنیم.

```
data = pd.read_csv('wdbc.data', header=None, sep=",")
data[1] = data[1].replace({'B': 0, 'M': 1})
test_ratio = 0.2

train_size = int(len(data)*(1-test_ratio))
data_train = data.loc[:train_size, :]

x_train = data.loc[0:train_size, 2:]
y_train = data.loc[0:train_size, 1]

x_test = data.loc[train_size:, 2:]
y_test = data.loc[train_size:, 1]
```

توزیع پیشین یا همان P(y) را به این صورت محاسبه می‌کنیم که تعداد ویژگی‌هایی که در کلاس صفر قرار دارند را بر تعداد کل داده‌های آموزشی تقسیم می‌کنیم و همین کار را برای کلاس یک هم انجام می دهیم.

```
#%% prior P(y)
prior = []
prior = (x_train.groupby(y_train).apply(lambda x: len(x))/train_size).to_numpy()
```

مقدار prior برای هر کلاس:

	0	1
0	0.59341	0.40879

با استفاده از داده آموزشی پارامتر های مدل گوسی را به دست می آوریم:

```
train_mean = x_train.groupby(y_train).apply(np.mean).to_numpy()
train_var = x_train.groupby(y_train).apply(np.var).to_numpy()
```

حال برای داده‌های تست مقدار x را در توزیع گوسی به دست آمده قرار می‌دهیم و کلاسی را که احتمال پسین بیشتری دارد انتخاب می کنیم.

```
def gaussian_probability(x_row,class_type, train_mean, train_var):
    a = np.exp((-1 / 2) * ((x_row - train_mean[class_type]) ** 2) / (2 * train_var[class_type]))
    b = np.sqrt(2 * np.pi * train_var[class_type])
    return a / b
```

در محاسبه posterior از همان فرمول گفته شده در ابتدا استفاده شده است.

```
predictions = []
for row in x_test.to_numpy():
    posteriors = {}
    for class_type in range(2):
        posterior = np.sum(np.log(gaussian_probability(row, class_type, train_mean, train_var))) + np.log(prior[class_type])
        posteriors[class_type] = posterior
    if posteriors[0] > posteriors[1]:
        predictions.append(0)
    else:
        predictions.append(1)
```

برای محاسبه دقت مدل گفته شده برای داده‌های تست، تعداد جواب‌های درس پیش‌بینی شده با استفاده از مدل را بر کل داده‌های تست تقسیم می کنیم.

```
#%% accuracy
accuracy = np.sum(y_test == predictions) / len(y_test)
```

accuracy:
0.9649122807017544

۲) احتمال خوش خیم بودن:

برای محاسبه دقت خوش خیم بودن باید دقت داده‌های تستی که مقدار ستون هدف آن‌ها خوش خیم بوده است را محاسبه کنیم.
یعنی تعداد پیشبینی های درست موارد خوش خیم بر روی تعداد کل خوش خیم ها.

```
acc = []
for i in range(len(y_test)):
    acc.append((predictions[i], y_test[i]))

benign_predicts = 0
benign_size = 0
for i in range(len(y_test)):
    if acc[i][1] == 0:
        benign_size += 1
    if acc[i][0] == acc[i][1] and acc[i][1] == 0:
        benign_predicts += 1
benign_accuracy = benign_predicts / benign_size
```

benign_accuracy = 0.9886

سؤال سوم:

۲) MLE vs MAP:

هر دو تخمین زن برای تخمین احتمال برای یک داده به جای به دست آوردن کل مدل احتمال به کار می روند.
:MLE

$$\theta_{MLE} = argmax_{\theta} P(X|\theta) = argmax_{\%that} \prod_i P(x_i|\theta)$$

به دلیل اینکه اگر این مقدار کمتر از ۱ باشد صفر و اگر به سمت بینهایت برود قابل محاسبه نمی باشد، معمولاً log این احتمال را محاسبه و max آن را محاسبه می کنند.

$$\theta_{MLE} = argmax_{\theta} \log P(X|\theta) = argmax_{\%that} \log \prod_i P(x_i|\theta) = argmax_{\theta} \Sigma logP(x_i|\theta)$$

تخمین MAP برای مدل های بیزین استفاده می شود. در این مدل ها برای θ یک توزیع پیشین $P(\theta)$ در نظر می‌گیریم و پس از دیدن مشاهدات با استفاده از MAP بیشینه توزیع‌های پسین را به دست می آوریم.

$$\theta_{MAP} = argmax_{\theta} \log P(X|\theta)P(\theta) = argmax_{\%that} \log \prod_i P(x_i|\theta) + logP(\theta) = argmax_{\theta} \Sigma logP(x_i|\theta) + logP(\theta)$$

درواقع اگر راجع به توزیع پیشین θ تخمینی داشته باشیم از روش MAP و در غیر این صورت از MLE استفاده می کنیم.
MLE مدل خاصی از MAP است که در آن توزیع θ به صورت یکنواخت یا uniform در نظر گرفته شده است.
در شرایطی که داده کمی داریم اگر توزیع θ را داشته باشیم یا بتوانیم فرضی برای آن در نظر بگیریم، روش MAP بهتر است.

۲) MLE with Boston dataset

مانند سؤال قبل داده‌ها را به دو بخش test و train تقسیم می کنیم.
میانگین داده‌های ستون هدف را به دست می آوریم و هر کدام که مقدار آن از میانگین کمتر بود به جای آن صفر و هر مقداری از هدف که از میانگین بیشتر بود را یک قرار می دهیم.
به این صورت داده‌های پیوسته را گسسته تبدیل می‌کنیم و می‌توانیم از classification برای پیش‌بینی محدوده قیمت خانه‌های تست استفاده کنیم.

```
data = pd.read_csv('Boston.csv')
medv_mean = data["medv"].mean()
data['medv'] = np.where(data['medv'] > medv_mean, 1, 0)
test_ratio = 0.2

train_size = int(len(data)*(1-test_ratio))
data_train = data.loc[1:train_size, :]

x_train = data.iloc[0:train_size, 1:14]
y_train = data.iloc[0:train_size, 14]

x_test = data.iloc[train_size:, 1:14]
y_test = data.iloc[train_size:, 14]

feature_size = len(x_train.iloc[1])
```

۳) MLE with Boston dataset

همانند سؤال قبل با استفاده از داده‌های آموزشی مقدار پارامتر های توزیع گوسی را به دست می آوریم و برای داده‌های تست تابع هدف را پیش‌بینی می کنیم.

```python
#%%
def gaussian_probability(x_row,class_type, train_mean, train_var):
    a = np.exp((-1 / 2) * ((x_row - train_mean[class_type]) ** 2) / (2 * train_var[class_type]))
    b = np.sqrt(2 * np.pi * train_var[class_type])
    return a / b
```

```python
#%%
train_mean = x_train.groupby(y_train).apply(np.mean).to_numpy()
train_var = x_train.groupby(y_train).apply(np.var).to_numpy()

np.seterr(divide='ignore')
predictions = []
for row in x_test.to_numpy():
    posteriors = {}
    for class_type in range(2):
        posterior = np.sum(np.log(gaussian_probability(row, class_type, train_mean, train_var)))
        posteriors[class_type] = posterior
    if posteriors[0] > posteriors[1]:
        predictions.append(0)
    else:
        predictions.append(1)
```

اگر ماتریس در هم ریختگی را به صورت زیر در نظر بگیریم:
ستون‌ها برای مقدار واقعی هدف برای داده تست و سطر نشان دهنده مقدار پیش‌بینی شده است

	0	1
0	TN	FP
1	FN	TP

خانه [0][0] → تعداد داده هایی که مقدار پیش‌بینی شده آن‌ها صفر و مقدار واقعی آن هم صفر است
خانه [1][1] →تعداد داده هایی که مقدار پیش‌بینی شده آن‌ها 1 و مقدار واقعی آن هم 1 است
خانه [0][1] →تعداد داده هایی که مقدار پیش‌بینی شده آن‌ها صفر و مقدار واقعی آن 1 است
خانه [1][0] →تعداد داده هایی که مقدار پیش‌بینی شده آن‌ها 1 و مقدار واقعی آن صفر است

با استفاده از کتابخانه pandas این ماتریس را به شکل زیر محاسبه می کنیم:

```python
#%% accuracy
data_confusion = pd.crosstab(y_test.reset_index(drop=True), pd.Series(predictions))
```

و مقدار آن برابر است با:

	0	1
0	83	9
1	7	3

دقت کل برابر است با: ۰.۸۴

۴) test ratio = 0.6

در این بخش دقیقاً مشابه قسمت قبل فقط test_ratio را برابر با ۰.۶ قرار می دهیم.
در مقایسه با بخش قبل دقت کاهش پیدا کرده است.
به دلیل اینکه تعداد داده‌های آموزشی از ۴۰۴ به ۲۰۲ کاهش پیدا کرده است دقت تست کاهش و دقت مدل افزایش پیدا کرده است بنابر این احتمالاً overfitting رخ داده است.
دقت :۰.۷۶
ماتریس در هم ریختگی:

	0	1
0	132	51
1	21	100

Refrences:

https://www.oreilly.com/library/view/generative-deep-learning/9781492041931/ch01.html#generative_model

https://medium.com/@akankshamalhotra24/generative-classifiers-v-s-discriminative-classifiers-1045f499d8cc#:~:text=Generative%20Classifiers%20tries%20to%20model,likely%20generated%20the%20given%20observation.

https://towardsdatascience.com/generating-passwords-with-generative-models-from-probabilistic-to-deep-learning-approaches-54d41d8810e3

https://stats.stackexchange.com/questions/103500/machine-learning-algorithms-to-handle-missing-data

مقاله: GAIN: Missing Data Imputation using Generative Adversarial Nets by Jinsung Yoon * James Jordon * Mihaela van der Schaar

Comparison of Generative and Discriminative Techniques for Object Detection and Classification y Ilkay Ulusoy 1 and Christopher M. Bishop2

https://towardsdatascience.com/implementing-naive-bayes-algorithm-from-scratch-python-c6880cfc9c41

https://dzone.com/articles/naive-bayes-tutorial-naive-bayes-classifier-in-pyt

https://wiseodd.github.io/techblog/2017/01/01/mle-vs-map/

https://github.com/Nitin1901/Confusion-Matrix