

۱۴۰۷

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشگاه مهندسی برق \_ گرایش کنترل

مینی پروژه شماره سه

یادگیری ماشین

نگارش

فاطمه امیری

۴۰۲۰۲۴۲۴

لينک گوگل كولب

لينک گیت هاب

استاد مربوطه

جناب آقای دکتر علیاری

بهار ۱۴۰۳

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

## فهرست مطالب

<b>۱</b>	<b>پرسش یک</b>
۲	بخش اول)
۱۶	بخش دوم)
۲۲	بخش سوم)
۲۸	بخش چهارم)
<b>۳۵</b>	<b>پرسش سه</b>
۳۵	بخش اول)
۴۰	بخش دوم)
۴۱	بخش سوم)
۵۰	بخش چهارم)
۵۳	بخش پنجم)
۵۵	بخش ششم)

## پرسش یک

هدف از این سوال آزمایش الگوریتم SVM در نمونه‌های مختلف روی دیتاست معروف گل زنبق<sup>۱</sup> است. مراحل زیر را یک به یک انجام دهید و موارد خواسته شده در گزارش خود به همراه کدها ارسال کنید.

در ابتدا به توضیح مختصری درباره این دیتاست می‌پردازیم:

دیتاست Iris یکی از پرکاربردترین و شناخته شده‌ترین مجموعه داده‌ها در حوزه یادگیری ماشین و آمار است. این دیتاست توسط رونالد فیشر، یکی از پیشگامان آمار و زیست‌شناسی، در سال ۱۹۳۶ معرفی شده است. دیتاست Iris برای آموزش و تست الگوریتم‌های یادگیری ماشین به خصوص در زمینه طبقه‌بندی مورد استفاده قرار می‌گیرد.

ویژگی‌های دیتاست Iris:

- تعداد نمونه‌ها: ۱۵۰ نمونه از گل‌های زنبق
- گونه‌های مختلف: سه گونه Iris virginica، Iris versicolor و Iris setosa که همان تارگت هستند که با لیبل‌های صفر یک و دو مشخص شده‌اند.

تعداد ویژگی‌ها: ۴ ویژگی که به صورت زیر هستند:

- طول کاسبرگ (Sepal Length) : طول کاسبرگ به سانتی‌متر
- عرض کاسبرگ (Sepal Width) : عرض کاسبرگ به سانتی‌متر
- طول گلبرگ (Petal Length) : طول گلبرگ به سانتی‌متر
- عرض گلبرگ (Petal Width) : عرض گلبرگ به سانتی‌متر

هدف دیتاست Iris :

هدف اصلی این دیتاست، طبقه‌بندی نمونه‌ها به یکی از سه گونه زنبق با استفاده از ویژگی‌های فوق است. به عبارت دیگر، با داشتن اطلاعات مربوط به طول و عرض کاسبرگ و گلبرگ، باید مشخص شود که نمونه مورد نظر به کدام یک از گونه‌های Iris virginica، Iris versicolor، Iris setosa تعلق دارد.

کاربردهای دیتاست Iris :

-آموزش الگوریتم‌های یادگیری ماشین: به دلیل ساده و متوازن بودن این دیتاست، به عنوان یک نقطه شروع عالی برای آموزش و تست الگوریتم‌های مختلف یادگیری ماشین به کار می‌رود.

-تحلیل داده‌ها: دیتاست Iris برای آموزش تکنیک‌های مختلف تحلیل داده‌ها و مصورسازی داده‌ها مورد استفاده قرار می‌گیرد.

-پژوهش‌های آکادمیک: بسیاری از مقالات و تحقیقات علمی در زمینه آمار و یادگیری ماشین از این دیتاست به عنوان نمونه‌ای برای تست مدل‌های پیشنهادی استفاده می‌کنند.

پس در کل معرفی این دیتاست توسط رونالد فیشر به عنوان یکی از اولین کاربردهای عملی آمار در زیست‌شناسی و علوم طبیعی، اهمیت تاریخی دارد. فیشر از این داده‌ها برای نشان دادن تکنیک‌های جدید آمار و تحلیل داده‌ها استفاده کرد که تاثیرات بزرگی بر روی توسعه این رشته‌ها داشت. دیتاست Iris با داشتن ساختاری ساده و متوازن، به محققان و دانشجویان کمک می‌کند تا مفاهیم پایه‌ای یادگیری ماشین و تحلیل داده‌ها را به خوبی فراگیرند و الگوریتم‌های مختلف را تست و ارزیابی کنند. ما در اینجا قصد داریم الگوریتم SVM را بررسی کنیم.

در بخش‌های بعدی به کار بر روی این دیتاست می‌پردازیم.

## بخش اول)

آ. در مرحله اول دیتاست را فراخوانی کنید و اطلاعاتی نظیر ابعاد، تعداد نمونه‌ها، میانگین، واریانس و همبستگی ویژگی‌ها را به دست آورید و نمونه‌های دیتاست را به تصویر بکشید (مثلاً با استفاده از t-SNE). سپس، با توجه به اطلاعات عددی، آماری و بصری بدست آمده، تحلیل کنید که آیا کاهش ابعاد می‌تواند در این دیتاست قابل استفاده باشد یا خیر.

در گام اول از sklearn داده‌ها را فراخوانی می‌کنیم، همان طور که در بالاتر گفتیم، داده‌ها شامل ۳ کلاس هستند که هم به صورت عددی (target) و هم به صورت کategirical آن‌ها را نمایش می‌دهیم:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	target_name	
0	5.1	3.5	1.4	0.2	0	setosa	
1	4.9	3.0	1.4	0.2	0	setosa	
2	4.7	3.2	1.3	0.2	0	setosa	
3	4.6	3.1	1.5	0.2	0	setosa	
4	5.0	3.6	1.4	0.2	0	setosa	
...	...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2	virginica	
146	6.3	2.5	5.0	1.9	2	virginica	
147	6.5	3.0	5.2	2.0	2	virginica	
148	6.2	3.4	5.4	2.3	2	virginica	
149	5.9	3.0	5.1	1.8	2	virginica	

150 rows × 6 columns

با توجه به شکل می بینیم که همانطور که در قبل گفتیم این دیتا ست دارای ۱۵۰ نمونه و ۴ ویژگی (که در تصویر زیر قابل مشاهده است) و سه کلاس است.

```
[8] iris.feature_names
```

```
['sepal length (cm)',  
 'sepal width (cm)',  
 'petal length (cm)',  
 'petal width (cm)']
```

```
len(iris.feature_names)
```

```
4
```

حالا لازم است چک شود که ایا این دیتاست بالانس است یا از روش هایی مانند undersampling برای بالانس کردن دیتاست باید استفاده کنیم !

```
[6] df['target_name'].value_counts()
```

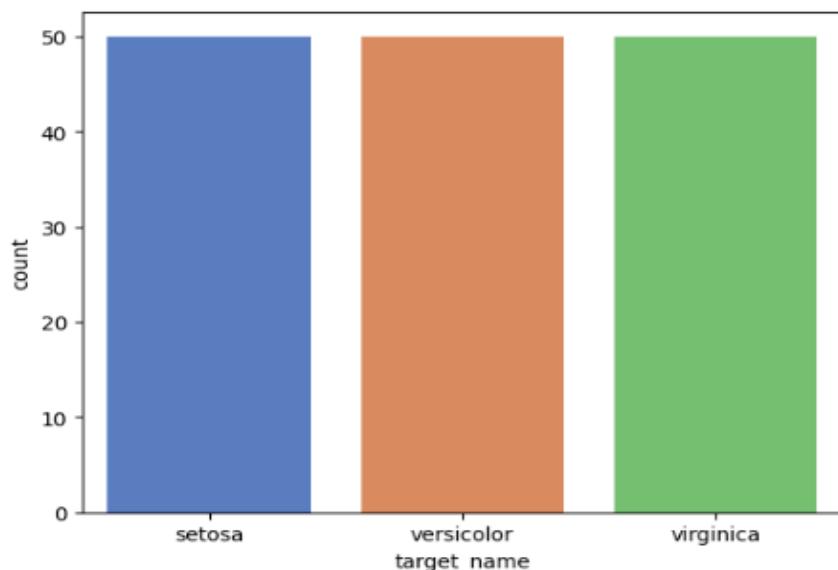
```
target_name
```

target_name	count
setosa	50
versicolor	50
virginica	50

```
Name: count, dtype: int64
```

```
sns.countplot(x='target_name', data=df, palette="muted")
```

```
<Axes: xlabel='target_name', ylabel='count'>
```



مشاهده می شود که دیتاست کاملا بالانس می باشد.

اکنون از دستور `df.describe()` استفاده می کنیم که با استفاده از کتابخانه `pandas`، یک خلاصه آماری از داده های عددی موجود در `df` DataFrame را بر می گرداند که شامل تعداد غیر تهی داده ها، میانگین، انحراف معیار، حداقل مقدار، صدک های بیست و پنجم، پنجاهم و هفتاد و پنجم، و حداکثر مقدار در هر ستون است.

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

مشاهده می کنیم که میانگین طول کاسبرگ حدود ۶ سانتی متر و عرض آن ۳ سانتی متر است، در حالی که میانگین طول گلبرگ حدود ۴ سانتی متر و عرض آن ۱ سانتی متر است. از نظر پراکندگی، طول گلبرگ بیشترین انحراف معیار را دارد و پس از آن طول کاسبرگ، عرض گلبرگ و عرض کاسبرگ قرار می گیرند. بیشترین و کمترین مقادیر این ویژگی ها نیز در شکل فوق نشان داده شده اند.

حال میانگین و میانه ویژگی ها را برای هر کلاس (تفکیک کلاس ها) بررسی می کنیم :

در شکل های زیر، میانگین و میانه و واریانس ویژگی ها برای هر کلاس از گل های زنبق نمایش داده شده است. از این نمودار می توان نتیجه گرفت که Iris-virginica بیشترین طول کاسبرگ را دارد و پس از آن Iris-versicolor قرار دارد. همچنین، تفاوت میانگین عرض گلبرگ بین سه کلاس قابل توجه است و می توان پیش بینی کرد که با استفاده از این ویژگی، کلاس Iris-setosa را به راحتی از دو کلاس دیگر جدا کرد.

[11] mean = df.groupby('target_name').mean() mean	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
target_name					
setosa	5.006	3.428	1.462	0.246	0.0
versicolor	5.936	2.770	4.260	1.326	1.0
virginica	6.588	2.974	5.552	2.026	2.0

```
med = df.groupby('target_name').median()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
target_name					
setosa	5.0	3.4	1.50	0.2	0.0
versicolor	5.9	2.8	4.35	1.3	1.0
virginica	6.5	3.0	5.55	2.0	2.0

و در زیر واریانس فیچر ها را برای هر کلاس مشاهده می کنیم :

```
variance = df.groupby('target_name').var()
```

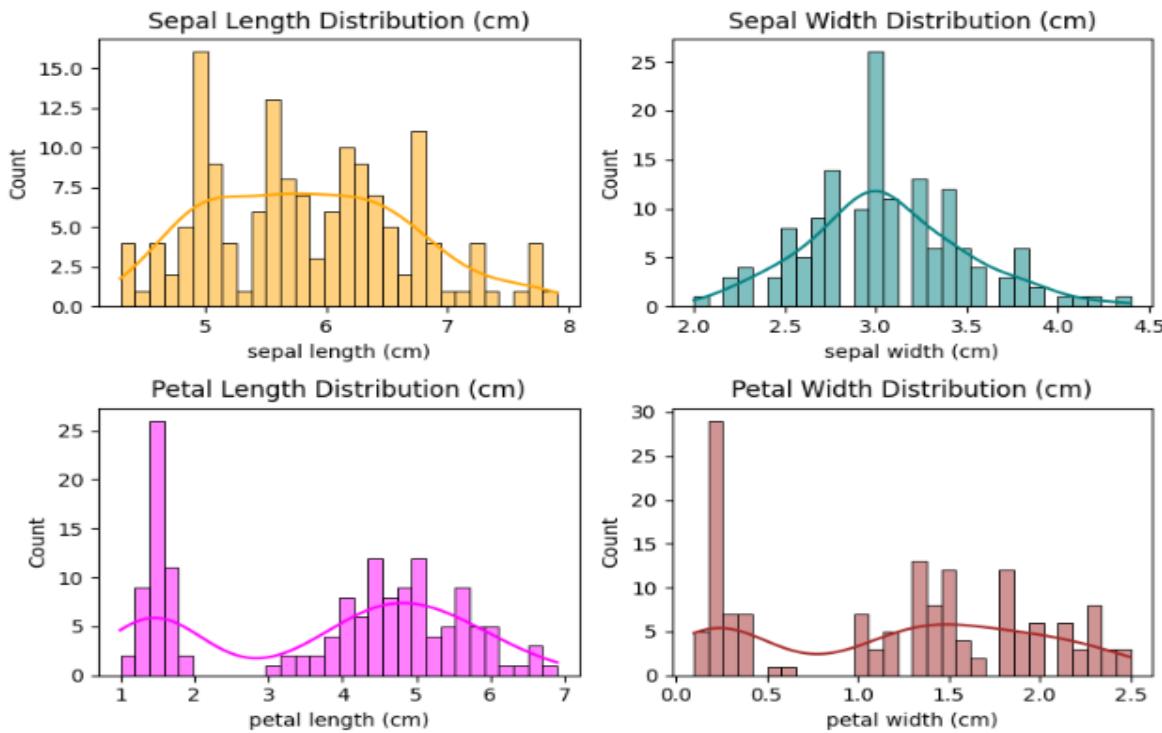
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
target_name					
setosa	0.124249	0.143690	0.030159	0.011106	0.0
versicolor	0.266433	0.098469	0.220816	0.039106	0.0
virginica	0.404343	0.104004	0.304588	0.075433	0.0

اکنون با توجه به شکل زیر و قسمتی که بالا نس بودن دیتابست چک شد، متوجه میشویم که داده Nan یا missing values نداریم :

```
| df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   sepal length (cm)    150 non-null   float64
 1   sepal width (cm)     150 non-null   float64
 2   petal length (cm)    150 non-null   float64
 3   petal width (cm)     150 non-null   float64
 4   target              150 non-null   int64  
 5   target_name          150 non-null   object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

به بررسی نمایش توزیع ویژگی ها با استفاده از نمودار میله ای، می پردازیم.



از شکل بالا می‌توانیم نکات زیر را برداشت کنیم:

#### توزیع طول و عرض کاسبرگ‌ها (Sepal) :

- توزیع نرمال با یک قله (Peak) : طول و عرض کاسبرگ‌ها دارای توزیعی تقریباً نرمال با یک قله هستند. این بدان معناست که بیشتر داده‌ها در اطراف یک مقدار مرکزی متتمرکز شده‌اند و از این مقدار به طور متقارن به دو طرف پراکنده می‌شوند. این نوع توزیع نشان می‌دهد که ویژگی‌های طول و عرض کاسبرگ‌ها در گونه‌های مختلف گل‌های زنبق دارای پراکندگی مشابهی هستند و تغییرات زیادی بین این مقادیر مشاهده نمی‌شود.

#### توزیع طول و عرض گلبرگ‌ها (Petal) :

- توزیع دو قله‌ای (Bimodal) : بر خلاف کاسبرگ‌ها، توزیع طول و عرض گلبرگ‌ها به صورت دو قله‌ای یا bimodal است. این بدان معناست که داده‌ها در دو نقطه مجزا متتمرکز شده‌اند که نشان‌دهنده وجود دو گروه مشخص در داده‌ها است. این نوع توزیع معمولاً به این دلیل رخ می‌دهد که ویژگی‌های طول و عرض گلبرگ‌ها در گونه‌های مختلف زنبق تفاوت‌های قابل توجهی دارند

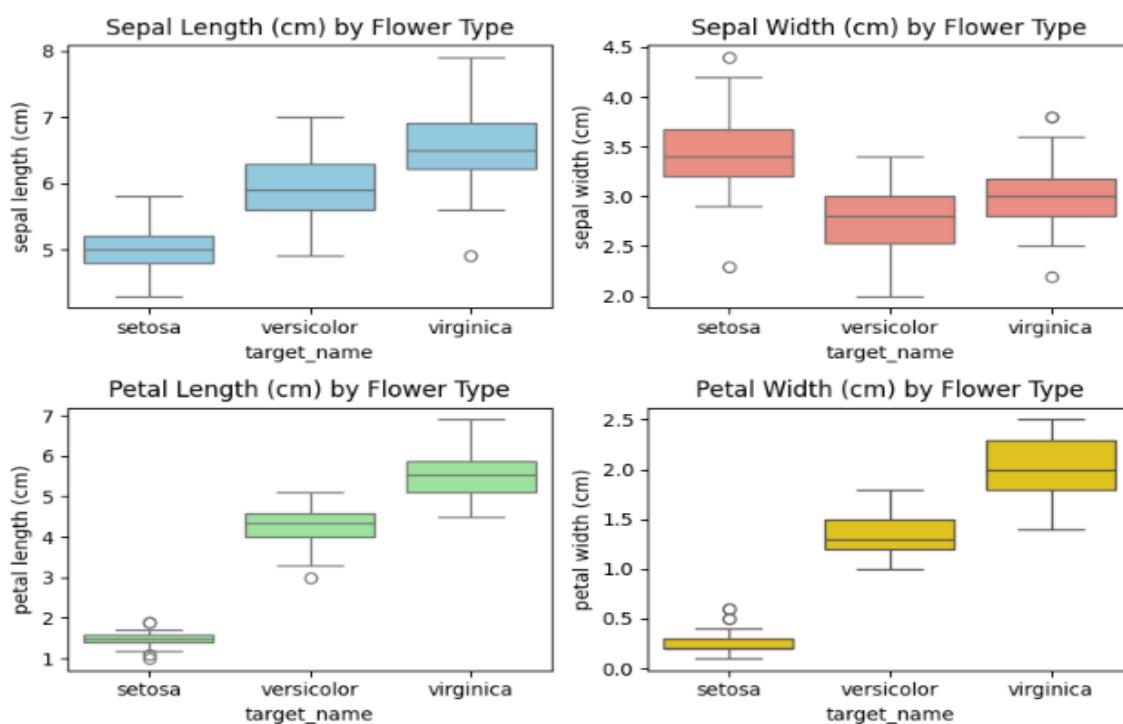
ویژگی‌های طول و عرض گلبرگ‌ها از اهمیت بالایی در تفکیک کلاس‌های مختلف گل‌های زنبق برخوردار هستند، زیرا تفاوت‌های مشاهده شده در توزیع این ویژگی‌ها نشان می‌دهد که می‌توانند به طور موثری گروه‌های متمایزی از داده‌ها را تشخیص دهند. به عنوان مثال، وجود دو قله در توزیع طول و عرض گلبرگ‌ها نشان‌دهنده این است

که این ویژگی‌ها دارای تفاوت‌های قابل توجهی بین گونه‌های مختلف گل‌های زنبق هستند. این تفاوت‌های توزیعی می‌توانند به عنوان اطلاعات مفیدی در مدل‌های یادگیری ماشین مورد استفاده قرار گیرند، به‌ویژه در الگوریتم‌های طبقه‌بندی که بهبود دقت و کارایی خود در تشخیص گونه‌های مختلف گل‌های زنبق را به دنبال دارند. این تحلیل از مهمترین ابعاد کاربردی و تفسیری داده‌های آماری ویژگی‌های طول و عرض گلبرگ‌ها در زمینه یادگیری ماشین است که به بهبود عملکرد مدل‌های پیش‌بینی کمک می‌کند.

پس در کل، با توجه به تحلیل توزیع ویژگی‌ها، می‌توان نتیجه گرفت که ویژگی‌های طول و عرض گلبرگ‌ها به دلیل داشتن توزیع دو قله‌ای، نقش مهمی در جدا کردن کلاس‌های مختلف داده‌ها دارند. این ویژگی‌ها می‌توانند به عنوان شاخص‌های کلیدی در فرآیند طبقه‌بندی مورد استفاده قرار گیرند و مدل‌های یادگیری ماشین می‌توانند از این اطلاعات برای بهبود عملکرد خود استفاده کنند. این تحلیل به ما کمک می‌کند تا بفهمیم که کدام ویژگی‌ها بیشتر در تشخیص گونه‌های مختلف زنبق موثر هستند و به این ترتیب می‌توانیم مدل‌های دقیق‌تری را طراحی کنیم.

در مرحله بعد از آن که داده‌ها را بر اساس هر کلاس (نوع گل) دسته‌بندی کردیم، نمودارهای جعبه‌ای (boxplot) را برای هر یک از ویژگی‌ها به ازای هر کلاس رسم می‌کنیم. این نمودارها به ما این امکان را می‌دهند که به طور دقیق‌تری به توزیع داده‌ها در هر کلاس نگاه کنیم و الگوهای مختلفی که در داده‌ها وجود دارد را تشخیص دهیم.

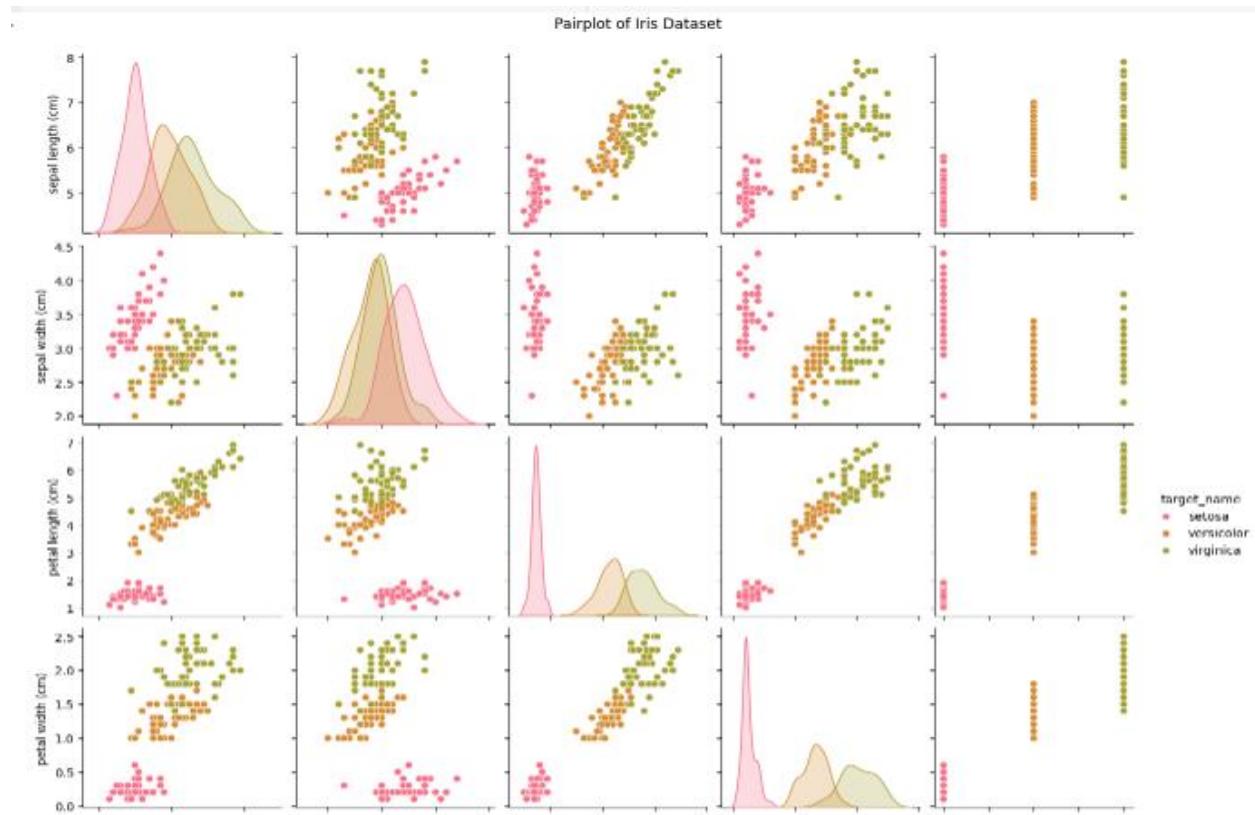
نمودار جعبه‌ای به صورت شکل زیر است :



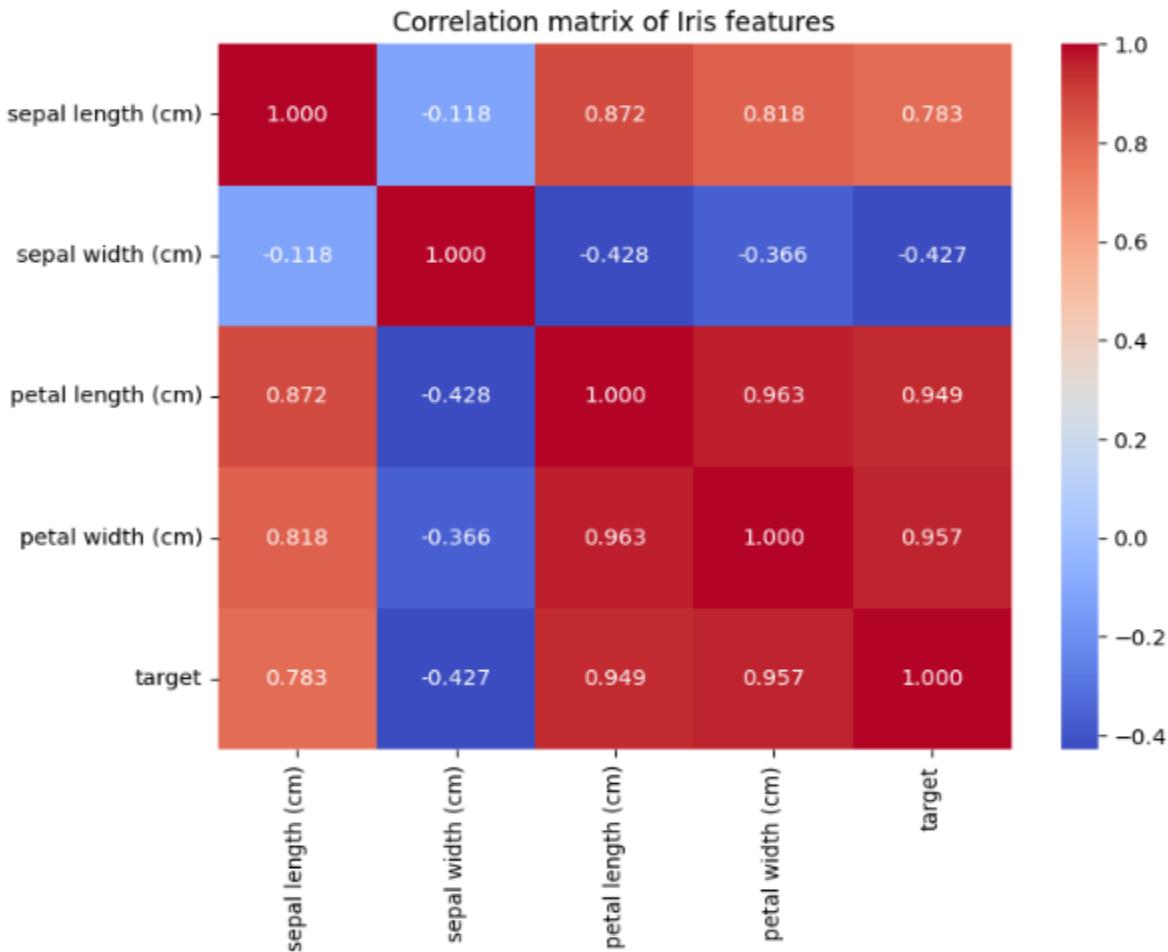
در شکل بالا که شامل نمودارهای جعبه‌ای است، هر ویژگی مانند طول و عرض کاسبرگ و گلبرگ بر اساس نوع گل (کلاس) نمایش داده می‌شود. از آنجا که این نمودارها بر اساس توزیع ویژگی‌های گلبرگ رسم شده‌اند، مشخص است که این ویژگی‌ها می‌توانند به خوبی کلاس‌های مختلف گل‌های زنبق را از یکدیگر جدا کنند. به طور خاص، کلاس Iris-setosa بر اساس این ویژگی‌ها قدرتمندی برای تمایز از سایر کلاس‌ها دارد، به گونه‌ای که انتظار می‌رود با دقت ۱۰۰ درصد این کلاس را از سایر کلاس‌ها جدا کیم.

دو کلاس دیگر، مانند Iris-virginica و Iris-versicolor، از نظر ویژگی‌های مربوط به کاسبرگ به هم نزدیک‌تر هستند. با این حال، با توجه به ویژگی‌های مربوط به گلبرگ، هنوز می‌توان آن‌ها را با دقت قابل قبولی از یکدیگر جدا کرد. این تحلیل نشان می‌دهد که ویژگی‌های مربوط به گلبرگ، به عنوان ابزاری موثر در دسته‌بندی و تمایز کلاس‌های مختلف گل‌های زنبق عمل می‌کنند و در ارتقای عملکرد مدل‌های یادگیری ماشین برای این اهداف بسیار موثر هستند.

همچنین با استفاده از نمودار Pairplot و پالت رنگی سفارشی، تفاوت‌ها و الگوهای مختلف بین ویژگی‌های داده‌های مجموعه Iris را به وضوح نمایش می‌دهیم. این نمودار به طور واضح و روشن تصویری از توزیع داده‌ها ارائه می‌دهد و به ما این امکان را می‌دهد که با استفاده از رنگ‌ها، الگوها و تفاوت‌های بین دسته‌های مختلف را به صورت بصری درک کنیم. نمودار به صورت زیر است :

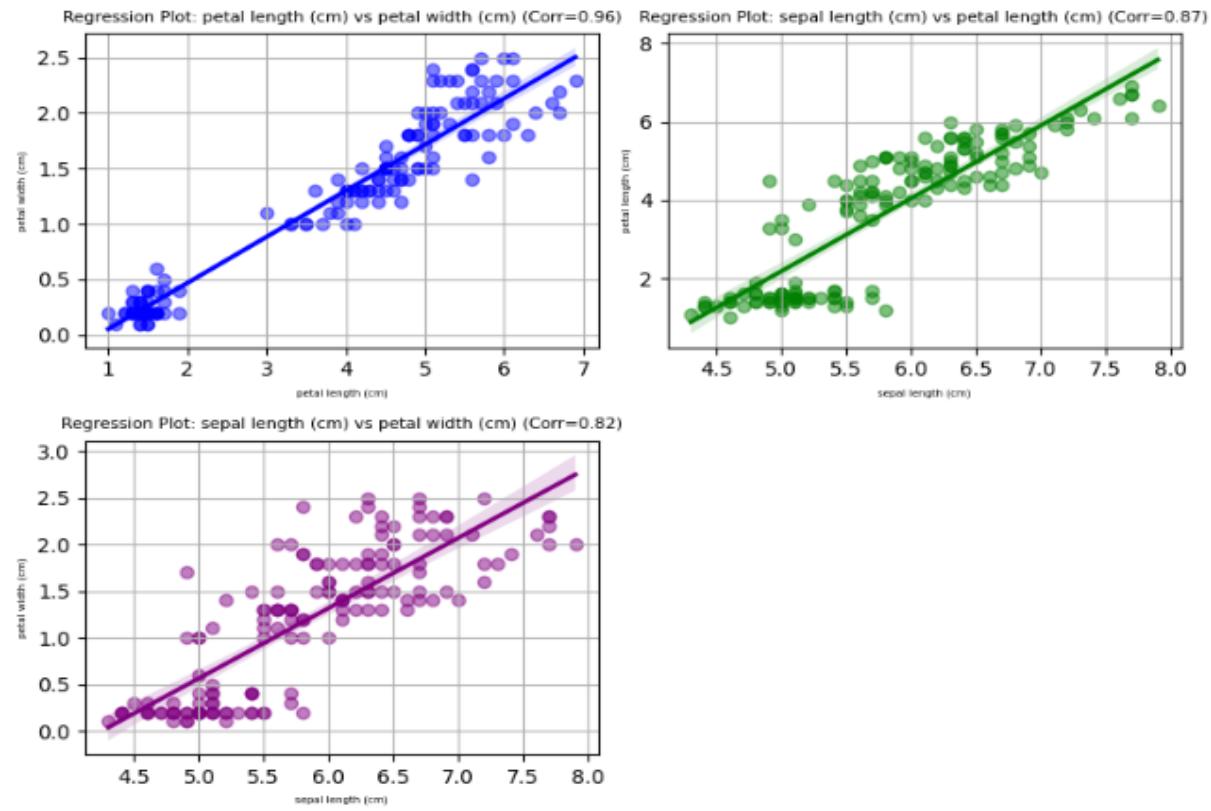


در ادامه کرولیشن ماتریکس را که همبستگی بین ویژگی ها را نشان می دهد برای دیتاست پلات می کنیم:



با توجه به شکل مشاهده می شود که ویژگی هایی که بیشترین همبستگی را با برچسبها (لیبل ها) دارند به ترتیب عرض گلبرگ، طول گلبرگ و عرض کاسبرگ هستند. این نتیجه نشان می دهد که این ویژگی ها می توانند بهترین نماینده ها برای تفکیک و دسته بندی داده ها به ازای برچسبها باشند. علاوه بر این، مشاهده می شود که این ویژگی ها با یکدیگر همبستگی خطی دارند، به این معنی که تغییر در یک ویژگی ممکن است با تغییر در ویژگی دیگر همراه باشد. به عنوان مثال، دو ویژگی عرض گلبرگ و طول گلبرگ به ضریب ۰.۹۶ با یکدیگر همبستگی دارند، که نشان می دهد این دو ویژگی با یکدیگر نزدیک و مرتبط هستند.

برای درک بهتر این ارتباطات، شکل زیر نیز ترسیم شده است که نمایش دقیق تری از همبستگی ها و توزیع ویژگی ها را فراهم می کند. این تحلیل ها می توانند به ما در انتخاب بهترین ویژگی ها برای استفاده در مدل های یادگیری ماشین کمک کنند و باعث بهبود عملکرد و دقت این مدل ها در پیش بینی و دسته بندی داده ها شوند. (با استفاده از sns.regplot، یک نمودار رگرسیون برای ویژگی های X و Y رسم می شود).



## به تصویر کشیدن نمونه های دیتاست

به تصویر کشیدن دیتاست با استفاده از متدهای t-SNE (t-distributed Stochastic Neighbor PCA (Principal Component Analysis) و Embedding) دو روش معمول و کارآمد در تجزیه و تحلیل داده های بزرگ و پیچیده در زمینه های مختلف می باشد و در ادامه به توضیح مختصراً درباره این روش ها می پردازیم و دیتاست را با استفاده از این متدها به تصویر می کشیم.

### ۱. PCA (Principal Component Analysis)

PCA یک روش تجزیه عاملی است که برای کاهش ابعاد داده های پیچیده استفاده می شود. هدف اصلی آن کاهش تعداد متغیرها (ویژگی ها) در دیتاست است به طوری که ویژگی های جدیدی به نام کامپوننت های اصلی بدست آید که می توانند حدود زیادی از تغییرات داده را توضیح دهند. این کاهش ابعاد اغلب به چند کامپوننت اصلی ترکیب شده که توانایی حفظ اطلاعات مهم دیتاست را داشته باشند.

### ۲. t-SNE (t-distributed Stochastic Neighbor Embedding)

t-SNE نیز یک روش کاهش ابعاد است، اما با تمرکز بر حفظ توابع فاصله میان نقاط در فضا. این روش به خوبی برای نگهداری از ارتباطات نزدیک و دور بین نقاط در داده‌ها مناسب است. تفاوت اصلی t-SNE با PCA در این است که PCA به دنبال کاهش ابعاد است که بیشترین تغییرات داده را در محورهای جدید توضیح دهد، در حالی که t-SNE به دنبال حفظ توابع فاصله بین نقاط است.

PCA و t-SNE هر دو روش مهمی در حوزه تحلیل داده‌ها هستند، با اهداف متفاوتی از جمله کاهش ابعاد و تصویرسازی داده‌ها. PCA بیشتر برای کاهش ابعاد داده‌ها و استفاده در مدل‌های یادگیری ماشین، بهبود دقت الگوریتم‌ها و کاهش پیچیدگی محاسباتی مورد استفاده قرار می‌گیرد. از سوی دیگر، t-SNE بهترین کارایی را در تصویرسازی داده‌های پیچیده در فضای دو یا سه بعدی دارد و به دست آوردن نمودارهایی که روابط پیچیده و چندگانه بین داده‌ها را نمایش دهد، انجام می‌دهد. هر کدام از این روش‌ها بسته به نیازهای تحلیلی و هدف اصلی مطالعه داده‌ها، می‌توانند به صورت موثر واقع شوند و در درک بهتر داده‌ها کمک کنند.

در نهایت، انتخاب بین PCA و t-SNE بستگی به مسئله مورد نظر، نوع داده‌ها و هدف اصلی تحلیل دارد که هر یک از این روش‌ها می‌توانند برای کاهش ابعاد و تصویرسازی داده‌ها به صورت موثر مورد استفاده قرار گیرند.

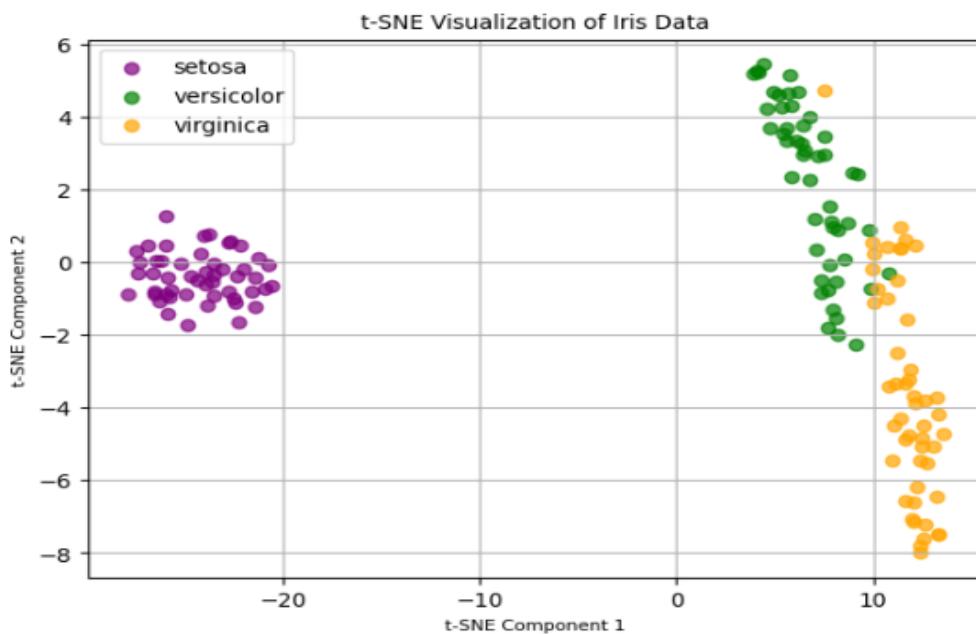
#### در ادامه از این روش‌ها برای به تصویر کشیدن نمونه‌های دیتاست استفاده می‌کنیم.

ابتدا، t-SNE را به کار می‌بریم که همانطور که گفتیم یک روش غیرخطی برای بصری‌سازی داده‌ها است. این الگوریتم بر اساس شباهت بین داده‌ها عمل می‌کند و از نظریه‌های احتمالاتی برای ایجاد یک نمایش بصری مناسب استفاده می‌کند. هدف t-SNE این است که کلاسترها و الگوهایی که ممکن است در ابعاد بالاتر پنهان باشند، به وضوح نشان داده شوند. یکی از مزایای مهم این روش، قابلیت نمایش بصری داده‌های ناهنجار و کلاسترها است. با این حال، t-SNE از نظر محاسباتی سنگین است و می‌تواند در ابعاد بالا کارآمدی خود را از دست بدهد. در چنین مواردی، استفاده از روش‌های خطی مانند PCA ممکن است مناسب‌تر باشد.

PCA یک الگوریتم خطی است که برای کاهش ابعاد داده‌ها استفاده می‌شود. این الگوریتم به دنبال پیدا کردن جهاتی است که در آن واریانس داده‌ها بیشترین مقدار را دارد. در PCA، ابتدا ماتریس کوواریانس داده‌ها محاسبه می‌شود، سپس مقادیر و بردارهای ویژه به دست می‌آیند و داده‌ها به کمک این بردارهای ویژه به فضاهای جدید نگاشت می‌شوند. در این فضاهای، بردارهای ویژه نسبت به هم عمود هستند و داده‌ها به گونه‌ای توزیع می‌شوند که واریانس در آن‌ها حداقل باشد. این فرآیند به کاهش ابعاد داده‌ها کمک می‌کند و اطلاعات اصلی داده‌ها را حفظ می‌کند.

دو هایپرپارامتر مهم در t-SNE وجود دارد:

۱. n-components : تعداد بعدهایی که می خواهیم دادهها به آن نگاشت شوند. (۲ در نظر گرفتیم)
  ۲. perplexity : تعداد نزدیکترین همسایگان که در الگوریتمهای یادگیری ماشین چندگانه استفاده می شود.
- با استفاده از t-SNE نمایش بصری نمونه ها به صورت زیر است : ( میبینیم به خوبی تفکیک شده اند مخصوصا کلاس بنفش رنگ)



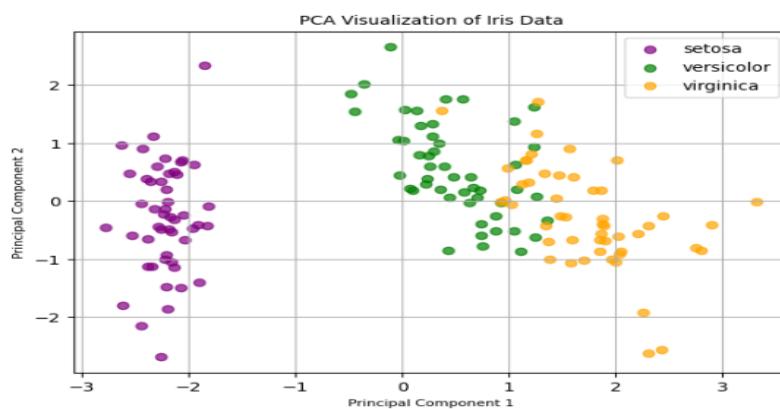
و با استفاده از PCA داریم :

با توجه به مشاهدهای قبلی، همان‌طور که توضیح داده شد، طول کاسبرگ (sepal length) و طول گلبرگ (petal length) همبستگی بالایی با یکدیگر دارند. به همین دلیل ، می‌توانیم راحت‌تر این دو ویژگی را از هم جدا کنیم. در مقابل، عرض کاسبرگ (sepal width) و طول کاسبرگ همبستگی بسیار کمی دارند، بنابراین نمی‌توان این دو ویژگی را به راحتی از هم تفکیک کرد و کاهش بعد را بر روی آن‌ها انجام داد.

بر اساس این مشاهدات، فرض اولیه ما این است که از چهار ویژگی موجود، می‌توانیم سه ویژگی را به یک ویژگی کاهش دهیم و به این ترتیب دو ویژگی اصلی خواهیم داشت. ابتدا این کار را با استفاده از تحلیل مؤلفه اصلی (PCA) با دو مؤلفه انجام می‌دهیم تا نتیجه را ببینیم.

تحلیل مؤلفه اصلی (PCA) یک تبدیل کننده خطی است که داده‌ها را بر روی مختصات متعامد (orthogonal) پخش می‌کند. این مختصات که به آن‌ها مؤلفه‌های اصلی (principal components) گفته می‌شود، به گونه‌ای هستند که بیشترین واریانس داده‌ها در مؤلفه اول یافت می‌شود و واریانس در مؤلفه‌های بعدی به صورت نزولی کاهش می‌یابد.

یکبار با کتابخانه Sklearn و یک بار بدون ان رسم می‌کنیم. (ابتدا در حالت دو مؤلفه اصلی)



از کد زیر استفاده کردیم و نتیجه به صورت نمودار فوق شد :

```
X = df[iris.feature_names].values
X_mean = np.mean(X, axis=0)
X_std = np.std(X, axis=0)
X_standardized = (X - X_mean) / X_std

cov_matrix = np.cov(X_standardized, rowvar=False)

eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

sorted_index = np.argsort(eigenvalues)[::-1]
sorted_eigenvalues = eigenvalues[sorted_index]
sorted_eigenvectors = eigenvectors[:, sorted_index]

# Select the top k eigenvectors (here we select top 2 for 2D visualization)
k = 2
eigenvector_subset = sorted_eigenvectors[:, 0:k]
X_reduced = np.dot(X_standardized, eigenvector_subset)

df_pca = pd.DataFrame(data=X_reduced, columns=['PC1', 'PC2'])
df_pca['target'] = df['target']
df_pca['target_name'] = df['target'].map(dict(enumerate(iris.target_names)))

plt.figure(figsize=(7, 5))
colors = ['purple', 'green', 'orange']
for target_name, color in zip(iris.target_names, colors):
    indices_to_plot = df_pca['target_name'] == target_name
    plt.scatter(df_pca.loc[indices_to_plot, 'PC1'],
                df_pca.loc[indices_to_plot, 'PC2'],
                label=target_name,
                color=color,
                alpha=0.7)

plt.xlabel('Principal Component 1', fontsize=8)
plt.ylabel('Principal Component 2', fontsize=8)
plt.title('PCA Visualization of Iris Data', fontsize=10)
plt.legend()
plt.grid(True)
plt.show()
```

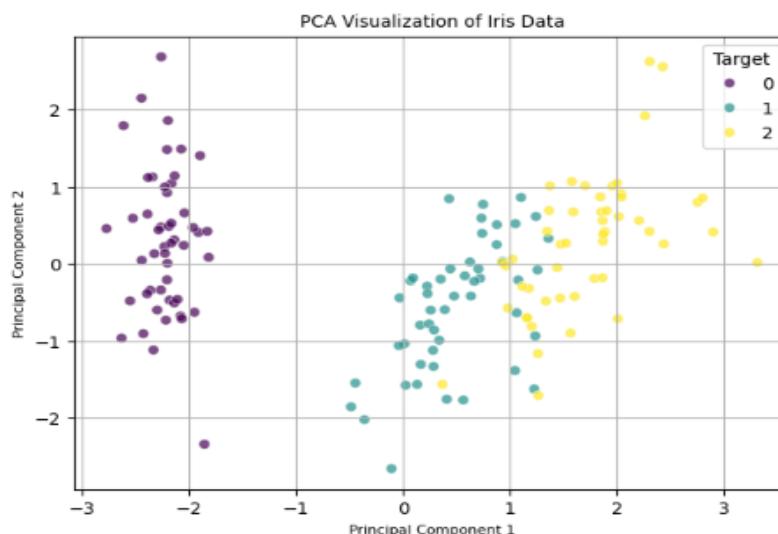
حالا یک بار هم از PCA برای sklearn با استفاده از کد زیر استفاده کردم ( این کد ساده‌تر و بدون نیاز به محاسبات دستی برای محاسبه ماتریس کوواریانس و بردارهای ویژه، از قابلیت‌های کتابخانه‌های sklearn و seaborn بهره می‌برد و تمرکز بیشتری بر روی تحلیل و بصری‌سازی داده‌ها دارد.)

```
#PCA using sklearn #method2
scaler = StandardScaler()
X_std = scaler.fit_transform(df[iris.feature_names])

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_std)

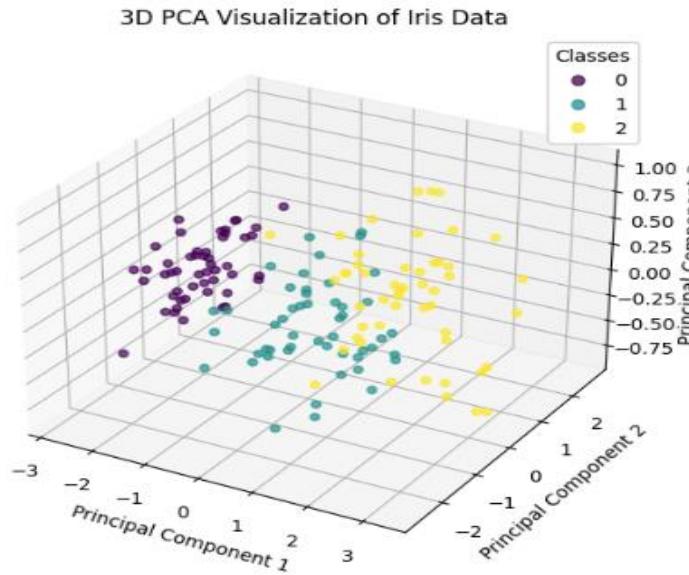
df_pca = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
df_pca['target'] = iris.target
```

و نتیجه به صورت شکل زیر است : (حالت دو مؤلفه اصلی )



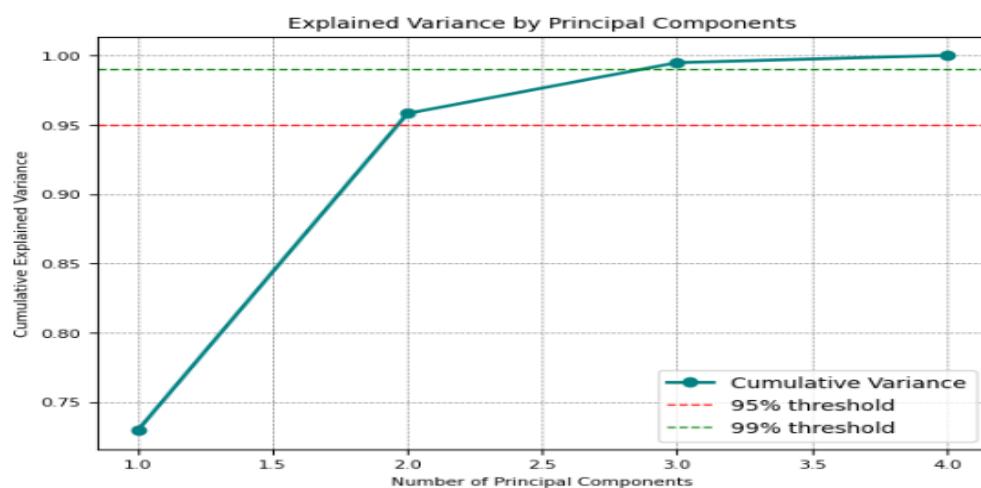
همان‌طور که مشاهده می‌کنید، کلاس داده‌های ۰ یعنی بنفس رنگ به راحتی از هم جدا می‌شوند، در حالی که کلاس‌های داده‌های ۱ و ۲ کمی نزدیک به هم هستند ولی باز هم می‌توان آن‌ها را به خوبی جدا کرد. این نتایج نشان می‌دهد که استفاده از دو مؤلفه اصلی به خوبی توانسته داده‌ها را تفکیک کند و می‌توان این کار را با سه مؤلفه نیز انجام داد.

در نتیجه، با توجه به این که تحلیل مؤلفه اصلی (PCA) با دو مؤلفه توانسته داده‌ها را به خوبی تفکیک کند، می‌توان انتظار داشت که با استفاده از سه مؤلفه نیز این تفکیک به خوبی انجام شود. این نکته نشان می‌دهد که کاهش بعد با استفاده از PCA می‌تواند ابزاری موثر برای تحلیل و تفکیک داده‌ها باشد. برای سه مؤلفه نمودار حاصل به صورت زیر است :



مشاهده می شود که نتیجه مانند دو مولفه، مطلوب است. پس همان دو مولفه انتخاب بهتری است.

همچنین، برای بررسی قابلیت کاهش ابعاد، از متدهای explained-variance-ratio استفاده می شود. این متدها نشان می دهد که چند ویژگی برای توصیف داده ها مناسب تر هستند. با بررسی مقادیر مختلف Components، مشاهده می شود که که PC1 و PC2 به ترتیب حدود ۷۲,۹۶٪ و ۸۵٪ از واریانس داده ها را توضیح می دهند، و مجموعاً این دو مولفه حدود ۹۵,۸۱٪ از واریانس داده ها را توضیح می دهند. این روش، نه تنها برای کاهش ابعاد و ساده سازی داده ها مفید است بلکه به تحلیل و تفسیر دقیق تر داده های پیچیده نیز کمک می کند، و به ما امکان می دهد با استفاده از داده های کمتر، اطلاعات بیشتری را درباره ویژگی های اصلی داده ها به دست آوریم. این نتایج در نمودار زیر نشان داده شده است.



```
Explained variance by each component: [0.72962445 0.22850762 0.03668922 0.00517871]
Cumulative explained variance: [0.72962445 0.95813207 0.99482129 1.]
```

علاوه بر این نمودار بالا، ماتریس همبستگی که در صفحات قبل رسم شد نیز نشان می‌دهد که برخی از ویژگی‌ها با هم همبستگی زیادی دارند، بنابراین می‌توان یکی از این ویژگی‌ها را حذف کرد تا داده‌ها ساده‌تر شوند.

در نهایت، نتایج بصری‌سازی داده‌ها با استفاده از روش‌های PCA و t-SNE نشان می‌دهد که کلاس Iris-setosa به راحتی از بقیه کلاس‌ها جدا می‌شود. این امر نشان می‌دهد که این ویژگی‌ها می‌توانند ابزار قدرتمندی برای استفاده در مدل‌های یادگیری ماشین باشند.

پس در کل کاهش ابعاد در علم داده یک ابزار قدرتمند برای فهم بهتر و کاهش پیچیدگی داده‌ها است. در دیتابست Iris که شامل چهار ویژگی و سه کلاس است، از تکنیک‌های مختلفی برای کاهش ابعاد استفاده می‌شود. تکنیک PCA مولفه‌هایی را شناسایی می‌کند که بیشترین تغییرات را توضیح می‌دهند، t-SNE به کشف ساختارهای پیچیده و روابط غیرخطی کمک می‌کند. تصویرسازی با این تکنیک‌ها می‌تواند دیدگاه بصری و مفیدی از توزیع داده‌ها و روابط بین نمونه‌ها ارائه دهد. در نتیجه، کاهش ابعاد به ساده‌سازی داده‌ها و فهم بهتر ساختار و روابط موجود در آن‌ها کمک می‌کند.

## بخش دوم)

ب. با استفاده از الگوریتم SVM، با هسته خطی، داده‌ها را طبقه‌بندی کنید و ماتریس درهم‌ریختگی آن را بدست آورید و مرزهای تصمیم‌گیری را در فضای دوبعدی (کاهش بعد از طریق یکی از روش‌های آموخته شده با ذکر دلیل) ترسیم کنید.

برای این قسمت سوال، گام‌ها و مراحل زیر را طی می‌کنیم :

۱. تقسیم داده : ابتدا داده‌ها به دو بخش اصلی تقسیم می‌شوند، یعنی بخش آموزش و بخش تست. این کار به منظور ارزیابی بهتر عملکرد مدل در شرایط واقعی و اعمال عملیاتی است که می‌خواهیم مدلمان را بر روی آن اعمال کنیم. تقسیم به این دو بخش از اهمیت بسیاری برخوردار است تا مدل با دقت بالاتری طراحی شود و دسته بندی بهتری ارائه دهد.

```
[ ] X_train, X_test, y_train, y_test = train_test_split(df[iris.feature_names], df['target'], test_size=0.2, random_state=24)
```

```
→ ((120, 4), (120,), (30, 4), (30,))
```

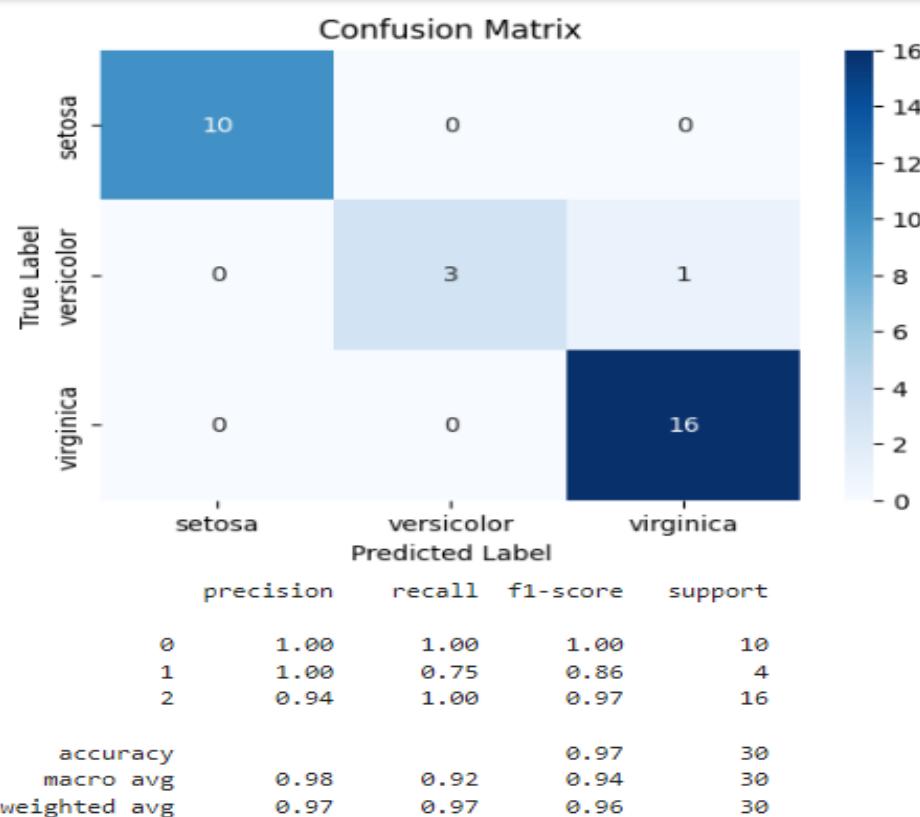
۲. مقیاس‌بندی داده: داده‌ها به وسیله‌ی 'Standard Scaler' مقیاس‌بندی می‌شوند، به این ترتیب که مقادیر آن‌ها به یک مقیاس استاندارد تبدیل می‌شود. این مرحله معمولاً برای بهبود عملکرد مدل مورد استفاده قرار می‌گیرد، زیرا که تفاوت مقادیر بزرگ در ویژگی‌های مختلف می‌تواند باعث اشتباهات در مدل شود.

```
[45] scaler = StandardScaler()
      X_train = scaler.fit_transform(X_train)
      X_test = scaler.transform(X_test)
```

۳. ایجاد مدل طبقه‌بندی: در این مرحله، یک مدل با استفاده از کرنل خطی برای داده‌های آموزشی ایجاد می‌شود. انتخاب کرنل مناسب می‌تواند به دقت و عملکرد کلی مدل کمک زیادی کند و در بسیاری از موارد، استفاده از کرنل خطی منجر به نتایج خوبی می‌شود.

```
[46] from sklearn.svm import SVC
      svm_linear = SVC(kernel='linear', random_state=24)
      svm_linear.fit(X_train, y_train)
      y_pred = svm_linear.predict(X_test)
```

۴. ارزیابی عملکرد مدل: در این مرحله، عملکرد مدل با استفاده از داده‌های تست ارزیابی می‌شود. نتایج طبقه‌بندی از طریق ماتریس درهم‌ریختگی و گزارش طبقه‌بندی، نشان داده می‌شود که مدل چقدر به درستی داده‌ها را در کلاس‌های مختلف تشخیص داده است و چه میزان از داده‌ها را به درستی طبقه‌بندی کرده است.



مدل طبقه‌بندی ارائه شده دارای **دقت کلی حدود ۹۷ درصد** است، که نشان می‌دهد عملکرد کلی مدل به طور معقول و به خوبی است. علاوه بر این، درصد دقیق تری از دقت برای هر کلاس نیز در نمودار نمایش داده شده است؛ به عنوان مثال، مدل توانسته حدود ۷۵ درصد از نمونه‌های کلاس ۱ را به درستی شناسایی کند و ۹۴ درصد از نمونه‌های کلاس ۲ را به درستی طبقه‌بندی کند. این نتایج نشان می‌دهند که طبقه‌بندی مدل به طور کلی عملکرد بسیار خوبی دارد. همچنین، ماتریس درهم‌ریختگی نیز در نمودار مشاهده می‌شود، که نشان می‌دهد مدل در اکثر نمونه‌ها به خوبی عمل کرده است.

پس از نتایج به دست آمده، به صورت کلی می‌توان نتیجه گرفت که مدل طبقه‌بندی به خوبی عمل کرده است، با این حال در موارد خاصی ممکن است به اشتباهاتی بخورد کند که می‌توان با بهبود روش‌ها و یا استفاده از تکنیک‌های مختلف اصلاح کرد. این فرایند به ما نشان می‌دهد که استفاده از روش‌های استاندارد و ارزیابی دقیق مدل می‌تواند به بهبود دقت و قابلیت پیش‌بینی آن کمک کند.

#### ➤ حال به بررسی چند ویژگی در مدل ایجاد شده می‌پردازیم :

: این ویژگی تعداد نمونه‌های پشتیبان (support vectors) را برای هر کلاس n\_support\_ نشان می‌دهد. در SVM، نمونه‌های پشتیبان نقاطی از داده هستند که در مرز تصمیم‌گیری قرار دارند یا به عبارت دیگر موجودیت هایی که باعث تصمیم‌گیری شبکه می‌شوند. تعداد این نقاط برای هر کلاس در svm\_linear.n\_support\_ ذخیره شده است.

```
[48] svm_linear.n_support_
array([ 2, 14, 11], dtype=int32)
```

: این ویژگی مجموعه‌ای از نمونه‌های پشتیبان را برای هر کلاس نشان می‌دهد. این نمونه‌ها دارای ویژگی‌های ویژه هستند که به کلاس جداگانه برگردانده می‌شوند.

```
svm_linear.support_vectors_
array([[ -0.86820189,   0.58628227,  -1.16195034,  -0.90483067],
       [-1.61772871,  -1.65428692,  -1.39777959,  -1.18539831],
       [ 0.25608833,   0.30994541,   0.48885437,   0.49800758],
       [ 1.13053628,  -0.08588849,   0.78364093,   0.77857522],
       [-0.49343849,  -0.08588849,   0.48885437,   0.49800758],
       [ 1.25545742,  -0.53400233,   0.66572631,   0.35772375],
       [ 1.38037855,   0.13816843,   0.72468362,   0.49800758],
       [-0.86820189,  -1.20617308,  -0.39550503,  -0.06312772],
       [ 0.38100946,  -0.30994541,   0.60676899,   0.35772375],
       [ 0.50593006,  -1.87834384,   0.48885437,   0.49800758],
       [ 0.88069401,  -0.53400233,   0.54781168,   0.49800758],
       [-0.24359621,  -0.08588849,   0.48885437,   0.49800758],
       [ 0.25608833,  -0.75805924,   0.84259824,   0.6382914 ],
       [ 0.63085174,   0.58628227,   0.60676899,   0.6382914 ],
       [ 0.13116719,   0.36222535,   0.66572631,   0.91885905],
       [ 0.63085174,  -1.65428692,   0.42989786,   0.21743993],
       [ 0.63085174,  -0.53400233,   0.84259824,   0.49800758],
       [ 0.38100946,  -0.98211616,   1.1373848 ,   0.35772375],
       [ 0.88069401,   0.36222535,   0.84259824,   1.1994267 ],
       [ 0.38100946,  -0.08588849,   0.72468362,   0.91885905],
       [-1.11804417,  -1.20617308,   0.48885437,   0.77857522],
       [ 0.25608833,  -0.08588849,   0.66572631,   0.91885905],
       [ 0.13116719,  -0.08588849,   0.84259824,   0.91885905],
       [ 0.50593006,  -0.53400233,   0.66572631,   0.91885905],
       [ 0.25608833,  -1.87834384,   0.78364093,   0.49800758],
       [ 0.63085174,  -0.75805924,   0.72468362,   0.91885905],
       [ 1.75514196,  -0.08588849,   1.25529942,   0.6382914 ]])
```

کد زیر به صورت جداگانه وزن‌ها و انحراف (bias) مدل SVM با استفاده از کرنل خطی را نمایش می‌دهند. آرایه‌ای از وزن‌های مربوط به هر یک از ویژگی‌ها را در فضای ورودی نمایش می‌دهد، که این وزن‌ها نشان‌دهنده تأثیر هر ویژگی در تصمیم‌گیری مدل SVM هستند. به عبارت دیگر، هر وزن نشان‌دهنده میزان اهمیت ویژگی مربوطه در تصمیم‌گیری مدل است. از سوی دیگر، تعیین کننده در فرایند تصمیم‌گیری مدل SVM استفاده می‌شود. انحراف نشان‌دهنده فاصله مدل از مرز تصمیمی است که توسط وزن‌ها و ترکیب خطی ویژگی‌ها تعیین می‌شود. این اطلاعات مفیدند تا درک بهتری از رفتار و تصمیم‌گیری مدل SVM با کرنل خطی پیدا کنیم و نحوه تأثیر ویژگی‌های مختلف در این فرآیند را درک کنیم.

```
print('weights ::',svm_linear.coef_[0], 'bias ::',svm_linear.intercept_[0])
weights : [-0.42097348  0.34031402 -0.81606519 -0.90831698] bias : -1.3352204521231705
```

وزن‌ها (weights) و انحراف (bias) که به صورت عددی در اختیار داریم، نمایانگر تأثیر هر ویژگی در مدل SVM با کرنل خطی هستند. این وزن‌ها برای هر ویژگی نشان می‌دهند که چقدر تغییر در آن ویژگی تأثیرگذار بر تصمیم‌گیری مدل است؛ به عبارت دیگر، ویژگی‌هایی که وزن مثبت دارند نشان‌دهنده این است که افزایش مقدار آن‌ها باعث افزایش احتمال تخصیص به کلاس مثبت می‌شود، در حالی که ویژگی‌هایی با وزن منفی، مانند  $-0.97348$ ،  $-0.42097348$ ، با افزایش مقدار، احتمال تخصیص به کلاس مثبت را کاهش می‌دهند. انحراف (bias) به عنوان یک مقدار ثابت، فاصله مدل از مرز تصمیم‌گیری را نشان می‌دهد؛ به عنوان مثال، اگر انحراف  $-0.51231705$  باشد، این نشان می‌دهد که مدل به طور میانگین از مرز تصمیم‌گیری به این فاصله دور است. در کل، این اعداد با استفاده از الگوریتم‌های یادگیری ماشین، به مدل SVM با کرنل خطی داده می‌شوند تا بتواند به درستی نمونه‌های جدید را به کلاس‌های مختلف تخصیص دهد.

## مرزهای تصمیم‌گیری

متد ۱ : برای رسم مرز تصمیم‌گیری از PCA استفاده می‌کنیم و این کار به دلایل زیر منطقی است: (که در بخش قبل هم به توضیح آن پرداخته شد).

۱. کاهش ابعاد داده: PCA به ما امکان می‌دهد تا ابعاد داده‌های پیچیده را به ابعاد کمتری تبدیل کنیم. این کاهش ابعاد باعث ساده‌تر شدن تحلیل داده و حفظ اطلاعات اصلی می‌شود. به عبارت دیگر، PCA ابعاد فضای ویژگی را به دسته‌های اساسی که بیشترین تغییرات را توضیح می‌دهند، کاهش می‌دهد.

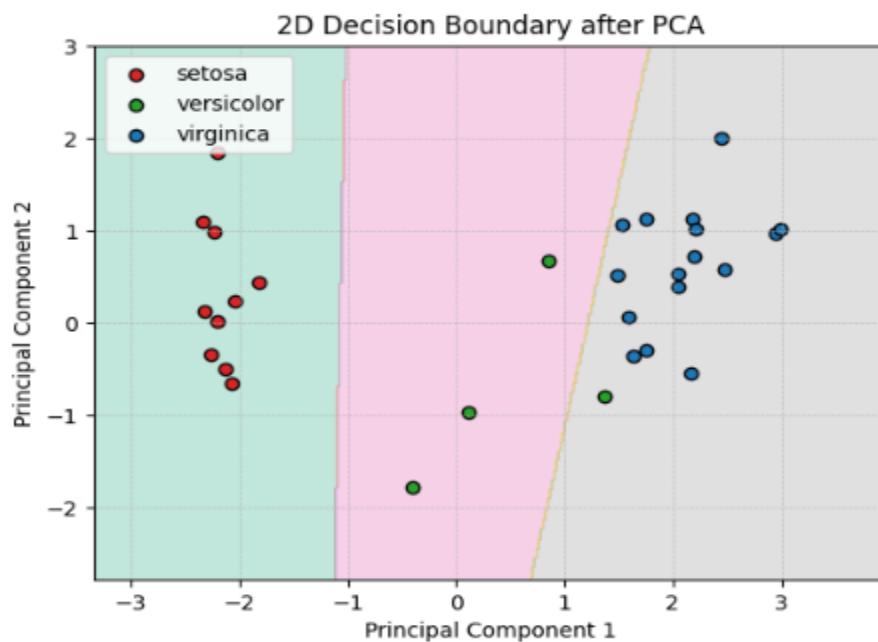
۲. تجزیه و تحلیل متغیرهای ویژگی : PCA با انتخاب مولفه‌های اصلی داده، به ما این امکان را می‌دهد که مهم‌ترین متغیرهای ویژگی که بیشترین واریانس را در داده‌ها توضیح می‌دهند را شناسایی کنیم. این ویژگی‌ها به عنوان مولفه‌های اصلی (Principal Components) نامیده می‌شوند و می‌توانند به عنوان محورهای جدید برای توصیف داده‌ها استفاده شوند.

۳. رسم مرز تصمیم‌گیری در فضای جدید: PCA با تبدیل داده‌ها به فضای مولفه‌های اصلی، این امکان را فراهم می‌کند که مرزهای تصمیم‌گیری را در فضای جدید (کم‌ابعاد) رسم کنیم. به عبارت دیگر، ما می‌توانیم مدل‌های یادگیری ماشین را بر روی این فضای کم‌ابعاد آموزش دهیم و مرزهای تصمیم‌گیری را در این فضا مشاهده کنیم.

۴. کاهش بار محاسباتی: با کاهش ابعاد داده، مدل‌های یادگیری ماشین از پیچیدگی کمتری برخوردار می‌شوند و عملکرد آن‌ها بهبود می‌یابد. این امر می‌تواند منجر به افزایش سرعت آموزش و پیش‌بینی مدل شود.

به طور کلی، PCA به عنوان یک ابزار قدرتمند برای کاهش ابعاد داده‌ها، ارائه یک تجزیه و تحلیل ساده‌تر و موثرتر از داده‌ها، و ارائه یک محور جدید برای رسم مرزهای تصمیم‌گیری، استفاده می‌شود که در بهبود کارایی و دقت مدل‌های یادگیری ماشین تأثیر مثبتی دارد.

ناحیه مرز تصمیم‌گیری مدل SVM با کرنل خطی و با کاهش بعد توسط PCA به صورت زیر است: (برای داده تست)



مشاهده می‌شود که ناحیه تصمیم‌گیری به صورت خطی جدا شده است.

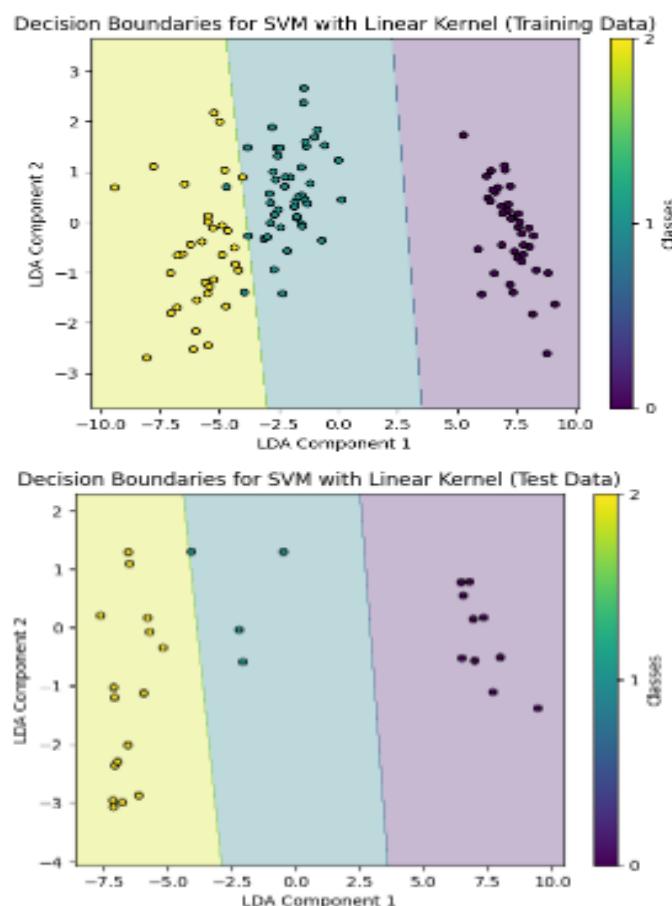
## متد ۲ : حال یک بار هم از LDA استفاده می کنیم :

دلیل اصلی استفاده از LDA این است که این روش برای مسائل نظارت شده (supervised) مناسب است، در حالی که تحلیل مؤلفه اصلی (PCA) برای مسائل بدون نظارت (unsupervised) به کار می رود.

یکی دیگر از دلایل انتخاب LDA این است که این روش تلاش می کند تا فاصله بین کلاس ها را بیشینه کرده و واریانس درون کلاس ها را کمینه کند. به عبارت دیگر، LDA به گونه ای عمل می کند که کلاس ها بهتر از یکدیگر جدا شوند. در مقابل، PCA به دنبال یافتن محوری است که بیشترین واریانس داده ها را فراهم کند، اما این محورها ممکن است لزوماً بهترین گزینه برای تمایز بین کلاس ها نباشند.

به طور خلاصه، LDA تمرکز خود را بر روی ویژگی هایی می گذارد که به بهترین نحو می توانند کلاس ها را از یکدیگر جدا کنند، در حالی که PCA بیشتر به پراکندگی کلی داده ها توجه دارد. این ویژگی LDA باعث می شود که برای مسائل طبقه بندی مناسب تر باشد.

در شکل زیر، نتایج برای داده های آموزش و تست رسم شده است که می بینیم عملکرد خیلی خوبی دارد.



## بخش سوم)

ج. بخش قبلی را با استفاده از هسته‌های چند جمله‌ای و با استفاده از کتابخانه scikit-learn از درجه یک تا ۱۰ پیاده سازی کنید و نتایج را با معیارهای مناسب گزارش کرده و مقایسه و تحلیل کنید. در نهایت، با استفاده از کتابخانه imageio جداسازی ویژگی‌های اصلی را (کاهش بعد از طریق یکی از روش‌های آموخته شده با ذکر دلیل) برای درجات ۱ تا ۱۰ در قالب یک GIF به تصویر بکشید و لینک دسترسی مستقیم به فایل GIF را درون گزارش خود قرار دهید.

در این بخش، هدف این است که یک سری مدل‌های طبقه‌بندی SVM با کرنل چندجمله‌ای (polynomial) با درجات مختلف از ۱ تا ۱۰ ایجاد کنیم و عملکرد هر مدل را ارزیابی کنیم. این کار با استفاده از یک حلقه for انجام می‌شود. در ابتدا، هر مدل SVM با درجه خاصی آموزش داده می‌شود و سپس دقیقت مدل و ماتریس درهم‌ریختگی (confusion matrix) برای هر درجه محاسبه و نمایش داده می‌شود.

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target

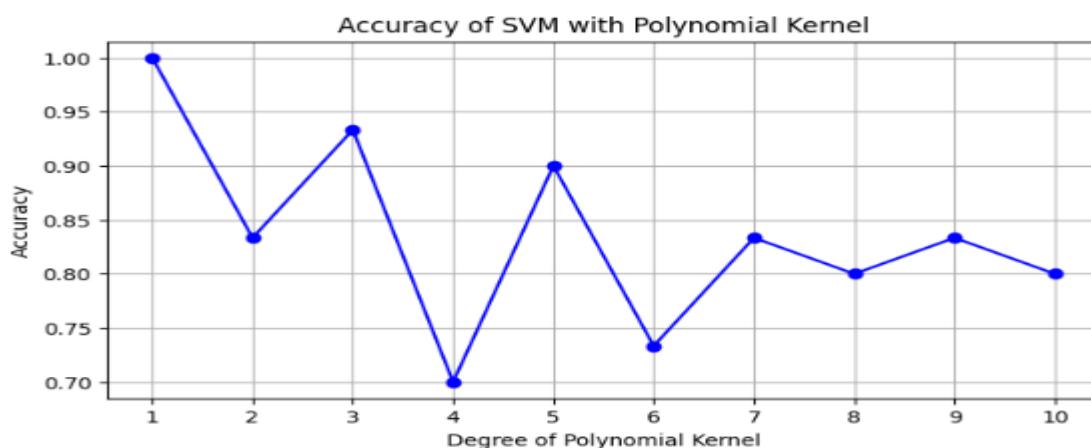
scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=24)

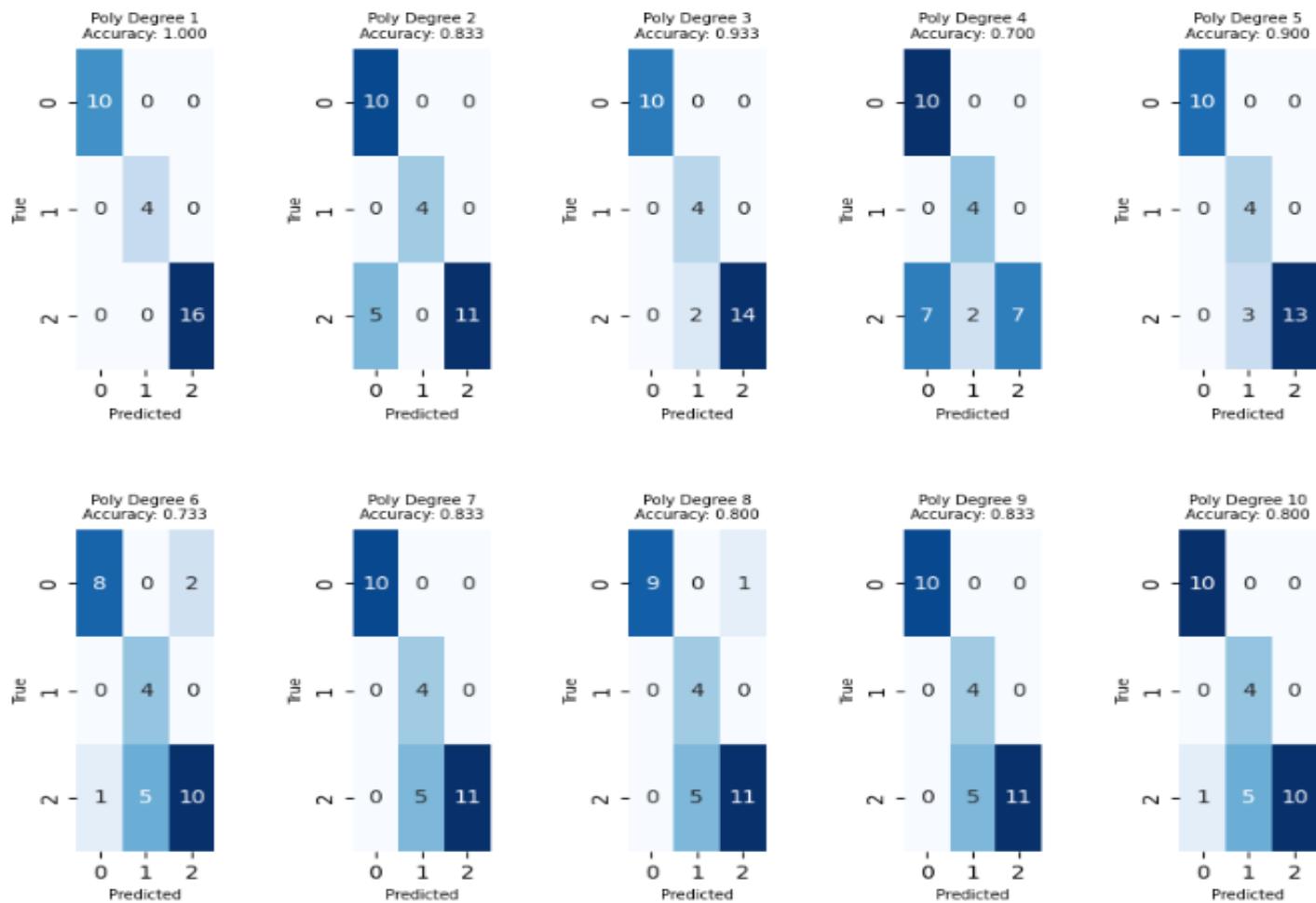
classifiers = [SVC(kernel='poly', degree=d, C=0.5) for d in range(1, 11)]
```

دقیقت مدل را با توجه به درجه کرنل در زیر می‌بینیم :

```
SVC with polynomial (degree 1) kernel: Accuracy = 1.000
SVC with polynomial (degree 2) kernel: Accuracy = 0.833
SVC with polynomial (degree 3) kernel: Accuracy = 0.933
SVC with polynomial (degree 4) kernel: Accuracy = 0.700
SVC with polynomial (degree 5) kernel: Accuracy = 0.900
SVC with polynomial (degree 6) kernel: Accuracy = 0.733
SVC with polynomial (degree 7) kernel: Accuracy = 0.833
SVC with polynomial (degree 8) kernel: Accuracy = 0.800
SVC with polynomial (degree 9) kernel: Accuracy = 0.833
SVC with polynomial (degree 10) kernel: Accuracy = 0.800
```

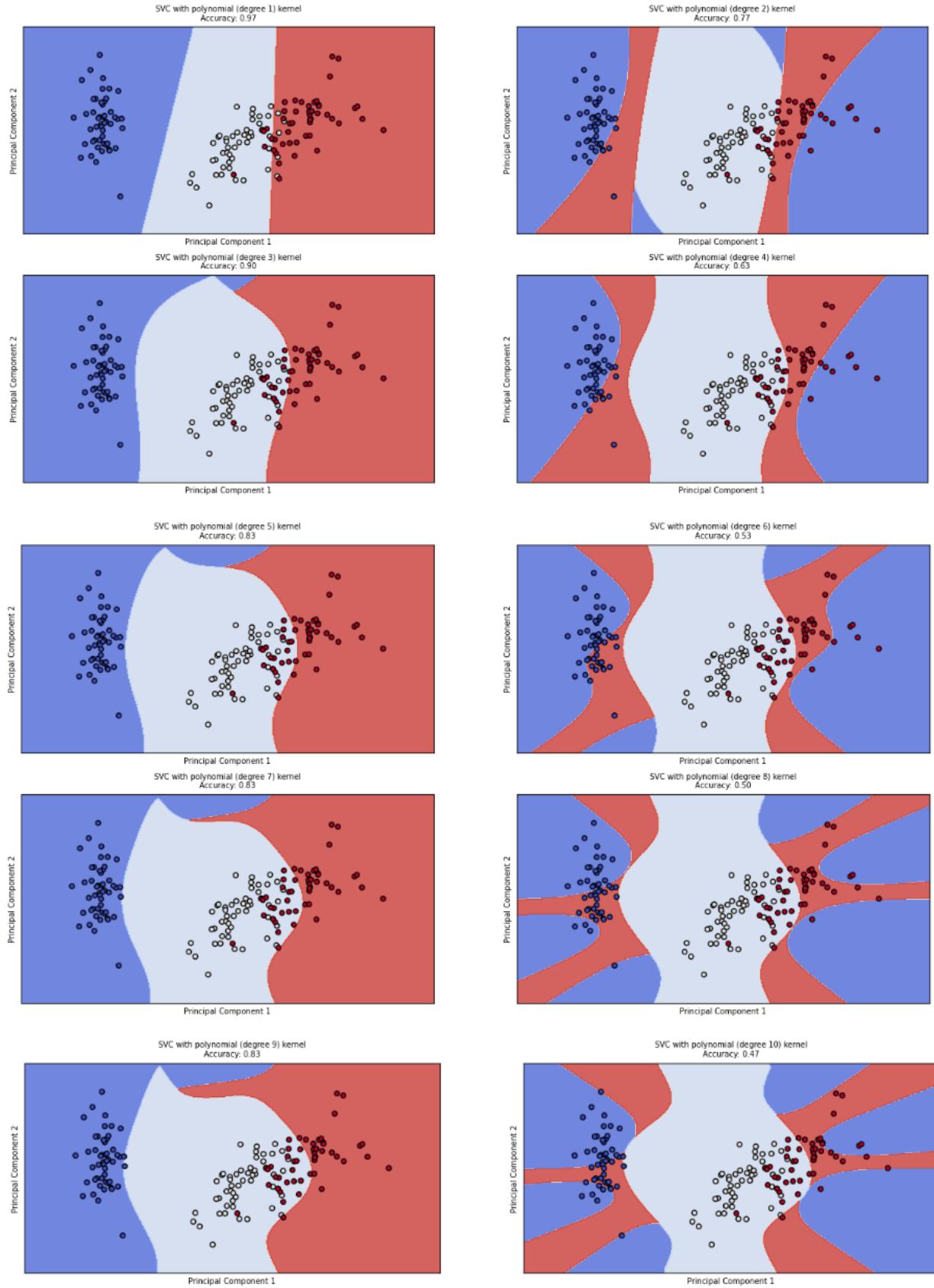


ماتریس های درهم ریختگی برای مدل با درجه کرنل از یک تا ۱۰ به صورت زیر است :



مشاهده می شود که با افزایش درجه کرنل، دقت طبقه بندی به صورت کلی کاهش می یابد. به طوری که در درجه ۱ (مدل خطی)، بهترین عملکرد را داریم با دقت ۱۰۰ درصد، اما با افزایش درجه، دقت کاهش می یابد و در درجه ۱۰ به ۸۰ درصد می رسد که نسبت به قبل خیلی افت کرده است.

برای نمایش عملکرد مدل طبقه بندی و ترسیم نواحی تصمیم گیری، از بصری سازی به کمک PCA و رسم کانتور استفاده می کنیم و در ادامه از LDA هم استفاده می کنیم. نتایج این بصری سازی در شکل زیر آمده است که به کمک مش گذاری و استفاده از PCA به دست آمده است.



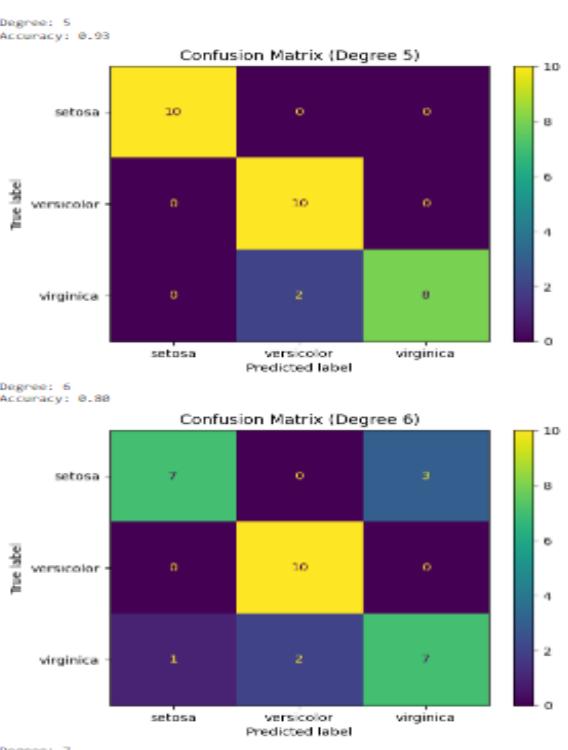
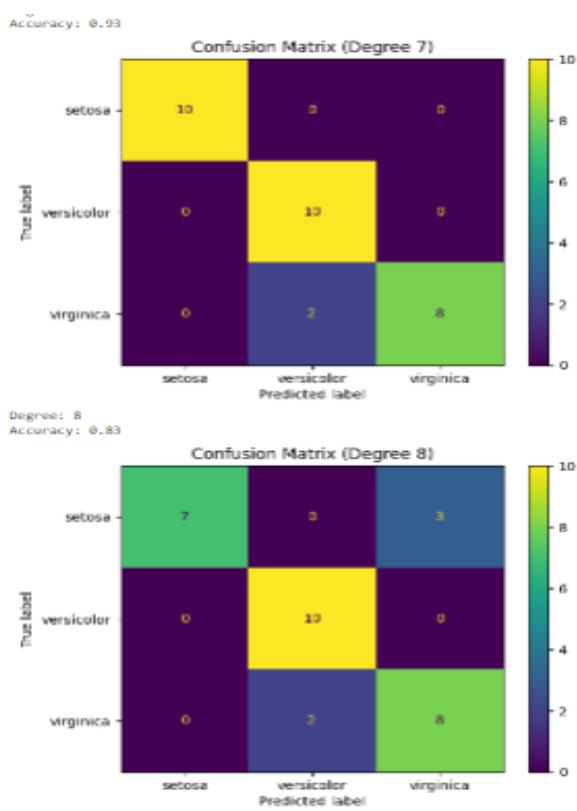
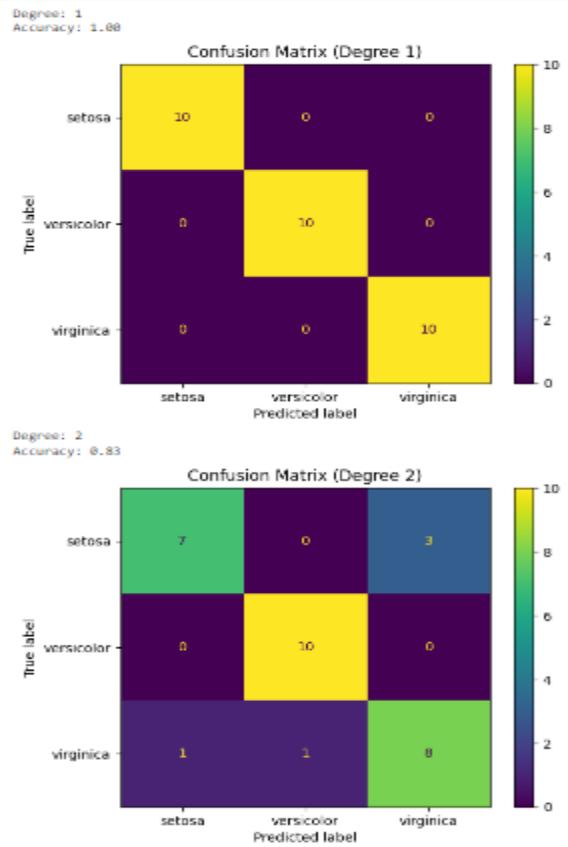
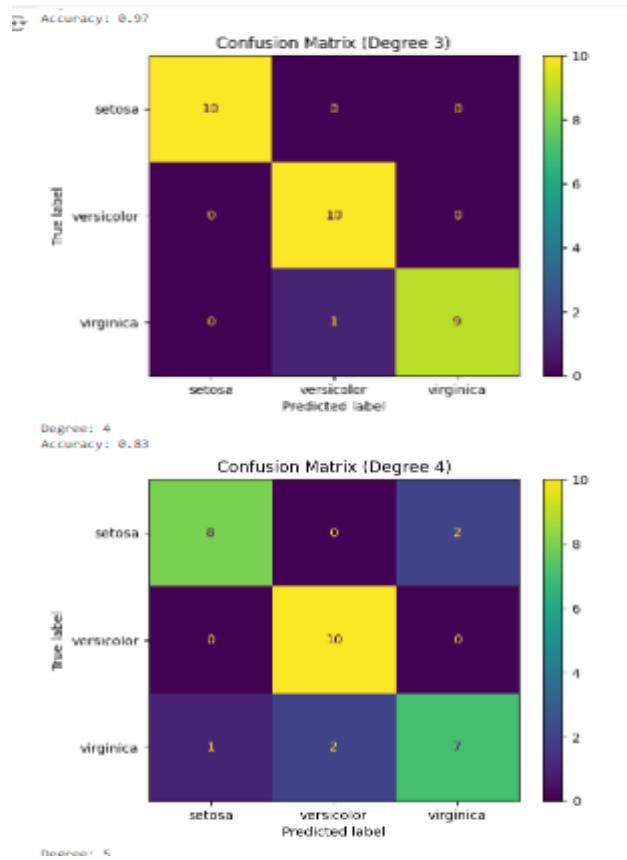
در شکل های بالا مشاهده می شود که بهترین عملکرد برای جدا کردن کلاس ها و طبقه بندی مربوط به مدل با درجه ۱ (degree = 1) است. همان طور که در این شکل پیداست، با افزایش درجه مدل، پیچیدگی آن نیز افزایش می یابد که منجر به بیش برازش (overfitting) می شود. این به این معناست که مدل نه تنها نویزها را یاد می گیرد، بلکه عملکرد طبقه بندی آن نیز ضعیف تر می شود.

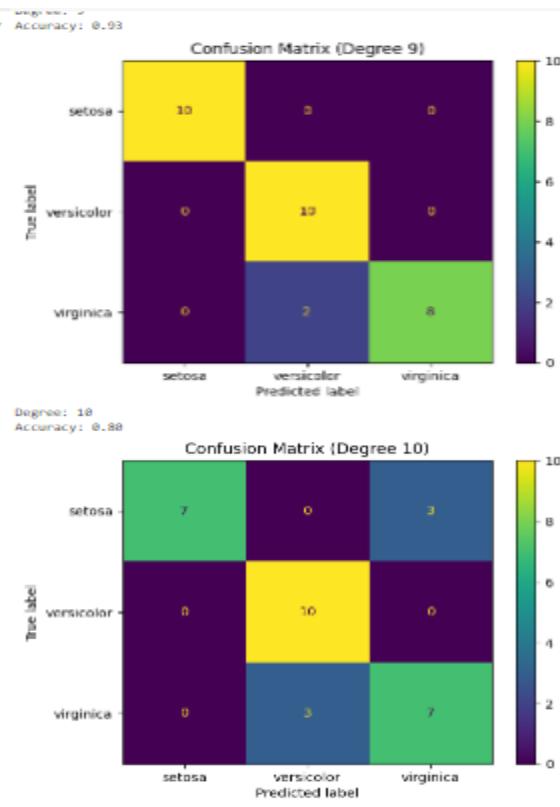
```
SVC with polynomial (degree 1) kernel: 0.97
SVC with polynomial (degree 2) kernel: 0.77
SVC with polynomial (degree 3) kernel: 0.90
SVC with polynomial (degree 4) kernel: 0.63
SVC with polynomial (degree 5) kernel: 0.83
SVC with polynomial (degree 6) kernel: 0.53
SVC with polynomial (degree 7) kernel: 0.83
SVC with polynomial (degree 8) kernel: 0.50
SVC with polynomial (degree 9) kernel: 0.83
SVC with polynomial (degree 10) kernel: 0.47
```

به طور کلی، از شکل مشخص است که با بالا رفتن درجه، انحنای نواحی تصمیم گیری بیشتر می شود و مدل بهتر می تواند پیچیدگی های مربوط به داده ها را یاد بگیرد. اما با توجه به نوع داده های ما در اینجا، بهترین نتیجه با طبقه بندی خطی که از مدل با درجه ۱ به دست می آید حاصل می شود. در نتیجه، این مدل دارای بهترین دقت است. بنابراین، این تحلیل نشان می دهد که استفاده از مدل های ساده تر مانند طبقه بندی خطی، زمانی که داده ها کم و ویژگی ها محدود هستند، به نتایج بهتری منجر می شود. این نتیجه گیری اهمیت انتخاب مدل مناسب با توجه به خصوصیات داده ها و هدف مسئله را برجسته می کند.

در این بخش، روش LDA با ۲ مؤلفه را به کار می گیریم. دلایل این انتخاب در بخش قبلی توضیح داده شده اند. مهم ترین دلیل این است که LDA برای بیشینه کردن فاصله بین کلاس ها طراحی شده است، در حالی که PCA عموماً برای داده هایی با ابعاد بالا استفاده می شود، در حالی که ابعاد داده های ما کم است.

ما مدل خود را با هسته چند جمله ای (polynomial kernel) از درجه ۱ تا ۱۰ آموزش دادیم و نتایجی که به دست آوردیم شامل دقت (accuracy) و ماتریس درهم ریختگی (confusion matrix) است.





همان‌طور که مشاهده می‌شود، بهترین نتیجه مربوط به مدل با درجه ۱ است. این امر به این دلیل است که با افزایش درجه مدل، مدل ما داده‌های آموزشی را به خوبی می‌آموزد و در نتیجه دچار بیش‌برازش (overfitting) می‌شود. با توجه به اینکه داده‌های ما کم هستند، نیازی به مدل پیچیده وجود ندارد. بدترین نتیجه متعلق به مدل با درجه ۱۰ و بهترین نتیجه متعلق به مدل با درجه ۱ (که همان مدل خطی است) می‌باشد. یکی از دلایل نوسانی بودن نتایج می‌تواند کم بودن تعداد داده‌ها باشد. همچنین، با توجه به اینکه ما فقط دو ویژگی داریم، استفاده از درجات بالای هسته چند جمله‌ای اشتباه است.

## تولید GIF

در ادامه برای تولید فایل GIF از تصاویری استفاده می‌شود که نمایش تصمیم‌گیری‌های یک مدل طبقه‌بندی SVC با هسته چند جمله‌ای را نمایش می‌دهند. در ابتدا، داده‌های آموزشی PCA تبدیل می‌شوند و سپس برای هر درجه از ۱ تا ۱۰، یک مدل SVC با هسته چند جمله‌ای با آن درجه آموزش داده می‌شود. سپس برای هر مدل، نقاط شبکه (mesh grid) برای دیدن ناحیه‌های تصمیم‌گیری تولید می‌شوند و بر روی آنها تصویر گرافیکی از

نتایج نمایش داده می شود. هر تصویر با نام مناسب ذخیره می شود و سپس این تصاویر به صورت یک فایل GIF با استفاده از کتابخانه imageio در درایو گوگل ذخیره می شوند. در نهایت، لینک به فایل GIF نمایش داده می شود که می توان از آن استفاده کرد.

لینک دسترسی به گیف در زیر آمده است، همچنین در فایل زیپ نهایی هم موجود است.

## لینک گیف

## بخش چهارم)

د. حال الگوریتم SVM را برای مورد قبلي، بدون استفاده از کتابخانه scikit-learn و به صورت From Scratch پیاده سازی کنيد. در اين بخش لازم است که يك کلاس SVM تعريف کنيد. اين کلاس می بايست حداقل داراي سه تابع (متده) Fit، Polynomial\_kernel و Predict باشد. متده Fit و Predict می بايست با دریافت درجه های ۱ تا ۱۰، هسته های چندجمله ای را محاسبه کند. دقت الگوریتم را با افزایش درجه گزارش کنید و نتایج حاصل را با بخش قبلي مقایسه کنید. در این قسمت نيز جداسازی ویژگی های اصلی را برای درجات ۱ تا ۱۰ در قالب يك GIF به تصویر بکشيد پيوند دسترسی مستقیم آن را در گزارش خود قرار دهيد.

در این بخش از کد حل تمرين که در اختيارمان قرار داده شده استفاده می کنيم و قسمتی از آن را گسترش می دهيم. اين کد شامل سه بخش اصلی است: الگوریتم SVM، تعمیم الگوریتم به صورت چند کلاسه و بصری سازی نتایج که در ادامه توضیح مختصری می دهیم :

### الگوریتم SVM

الگوریتم SVM یا ماشین بردار پشتیبان یکی از محبوب ترین الگوریتم های یادگیری ماشین است که برای دسته بندی داده ها استفاده می شود. این الگوریتم به دنبال یافتن یک مرز تصمیم گیری (هایپر پلین) است که داده ها را به بهترین شکل ممکن از هم جدا کند. در این کد، ما از یک کرنل خاص برای SVM استفاده می کنيم که باید به درستی تعریف شود. کرنل ها تابع های ریاضی هستند که داده ها را به یک فضای با بعد بالاتر می برنند تا جداسازی آن ها ساده تر شود.

تعمیم الگوریتم به صورت چند کلاسه

الگوریتم SVM به صورت پیش‌فرض برای دسته‌بندی دودویی (باینری) طراحی شده است، اما می‌توانیم آن را به صورت چند کلاسه (multi-class) نیز به کار بگیریم. یکی از روش‌های معمول برای این کار، روش One vs Rest است. در این روش، برای هر کلاس یک مدل SVM جداگانه آموزش داده می‌شود که آن کلاس را از سایر کلاس‌ها جدا می‌کند. به این ترتیب، اگر  $n$  کلاس داشته باشیم، به  $n$  مدل SVM نیاز خواهیم داشت.

### تعریف کرنل

در این کد، ما از کرنل چند جمله‌ای (poly) با ثابت ۱ و درجه ۳ استفاده کرده‌ایم. کرنل پلی‌نوم داده‌ها را به یک فضای با بعد بالاتر می‌برد که در آن، داده‌ها به راحتی قابل جداسازی هستند. این کرنل به صورت زیر تعریف می‌شود:

$$K(x, y) = (1 + x \cdot y)^3$$

### ورودی‌های کلاس SVM

در قسمت بعدی، به ورودی‌های کلاس SVM می‌پردازیم. این ورودی‌ها شامل داده‌ها، تارگت‌ها و نوع و پارامترهای کرنل هستند. علاوه بر این، کلاس SVM به ما پیش‌بینی‌های ۷ و همچنین ثابت‌ها و نوع کرنل را بازمی‌گرداند. این کلاس از کتابخانه CVX برای بهینه‌سازی استفاده می‌کند.

### بهینه‌سازی با CVX

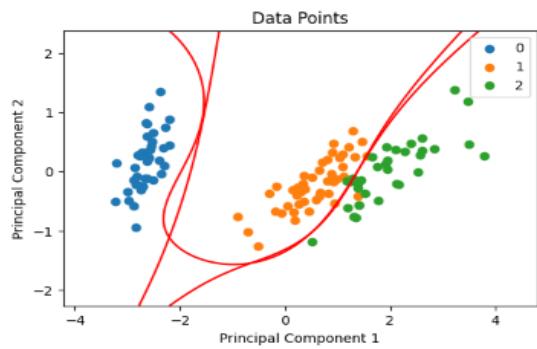
کتابخانه CVX یک ابزار بهینه‌سازی قدرتمند است که برای حل مسائل بهینه‌سازی محدب استفاده می‌شود. در این کد، از CVX برای پیدا کردن مقادیر بهینه پارامترهای SVM استفاده شده است. این کار تضمین می‌کند که مرز تصمیم‌گیری به بهترین شکل ممکن داده‌ها را جدا می‌کند.

### بصری‌سازی نتایج

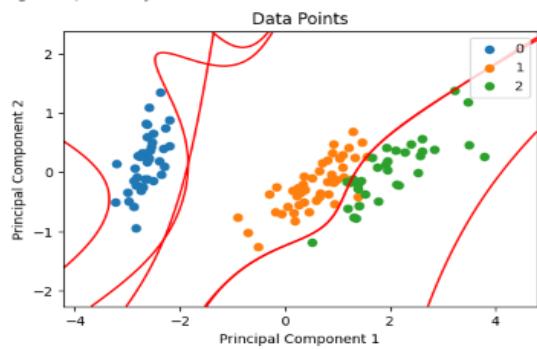
بخش آخر کد شامل بصری‌سازی نتایج است. ما به کمک سه کلاس مختلف، نواحی تصمیم و دقت را با یک حلقه for رسم می‌کنیم. این کار در شکل‌های زیر نشان داده شده است. این بصری‌سازی به ما کمک می‌کند تا بهتر متوجه شویم که چگونه الگوریتم SVM داده‌ها را دسته‌بندی کرده است.

در صفحه بعد به ازای کرنل چند جمله‌ای از درجه یک تا ده ناحیه‌های تصمیم و دقت را از روش SVM دستی رسم می‌کنیم:

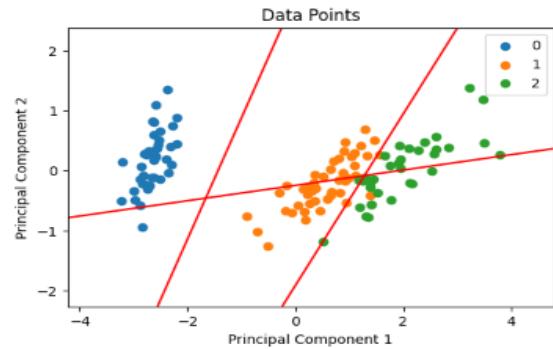
Training with polynomial degree 3  
Degree: 3, Accuracy: 0.9666666666666667



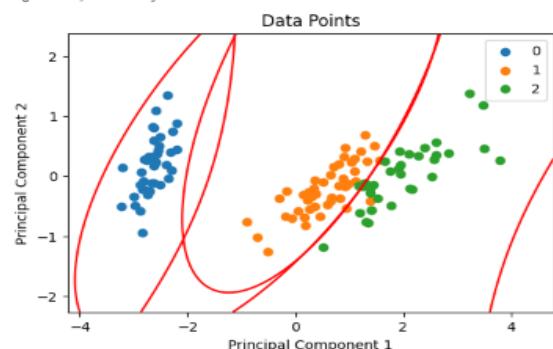
Training with polynomial degree 4  
Degree: 4, Accuracy: 0.9666666666666667



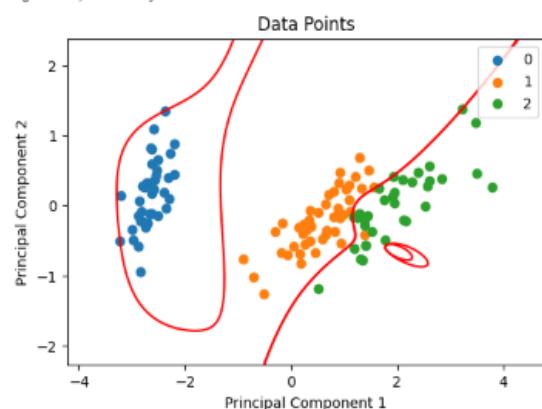
Training with polynomial degree 1  
Degree: 1, Accuracy: 0.9666666666666667



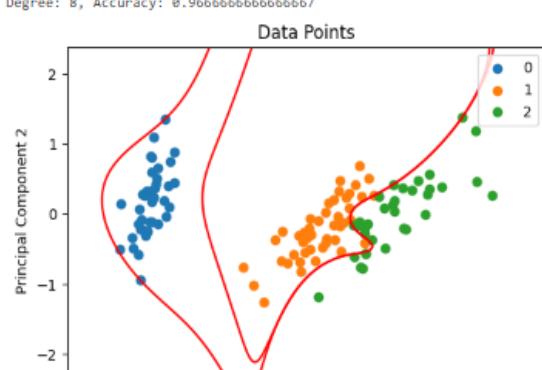
Training with polynomial degree 2  
Degree: 2, Accuracy: 0.9666666666666667



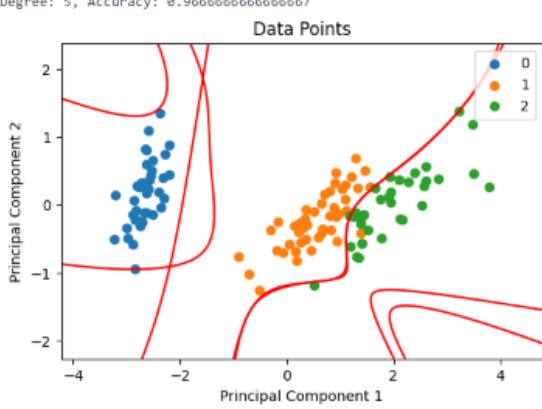
Training with polynomial degree 7  
Degree: 7, Accuracy: 0.9666666666666667



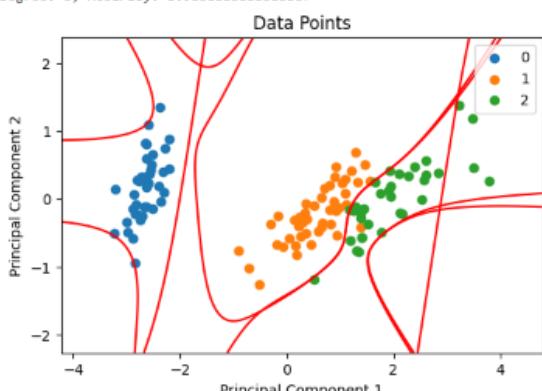
Training with polynomial degree 8  
Degree: 8, Accuracy: 0.9666666666666667

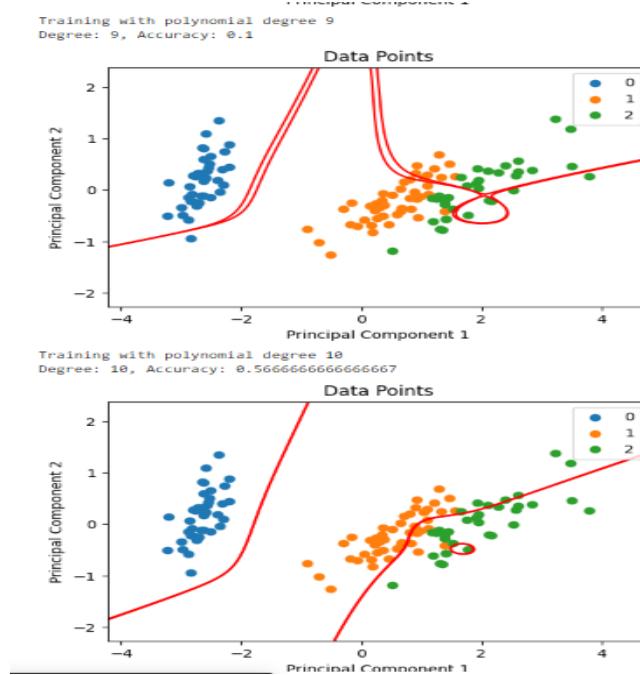


Training with polynomial degree 5  
Degree: 5, Accuracy: 0.9666666666666667

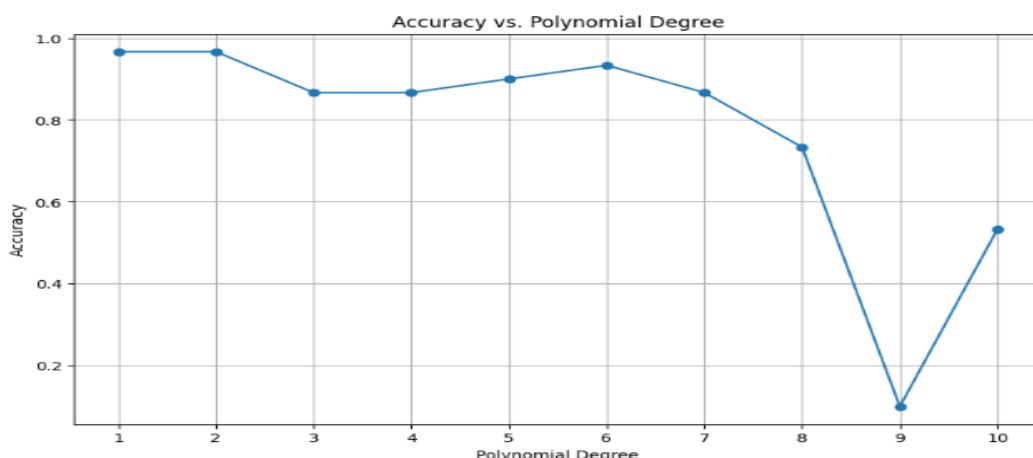


Training with polynomial degree 6  
Degree: 6, Accuracy: 0.9666666666666667





مشاهده می کنیم که نتایجی مشابه قسمت قبل دارد و با پیچیده شدن طبقه بند و با افزایش درجه، دقت افت داشته است که برای بررسی دقیق تر از نمودار زیر که دقت بر حسب درجه چندجمله ای است، استفاده می کنیم:



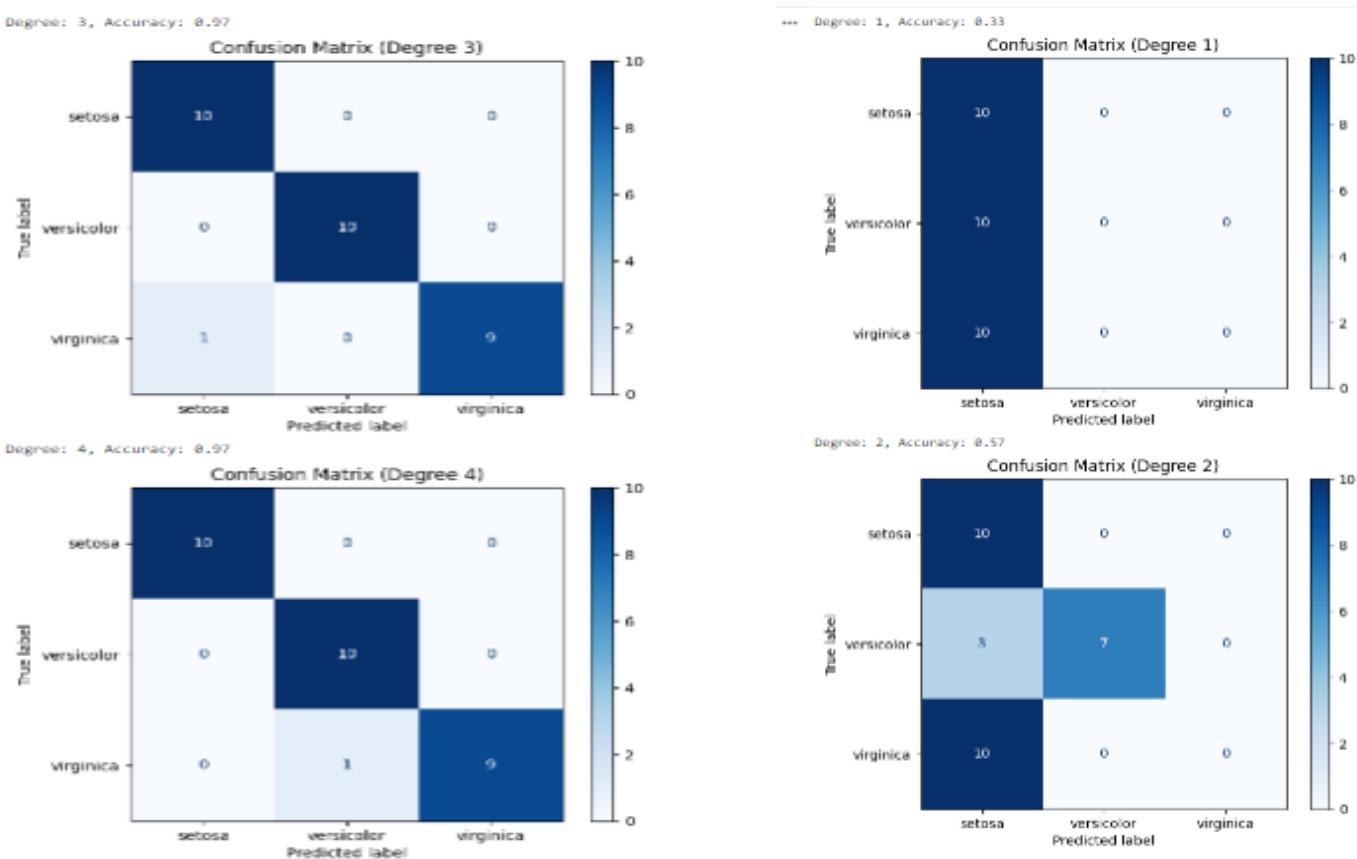
```
Training with polynomial degree 1
Degree: 1, Accuracy: 0.9666666666666667
Training with polynomial degree 2
Degree: 2, Accuracy: 0.9666666666666667
Training with polynomial degree 3
Degree: 3, Accuracy: 0.8666666666666667
Training with polynomial degree 4
Degree: 4, Accuracy: 0.8666666666666667
Training with polynomial degree 5
Degree: 5, Accuracy: 0.9
Training with polynomial degree 6
Degree: 6, Accuracy: 0.9333333333333333
Training with polynomial degree 7
Degree: 7, Accuracy: 0.8666666666666667
Training with polynomial degree 8
Degree: 8, Accuracy: 0.7333333333333333
Training with polynomial degree 9
Degree: 9, Accuracy: 0.1
Training with polynomial degree 10
Degree: 10, Accuracy: 0.5333333333333333
```

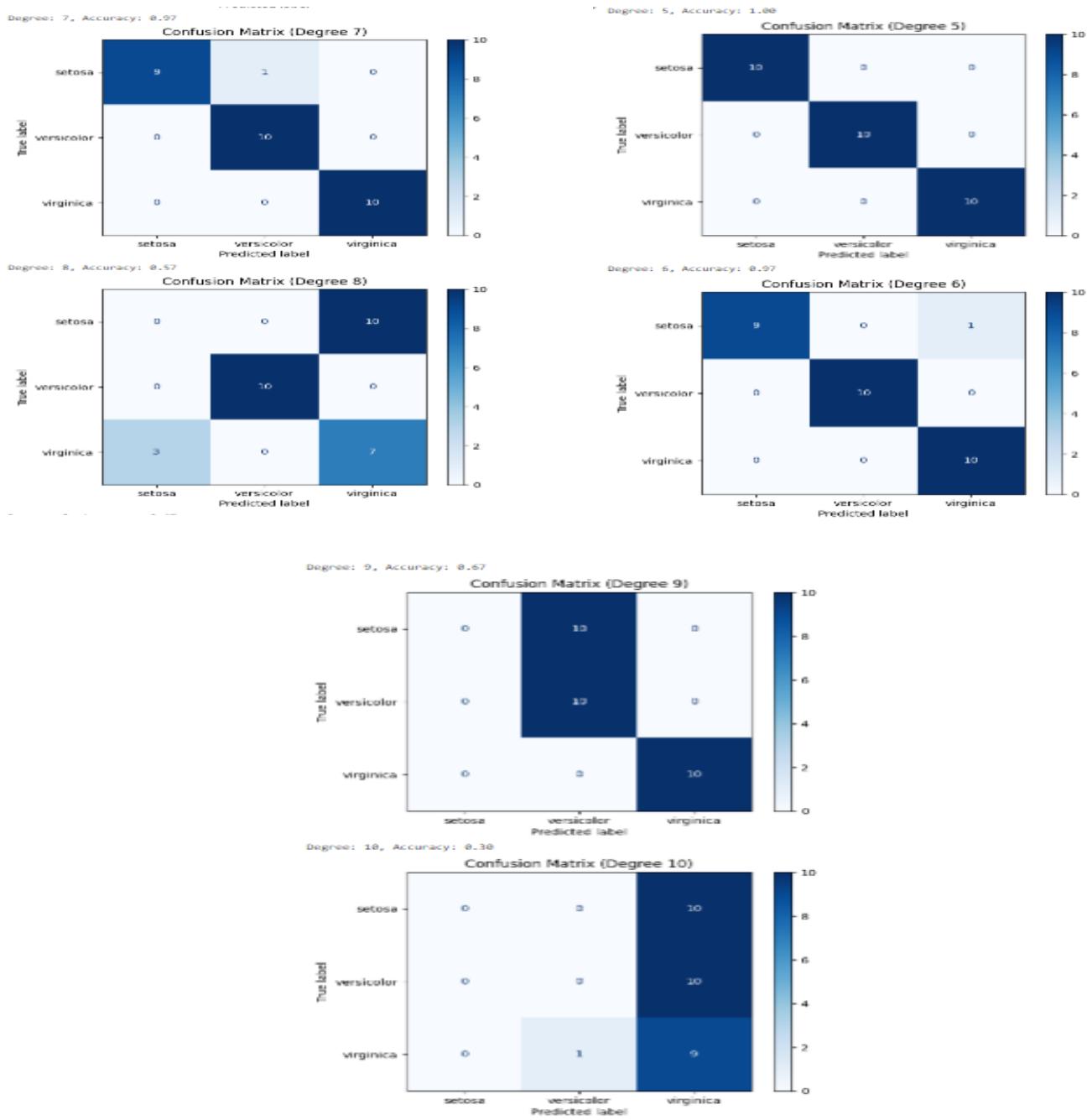
در شکل فوق نیز میزان دقت با توجه به هر درجه آورده شده است. مشاهده می‌شود که با پیچیده‌تر شدن طبقه‌بند و افزایش درجه، دقت کاهش محسوسی داشته است. این امر ممکن است به دلیل overfitting باشد، یعنی مدل بیش از حد به داده‌های آموزشی وابسته شده و عملکرد خوبی بر روی داده‌های جدید ندارد.

در گام آخر هم برای مشاهده گیف مانند قسمت قبل از لینک زیر استفاده می‌شود و همینطور در فایل زیپ گزارش هم موجود می‌باشد.

## لینک گیف

**(اضافی) :** حالا یک بار هم از LDA استفاده می‌کنیم و دقت را در این حالت هم برای کرنل چند جمله‌ای از درجه یک تا ده بررسی می‌کنیم:





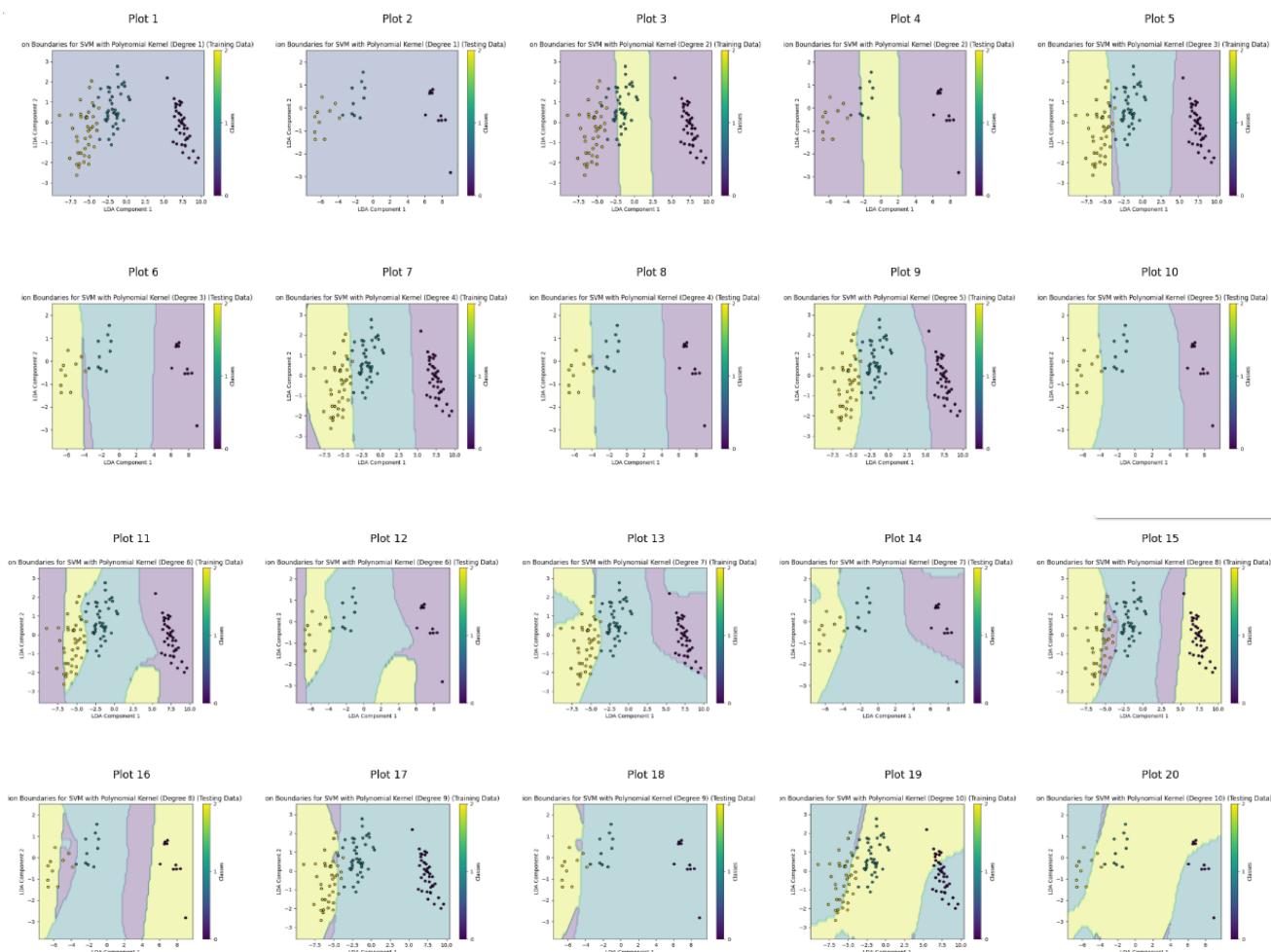
مدل دستی که پیاده‌سازی کردیم نسبت به مدلی که در کتابخانه scikit-learn استفاده کردیم (در بخش قبل) دقیق‌تری دارد. به عنوان مثال، در درجه ۱ مدل scikit-learn دقیق ۱۰۰ درصد داشته است در حالی که مدل دستی ما تنها دقیق در حدود ۳۳ درصد را داشته است. با افزایش درجه، دقیق مدل بهبود یافته و به درجه ۵ رسیده است، اما مشاهده می‌شود که دقیق نسبت به درجه‌های پایین‌تر به شدت کاهش یافته است که این نشان‌دهنده‌ی مشکل overfitting است. یکی از چالش‌های اصلی در پیاده‌سازی این مدل دستی، عدم توانایی

در حل مسئله Q.P بوده است برای ایجاد وزن های  $W$  برای حل این مشکل، تصمیم گرفتیم که ماتریس  $P$  را بزرگ تر کنیم تا بتوانیم درجات بالا را شناسایی کنیم و به درستی حل کنیم، پس داریم :

```
y = np.array([...]) # بردار مقادیر هدف
K = np.array([...]) # ماتریس هسته
n_samples = len(y)
P = cvxopt.matrix(np.outer(y, y) * K + 1e+5 * np.identity(n_samples))
```

در این کد، ماتریس  $P$  با افزودن یک مقدار بسیار بزرگ به مقادیر قطری آن (identity matrix) برای جلوگیری از تعريف شده است. این اقدام باعث می شود که مدل قادر به تشکیل وزن های مناسب برای درجات بالای مدل باشد و در نتیجه، دقت مدل بهبود یابد.

همچنین نمودار مرز تصمیم گیری برای این حالت یعنی الگوریتم SVM با کرنل چند جمله ای با درجه یک تا ده با روش LDA برای داده های تست و آموزش هم به صورت زیر هستند که نتیجه فوق را تایید می کنند :



## پرسش سه

مقاله [Credit Card Fraud Detection Using Autoencoder Neural Network](#) برای پیاده‌سازی این قسمت در نظر گرفته شده است. پس از مطالعه مقاله به سوالات زیر پاسخ دهید.

### بخش اول)

بزرگ ترین چالش ها در توسعه مدل های تشخیص تقلب چیست؟ این مقاله برای حل این چالش ها از چه روش هایی استفاده کرده است؟

به طور خلاصه این مقاله به بررسی مسئله طبقه‌بندی داده‌های نامتوازن در زمینه تشخیص کلاهبرداری با کارت اعتباری می‌پردازد. برای متوازن‌سازی نمونه‌ها بین کلاس‌های اکثریت و اقلیت، از الگوریتم بیش‌نمونه‌گیری استفاده می‌شود که می‌تواند نویز ایجاد کند. در این راستا، یک الگوریتم شبکه عصبی خودرمزنگار رفع نویز (DAE) پیشنهاد شده است که علاوه بر بیش‌نمونه‌گیری از طریق هزینه نادرست‌طبقه‌بندی، نویز را رفع کرده و مجموعه داده را طبقه‌بندی می‌کند. آزمایش‌ها نشان می‌دهند که این الگوریتم پیشنهادی دقیق‌تر طبقه‌بندی کلاس اقلیت در مجموعه داده‌های نامتوازن را بهبود می‌بخشد و در مقایسه با روش‌های سنتی عملکرد بهتری دارد.

در ادامه دقیق‌تر به بررسی چالش‌ها و راه حل‌ها می‌پردازیم.

✓ بزرگ‌ترین چالش‌ها در توسعه مدل‌های تشخیص تقلب به صورت زیر هستند:

۱. پروفایل رفتارهای تقلیبی پویا:

پروفایل رفتارهای تقلیبی دائمًا تغییر می‌کند و تراکنش‌های تقلبی تمایل دارند شبیه به تراکنش‌های قانونی به نظر برسند. این تغییرات پویا، تشخیص تراکنش‌های تقلبی را سخت می‌کند.

۲. عدم توازن داده‌ها:

در بیشتر موارد، داده‌های تراکنش‌های مالی به شدت نامتوازن هستند؛ به این معنا که تعداد تراکنش‌های قانونی بسیار بیشتر از تعداد تراکنش‌های تقلبی است. این عدم توازن باعث می‌شود که مدل‌های سنتی دقیق‌تر در تشخیص تراکنش‌های تقلبی داشته باشند. در واقع مجموعه داده‌های نامتوازن یک مشکل رایج در یادگیری ماشین است، زیرا اکثر مدل‌های طبقه‌بندی سنتی یادگیری ماشین نمی‌توانند با مجموعه داده‌های نامتوازن مقابله کنند. هزینه بالای نادرست‌طبقه‌بندی اغلب برای کلاس اقلیت رخ می‌دهد، زیرا مدل طبقه‌بندی سعی می‌کند تمام نمونه‌های داده را به کلاس اکثریت طبقه‌بندی کند.

#### ۳. انتخاب ویژگی‌های بهینه:

انتخاب متغیرها و ویژگی‌های مناسب برای مدل‌سازی یکی از چالش‌های بزرگ است. انتخاب نادرست ویژگی‌ها می‌تواند عملکرد مدل را به شدت کاهش دهد.

#### ۴. معیارهای ارزیابی مناسب:

انتخاب معیارهای مناسب برای ارزیابی عملکرد مدل‌ها در داده‌های نامتوازن یکی دیگر از چالش‌های است. معیارهای سنتی دقت (Accuracy) نمی‌توانند به خوبی عملکرد مدل را در چنین شرایطی نشان دهند.

پس در ارزیابی مدل‌های طبقه‌بندی، استفاده از معیار دقت (Accuracy) به تنها یکی کافی نیست، به ویژه زمانی که با مجموعه داده‌های نامتوازن مواجه هستیم. به عنوان مثال، فرض کنید یک مجموعه داده داریم که ۹۹,۹٪ آن شامل داده‌های غیرعادی است. اگر یک مدل طبقه‌بندی همه نمونه‌ها را به عنوان داده‌های عادی و تنها ۰,۱٪ آن شامل داده‌های غیرعادی دقت آن مدل ۹۹,۹٪ خواهد بود. این عدد ممکن است در نگاه اول بسیار خوب به نظر برسد، اما در واقعیت مدل نتوانسته هیچ‌یک از داده‌های غیرعادی را شناسایی کند و در شناسایی ناهنجاری‌ها کاملاً ناکارآمد است.

✓ حالا مقاله روش‌های پیشنهادی ای برای حل این چالش‌ها مطرح کرده است که به صورت زیر هستند :

#### ۱. استفاده از الگوریتم **Oversampling** :

برای رفع مشکل عدم توازن داده‌ها، این مقاله از روش **Oversampling** استفاده می‌کند تا تعداد نمونه‌های کلاس اقلیت (تراکنش‌های تقلیبی) افزایش یابد و اطلاعات اولیه بهتر حفظ شود. این روش به مدل کمک می‌کند تا دقت بهتری در تشخیص تراکنش‌های تقلیبی داشته باشد.

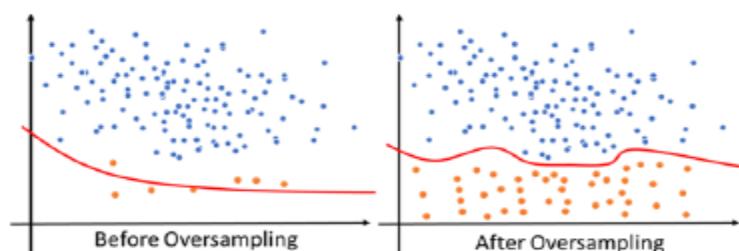


Fig. 3 Benefit of using oversampling

SMOTE (تکنیکبیش نمونه‌گیری اقلیت مصنوعی) یکی از تکنیک‌های پرکاربرد در حوزه یادگیری ماشین برای مقابله با مشکل مجموعه داده‌های نامتوازن است. این تکنیک بهویژه زمانی مفید است که داده‌های کلاسی کمتر دیده می‌شوند (کلاس اقلیت) به نسبت کلاس‌های دیگر بسیار کمتر هستند. در چنین شرایطی، مدل‌های یادگیری ماشین معمولاً به سمت کلاس اکثریت تمایل پیدا می‌کنند و داده‌های کلاس اقلیت را نادیده می‌گیرند. SMOTE با ایجاد نمونه‌های جدید از کلاس اقلیت به بهبود این وضعیت کمک می‌کند.

فرایند SMOTE به شرح زیر است:

الف. شناسایی -k‌نزدیک‌ترین همسایه‌ها:

ابتدا برای هر نمونه از کلاس اقلیت، نزدیک‌ترین همسایه‌های آن در فضای ویژگی‌ها شناسایی می‌شوند. تعداد این همسایه‌ها با پارامتر  $k$  تعیین می‌شود. همسایه‌های نزدیک یعنی نقاطی که بیشترین شباهت را به نمونه مورد نظر دارند.

ب. انتخاب تصادفی یک همسایه:

از میان  $k$ -نزدیک‌ترین همسایه‌های هر نمونه، به صورت تصادفی یک نقطه انتخاب می‌شود. این انتخاب تصادفی کمک می‌کند که نمونه‌های جدید متنوع‌تری ایجاد شوند.

ج. ایجاد نقطه داده جدید:

با استفاده از نمونه اصلی و نقطه همسایه انتخاب شده، یک نمونه جدید ایجاد می‌شود. این کار با استفاده از میانگین وزنی انجام می‌شود، به این صورت که مقداری از فاصله بین دو نقطه را بر اساس یک ضریب تصادفی محاسبه می‌کنند و این مقدار را به نقطه اصلی اضافه می‌کنند. به این ترتیب، نقطه داده جدید در میان دو نقطه اصلی و همسایه قرار می‌گیرد و ترکیبی از ویژگی‌های هر دو را دارد.

این فرایند تولید نمونه‌های مصنوعی باعث می‌شود تا توزیع کلاس اقلیت در مجموعه داده متوازن‌تر شود و مدل‌های یادگیری ماشین قادر به یادگیری بهتر و دقیق‌تر این کلاس‌ها باشند. در نتیجه، دقت کلی مدل بهبود یافته و احتمال نادیده گرفته شدن کلاس اقلیت کاهش می‌یابد.

SMOTE به دلیل سادگی و کارایی بالا، به یکی از تکنیک‌های استاندارد برای مقابله با مشکل داده‌های نامتوازن تبدیل شده است و در بسیاری از مسائل دنیای واقعی مانند تشخیص کلاهبرداری، پیش‌بینی بیماری‌ها و تحلیل ریسک مالی استفاده می‌شود.

<sup>1</sup> Synthetic Minority Oversampling Technique

## ۲. استفاده از شبکه‌های عصبی : Autoencoder

مقاله از شبکه‌های عصبی خودرمزگذار (Autoencoder) برای حذف نویز و دسته‌بندی داده‌ها استفاده می‌کند. هدف از این شبکه‌ها کاهش ابعاد داده‌ها و بازسازی آنهاست تا مدل بتواند الگوهای تقلیلی را بهتر تشخیص دهد.

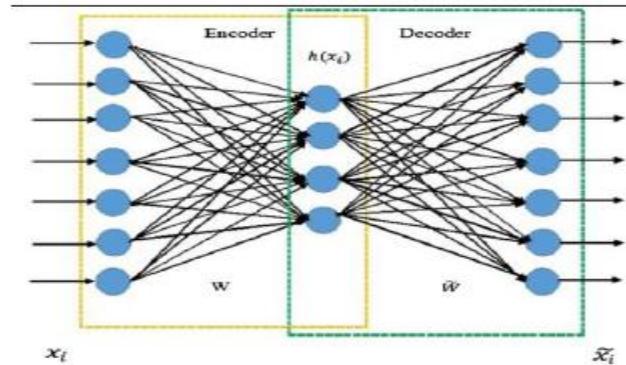


Fig. 1 architecture of autoencoder neural network

## ۳. استفاده از Denoising Autoencoder :

برای مقابله با مشکل نویز، مقاله از یک Denoising Autoencoder استفاده می‌کند که توانایی حذف نویز از داده‌های آموزشی را دارد. این روش به بهبود دقیق دسته‌بندی تراکنش‌های تقلیلی کمک می‌کند.

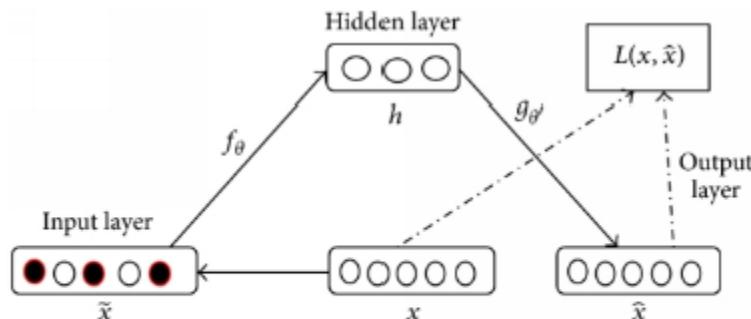


Fig. 2 Denoising autoencoder neural network

## ۴. استفاده از معیارهای ارزیابی مختلف:

برای ارزیابی عملکرد مدل‌ها، از معیارهایی مانند نرخ بازشناسی (Accuracy) و دقت (Recall Rate) استفاده شده است. این معیارها نشان می‌دهند که مدل پیشنهادی در تشخیص تراکنش‌های تقلیلی با دقت بالاتری نسبت به مدل‌های سنتی عمل می‌کند.

پس در شرایطی که با مجموعه داده‌های نامتوازن روبرو هستیم، نیاز به استفاده از معیارهای دیگری برای ارزیابی عملکرد مدل‌های طبقه‌بندی داریم. یکی از این معیارها نرخ یادآوری (Recall) یا نرخ تشخیص است. این معیار نشان می‌دهد که چه تعداد از ناهنجاری‌ها توسط مدل به درستی شناسایی شده‌اند، به این ترتیب به ما کمک می‌کند تا عملکرد واقعی مدل را در شناسایی ناهنجاری‌ها ارزیابی کنیم. برای این منظور، از ماتریس درهم‌ریختگی استفاده می‌شود که نتایج پیش‌بینی مدل را به چهار دسته تقسیم می‌کند: True Positive (TP) برای ناهنجاری‌هایی که به درستی شناسایی شده‌اند، True Negative (TN) برای داده‌های عادی که به درستی شناسایی شده‌اند، False Positive (FP) برای داده‌هایی که اشتباهًا به عنوان ناهنجاری شناسایی شده‌اند، و False Negative (FN) برای ناهنجاری‌هایی که به اشتباه به عنوان داده‌های عادی شناسایی شده‌اند. با استفاده از فرمول محاسبه نرخ یادآوری، می‌توانیم عملکرد مدل در شناسایی ناهنجاری‌ها را به طور دقیق‌تر ارزیابی کنیم. این معیار به ویژه در مواردی که ناهنجاری‌ها اهمیت بیشتری دارند، مانند تشخیص کلاهبرداری یا بیماری، بسیار مفید است. به این ترتیب، استفاده از معیارهای مناسب‌تر مانند نرخ یادآوری به ما کمک می‌کند تا به‌طور دقیق‌تر عملکرد مدل در مواجهه با داده‌های نامتوازن را ارزیابی کنیم و از نقاط ضعف مدل آگاه شویم.

در نتیجه با استفاده از این روش‌ها، مقاله موفق شده است تا مدل تشخیص تقلب را با دقت و بازشناسی بالاتری توسعه دهد و چالش‌های موجود در این زمینه را به خوبی مدیریت کند.

به طور کلی روال حل این چالش‌ها مطابق با شکل زیر هستند (که در بالاتر هم بیان شدند) :

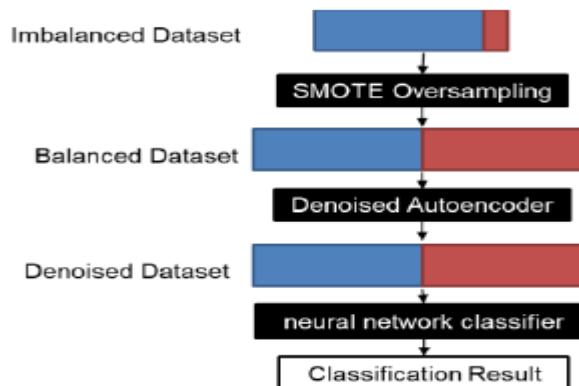


Fig. 5 Flowchart of the porcess

## بخش دوم)

در مورد معماری شبکه ارائه شده در مقاله توضیح دهید.

معماری شبکه ارائه شده در مقاله شامل دو بخش اصلی است:

### ۱. شبکه عصبی خود رمزگذار نویزگیری شده (Denoising Autoencoder) :

- این شبکه شامل ۷ لایه است که برای فرآیند نویزگیری طراحی شده است. بعد از اعمال نویز گوسی به داده‌های آموزشی، داده‌های نویزدار به این شبکه وارد می‌شوند و مدل خود رمزگذار آموزش می‌بیند تا قابلیت نویزگیری داده‌ها را در فرآیند پیش‌بینی به دست آورد.

- لایه‌ها شامل:

- لایه ورودی با داده‌های نویزدار

- چندین لایه کاملاً متصل با تعداد نورون‌های متغیر

- استفاده از تابع زیان مربع برای بهینه‌سازی

$$J_{DA,E} = \frac{1}{m} \sum_{i=1}^m \left( \frac{1}{2} \|\hat{x}_i - x_i\|^2 \right)$$

Table 2. Model design for denoised autoencoder

Dataset with noise (29)
Fully-Connected-Layer (22)
Fully-Connected-Layer (15)
Fully-Connected-Layer (10)
Fully-Connected-Layer (15)
Fully-Connected-Layer (22)
Fully-Connected-Layer (29)
Square Loss Function

### ۲. طبقه‌بند:

- این بخش شامل یک شبکه عصبی کاملاً متصل عمیق (Deep Fully Connected Neural Network) با ۶ لایه است. پس از نویزگیری داده‌های آموزشی، داده‌ها به این طبقه‌بند وارد می‌شوند.

لایه‌ها شامل:

- لایه ورودی با داده‌های نویزگیری شده

- چندین لایه کاملاً متصل با تعداد نورون‌های متغیر

- استفاده از تابع زیان انتروپی متقاطع SoftMax برای طبقه‌بندی نهایی

این معماری با ترکیب شبکه عصبی خود رمزگذار نویزگیری شده و الگوریتم نمونه‌برداری بیش از حد، توانسته است دقیق طبقه‌بندی را بهبود بخشد و مشکلات مرتبط با داده‌های نامتوازن را برطرف نماید.

Table 3. Model design for classifier

Denoised Dataset (29)
Fully-Connected-Layer (22)
Fully-Connected-Layer (15)
Fully-Connected-Layer (10)
Fully-Connected-Layer (5)
Fully-Connected-Layer (2)
SoftMax Cross Entropy Loss Function

## بخش سوم)

ج. مدل ارائه شده را پیاده‌سازی کرده و با استفاده از این دیتاست آموزش دهید. برای جلوگیری از بیش‌برازش، آموزش مدل را طوری تنظیم کنید که در انتهای آموزش، بهترین وزن‌های مدل بر اساس خطای قسمت اعتبارسنجی بازگردانده شود.

مقاله به صورت زیر عمل کرده است :

در ابتدا، از مجموعه داده‌ای که شامل ۲۸۳۱۵ تراکنش کارت اعتباری است و ۵٪ از آن‌ها به عنوان کلاهبرداری دسته‌بندی شده‌اند، استفاده کرده است. سپس با استفاده از بیش‌نمونه‌گیری، مجموعه داده را به یک مجموعه داده متعادل تبدیل کرده است. بعد از آن، از اتوانکوادر دنویز شده برای دریافت مجموعه داده دنویز شده استفاده شده است. در نهایت، از یک مدل شبکه عصبی کاملاً متصل و عمیق برای طبقه‌بندی نهایی استفاده می‌کند. در فرآیند پیش‌پردازش داده، داده "زمان" حذف و بخش "مقدار" نرمال‌سازی می‌شود. برای بیش‌نمونه‌گیری، تنها بر روی مجموعه داده آموزش این عمل انجام می‌شود و سپس مجموعه داده آموزش به یک مجموعه داده شامل تعداد

مساوی نمونه‌های عادی و ناهنجار تبدیل می‌شود. سپس با استفاده از یک اتوانکودر دنویز کننده، مجموعه داده را دنویز می‌کنند تا بتوانند از آن در فرآیند طبقه‌بندی استفاده کنند. در پایان، از یک مدل طبقه‌بندی با استفاده از شبکه عصبی کاملاً متصل و عمیق برای طبقه‌بندی نهایی استفاده می‌کنند.

در بخش ارزیابی و نتایج، ابتدا جزئیات اجرا بررسی شده و سپس نتایج ارزیابی مدل با و بدون بیش‌نمونه‌گیری مقایسه شده است. برای نرمال‌سازی مجموعه داده از توابع "sklearn" استفاده شده و برای بیش‌نمونه‌گیری از تابع "SMOTE" استفاده شده است. علاوه بر این، مدل اتوانکودر دنویز شده و طبقه‌بند شبکه عصبی کاملاً متصل با استفاده از "TensorFlow" پیاده‌سازی شده‌اند.

روال کار ما نیز به همین صورت باید باشد. پس داریم:

در ابتدا از kaggle ڈیتابست را دانلود می‌کنیم.

این مجموعه داده شامل تراکنش‌هایی است که توسط کارت‌های اعتباری در ماه سپتامبر ۲۰۱۳ توسط دارندگان کارت اروپایی انجام شده است. این مجموعه داده شامل تراکنش‌هایی است که در دو روز اتفاق افتاده است، ما ۴۹۲ تقلب را از بین ۲۸۴۸۰۷ تراکنش داریم. این مجموعه داده بسیار نامتوازن است، کلاس مثبت (تقلب‌ها) حدود ۱۷۲٪ از کل تراکنش‌ها را تشکیل می‌دهد. این شامل فقط متغیرهای ورودی عددی است که نتیجه تبدیل PCA است. ویژگی‌های V1, V2, ..., V28 کامپوننت‌های اصلی است که با PCA به دست آمدند، تنها ویژگی‌هایی که با PCA تبدیل نشده‌اند، "زمان" و "مقدار" هستند. ویژگی 'زمان' حاوی ثانیه‌های گذشته شده بین هر تراکنش و اولین تراکنش در مجموعه داده است. ویژگی 'مقدار' مبلغ تراکنش است، این ویژگی می‌تواند برای یادگیری وابسته به مثال با حساسیت به هزینه استفاده شود. ویژگی 'کلاس' متغیر پاسخ است و مقدار ۱ را در صورت وقوع تقلب و در غیر این صورت صفر می‌گیرد. (۲۸۴۸۰۷ ردیف و ۳۱ ستون)

<sup>2</sup> <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud?resource=download>

دیتاست به فرم کلی زیر است :

```
[2] import pandas as pd  
df = pd.read_csv('/content/drive/MyDrive/MachineLearning/HW3/creditcard.csv')  
df
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.076803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79	0
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00	0
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0

284807 rows x 31 columns

هیستوگرام داده ها به صورت زیر است :

```
Text(0, 0.5, 'Frequency')
```



میبینیم که داده ها در دو کلاس بسیار آنبالانس هستند که به صورت دیگر هم می توان نمایش داد :

```

[11] frauds = df[df.Class == 1]
      normal = df[df.Class == 0]

[12] frauds.shape
→ (492, 31)

[13] normal.shape
→ (284315, 31)

```

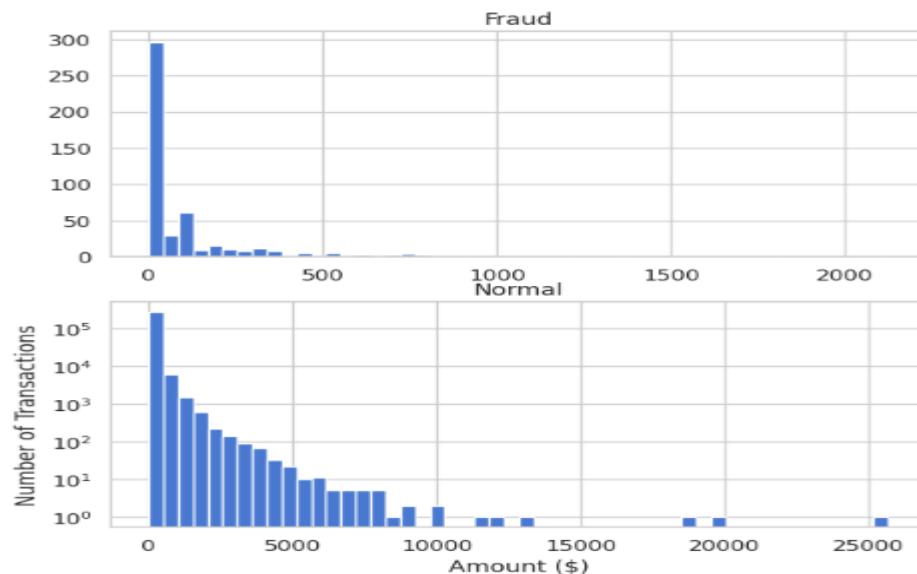
که آنبالانس بودن داده ها مشخص است.

را چک می کنیم و میبینیم که missing values Missing values نداریم.

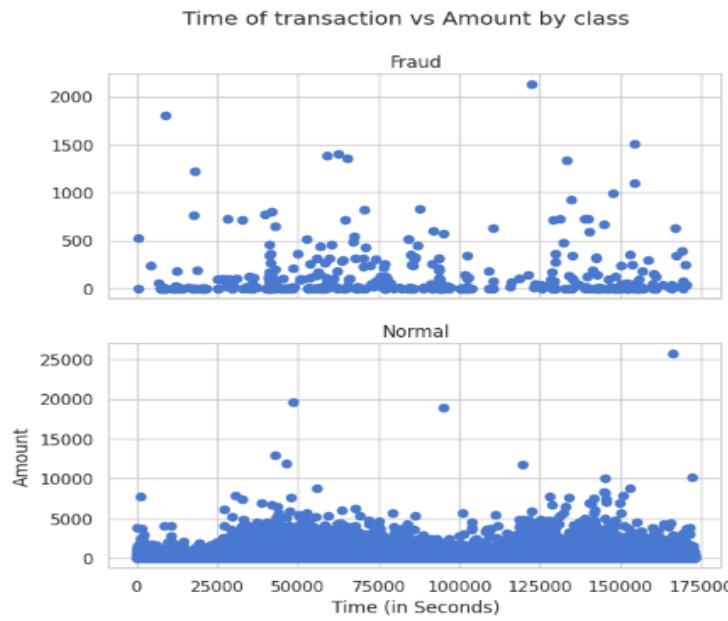
```
[ ] df.isnull().sum().sum()
→ 0
```

No missing value.

حالا بررسی می کنیم که میزان پول استفاده شده در کلاس های مختلف تراکنش چقدر متفاوت است در واقع نمودار مبلغ به ازای هر تراکنش بر اساس کلاس را به صورت زیر رسم می کنیم.

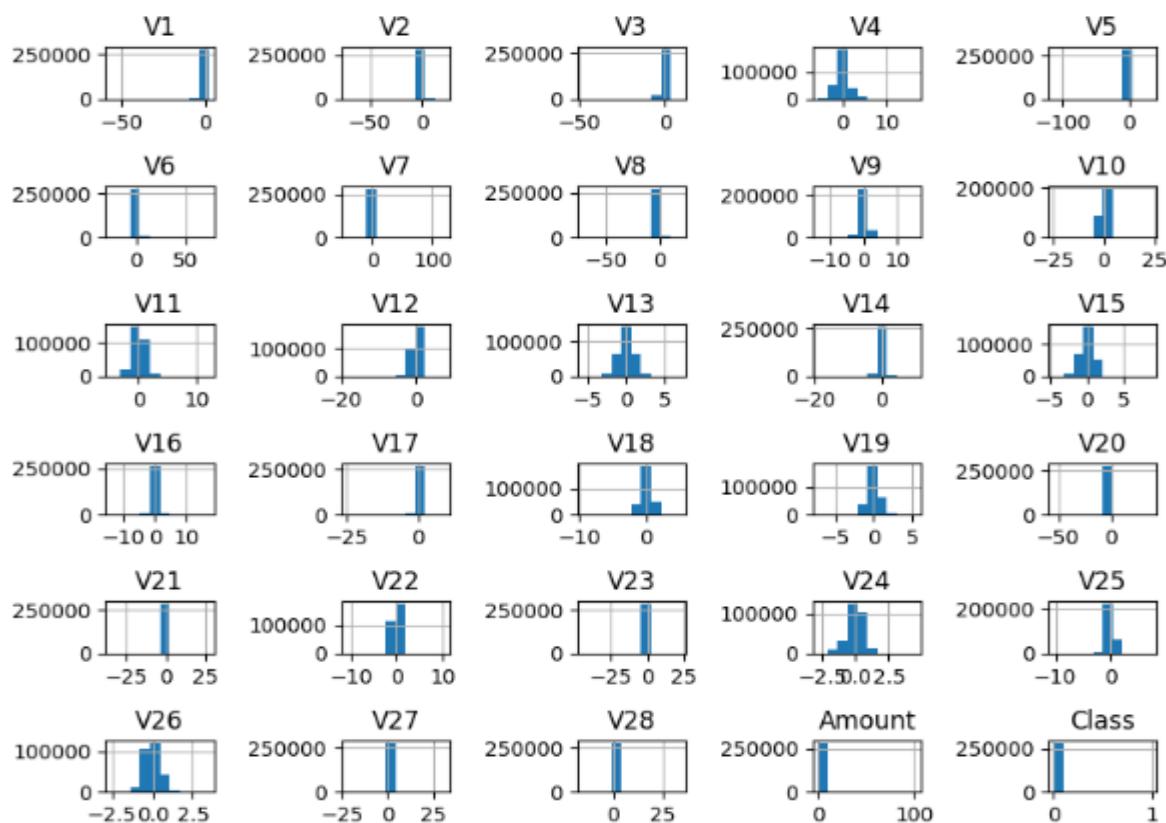


همینطور بررسی می کنیم که آیا معاملات متقلبانه در زمان معینی بیشتر اتفاق می افتد؟



با توجه به نمودار به نظر نمی رسد که زمان معامله واقعاً مهم باشد.

برای بررسی دقیق‌تر، نمودار‌های هیستوگرام‌های ستون‌های عددی DataFrame را به صورت زیر مشاهده می‌کنیم:



حالا برای پیاده سازی مقاله به صورت زیر عمل می کنیم:

در ابتدا ما دیتاست را به سه دسته آموزش، اعتبارسنجی و ارزیابی تقسیم می کنیم. همان‌طور که در شکل زیر نشان داده شده است، ۶۰ درصد از داده‌ها به دسته آموزش اختصاص یافته‌اند و ۴۰ درصد باقی‌مانده به طور مساوی بین اعتبارسنجی و ارزیابی تقسیم شده‌اند.

```
[ ] X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=24)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.25, random_state=24) # 0.25 * 0.8 = 0.2

❷ # calculate percentages
train_percent = len(X_train) / len(X) * 100
val_percent = len(X_val) / len(X) * 100
test_percent = len(X_test) / len(X) * 100

# Display the shapes and percentages
shapes = {
    "X_train_shape": X_train.shape,
    "X_val_shape": X_val.shape,
    "X_test_shape": X_test.shape,
    "y_train_shape": y_train.shape,
    "y_val_shape": y_val.shape,
    "y_test_shape": y_test.shape
}

percentages = {
    "train_percent": train_percent,
    "val_percent": val_percent,
    "test_percent": test_percent
}
shapes, percentages
```

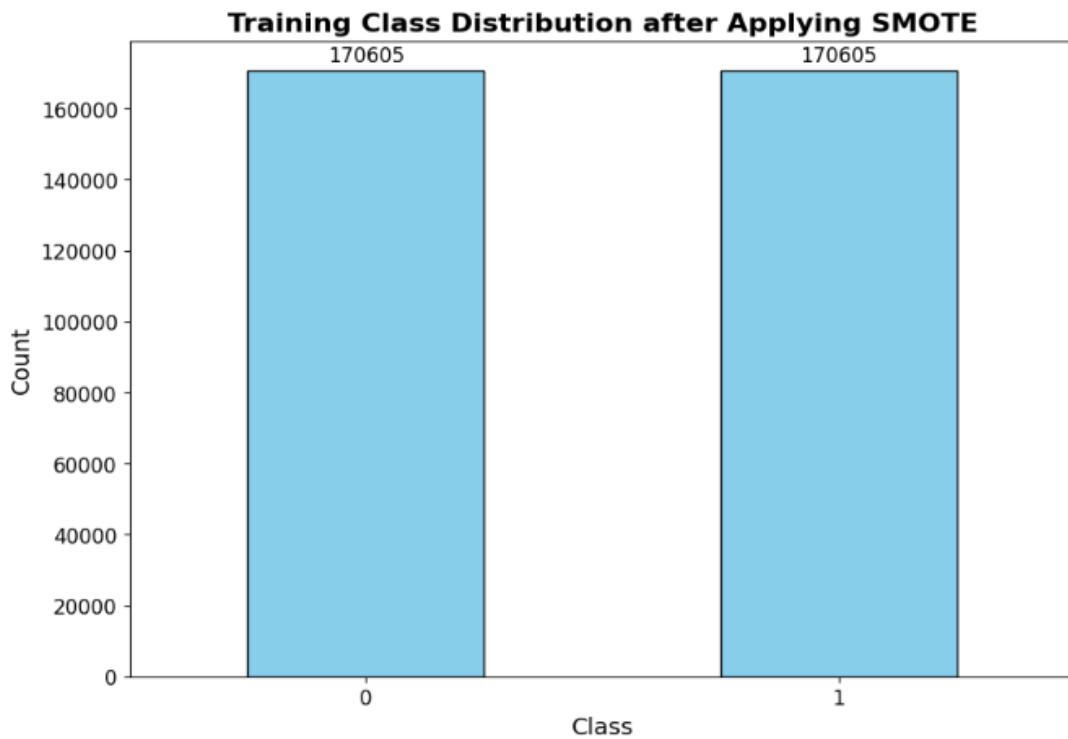
⤵ ({'X\_train\_shape': (170883, 29),  
 'X\_val\_shape': (56962, 29),  
 'X\_test\_shape': (56962, 29),  
 'y\_train\_shape': (170883,),  
 'y\_val\_shape': (56962,),  
 'y\_test\_shape': (56962,),  
 'train\_percent': 59.99957866204131,  
 'val\_percent': 20.000210668979342,  
 'test\_percent': 20.000210668979342})

برای بهبود عملکرد مدل و مقابله با عدم تعادل داده‌ها، از الگوریتم SMOTE استفاده می‌کنیم. این الگوریتم با استفاده از مازول آمده‌ای که در کتابخانه‌های مرتبط موجود است، پیاده‌سازی شده است. SMOTE با ایجاد نمونه‌های مصنوعی در اطراف نقاط کلاس اقلیت، تعادل کلاس‌ها را برقرار می‌کند. این کار با استفاده از تعداد مشخصی از نزدیک‌ترین همسایه‌ها انجام می‌شود که این تعداد می‌تواند به عنوان یک هایپرپارامتر تنظیم شود. همچنین، هایپرپارامتر sampling strategy که می‌تواند یک عدد یا استرینگ باشد، در اینجا به معنای oversampling کلاس اقلیت تا زمانی که تعداد نمونه‌ها در هر دو کلاس برابر شود، تنظیم شده است. بر اساس مقاله، این الگوریتم تنها روی داده‌های آموزش اعمال می‌شود. (به صورت زیر)

**apply SMOTE exclusively on the training set**

```
[83] from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy='minority', random_state =24)
X_train, y_train = smote.fit_resample(X_train, y_train)
```

نتایج این مرحله در شکل زیر نمایش داده شده‌اند که نشان می‌دهد داده‌ها به طور کامل متعادل شده‌اند.



در مرحله بعد، برچسب‌ها را به صورت One Hot Encode تبدیل می‌کنیم. این مرحله در شکل زیر نوشته شده است.

```
| from tensorflow.keras.utils import to_categorical  
| y_train = to_categorical(y_train, num_classes=2)  
| y_val = to_categorical(y_val, num_classes=2)  
| y_test = to_categorical(y_test, num_classes=2)
```

سپس به صورت زیر ، به داده‌ها نویز گوسی اضافه می‌کنیم تا بتوانیم شبکه اتوانکودر را آموزش دهیم. ما یک نویز با شدت ۲۰ درصد به داده‌ها اضافه می‌کنیم که این شدت به عنوان یک هایپرپارامتر قابل تنظیم است و می‌تواند کم و زیاد شود.

#### Introduce noise into the data

```
[86] noise_factor = 0.2  
| X_train_noisy = X_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train.shape)  
| X_val_noisy = X_val + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_val.shape)  
| X_test_noisy = X_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_test.shape)
```

سپس به صورت کد زیر، یک اتوانکودر مطابق با مشخصات ذکر شده در مقاله می‌سازیم و با استفاده از بهترین وزن‌ها که مطابق checkpoint ذخیره شده‌اند، آن را آموزش می‌دهیم. این شبکه دارای تابع فعال‌ساز ReLU در MSE و معیار خطای لایه‌های پنهان و sigmoid در لایه خروجی است. برای بهینه‌سازی از الگوریتم Adam استفاده شده است.

define the Autoencoder for denoising

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.callbacks import ModelCheckpoint

# Build the denoising autoencoder
input_dim = X_train.shape[1]

input_layer = Input(shape=(input_dim,))
encoded = Dense(22, activation='relu')(input_layer)
encoded = Dense(15, activation='relu')(encoded)
encoded = Dense(10, activation='relu')(encoded)
decoded = Dense(15, activation='relu')(encoded)
decoded = Dense(22, activation='relu')(decoded)
output_layer = Dense(input_dim, activation='sigmoid')(decoded)

autoencoder = Model(input_layer, output_layer)
autoencoder.compile(optimizer='adam', loss='mse')

# Define checkpoint callback to save the best autoencoder model
autoencoder_checkpoint = ModelCheckpoint('best_autoencoder.h5', monitor='val_loss', save_best_only=True, mode='min')

# Train the autoencoder
autoencoder.fit(X_train_noisy,
                 X_train,
                 epochs=50,
                 batch_size=256,
                 shuffle=True,
                 validation_data=(X_val, X_val),
                 callbacks=[autoencoder_checkpoint],
                 verbose = 1)
```

نتایج این آموزش به صورت تصویر زیر است : ( ۵ ایپاک اول و آخر )

```
Epoch 1/50
1333/1333 [=====] - 6s 4ms/step - loss: 14.0674 - val_loss: 0.9779
Epoch 2/50
1333/1333 [=====] - 6s 5ms/step - loss: 13.9525 - val_loss: 0.9416
Epoch 3/50
1333/1333 [=====] - 4s 3ms/step - loss: 13.9321 - val_loss: 0.9275
Epoch 4/50
1333/1333 [=====] - 4s 3ms/step - loss: 13.9203 - val_loss: 0.9179
Epoch 5/50
1333/1333 [=====] - 5s 4ms/step - loss: 13.9147 - val_loss: 0.9115
.
.
.

Epoch 45/50
1333/1333 [=====] - 7s 5ms/step - loss: 13.8233 - val_loss: 0.8710
Epoch 46/50
1333/1333 [=====] - 4s 3ms/step - loss: 13.8232 - val_loss: 0.8709
Epoch 47/50
1333/1333 [=====] - 4s 3ms/step - loss: 13.8233 - val_loss: 0.8709
Epoch 48/50
```

```

1333/1333 [=====] - 6s 4ms/step - loss: 13.8231 - val_loss: 0.8706
Epoch 49/50
1333/1333 [=====] - 5s 4ms/step - loss: 13.8230 - val_loss: 0.8706
Epoch 50/50
1333/1333 [=====] - 4s 3ms/step - loss: 13.8229 - val_loss: 0.8703
<keras.src.callbacks.History at 0x7c160c2b17b0>

```

ما از این شبکه برای حذف نویز استفاده می‌کنیم و خروجی آن به ورودی برنامه زیرکه طبقه‌بند ما است، داده می‌شود. لازم به ذکر است که در این طبقه‌بند، به عنوان تابع فعال‌ساز لایه آخر از softmax استفاده شده و همانطور که در مقاله ذکر شده، تابع هزینه categorical crossentropy و بهینه‌ساز Adam است.

```

# Build the classification model
classifier_input = Input(shape=(input_dim,))
x = Dense(22, activation='relu')(classifier_input)
x = Dense(15, activation='relu')(x)
x = Dense(10, activation='relu')(x)
x = Dense(5, activation='relu')(x)
x = Dense(2, activation='softmax')(x)

classifier = Model(classifier_input, x)
classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Define checkpoint callback to save the best classifier model
classifier_checkpoint = ModelCheckpoint('best_classifier.h5', monitor='val_loss', save_best_only=True, mode='min')

# Train the classifier
classifier.fit(X_train_denoised,
                y_train,
                epochs=50,
                batch_size=256,
                shuffle = True,
                validation_data=(X_val_denoised, y_val),
                callbacks=[classifier_checkpoint],
                verbose = 1)

```

در نهایت، می‌توانیم بهترین ضرایب را بازگردانیم. این فرآیند باعث می‌شود که مدل ما بهینه شده و عملکرد بهتری داشته باشد. (این دستور، وزن‌های بهترین مدل آموزش‌دیده را که در فایل `best\_classifier.h5` ذخیره شده‌اند، بارگذاری می‌کند. این وزن‌ها مجموعه‌ای از پارامترهای بهینه‌سازی شده هستند که به مدل یادگیری ماشین کمک می‌کنند تا بدون نیاز به آموزش دوباره، پیش‌بینی‌ها یا ارزیابی‌های مختلف را انجام دهد. استفاده از این روش، به بهینه‌سازی و بهبود بهره‌وری در فرآیند یادگیری ماشین کمک می‌کند).

```

# Load the best classifier model
classifier.load_weights('best_classifier.h5')

```

خروجی برای ۵ ایپاک اول و آخر به صورت زیر است :

```

Epoch 1/50
1333/1333 [=====] - 8s 4ms/step - loss: 0.1214 - accuracy: 0.9533 - val_loss: 0.0728 -
val_accuracy: 0.9722
Epoch 2/50
1333/1333 [=====] - 4s 3ms/step - loss: 0.0716 - accuracy: 0.9712 - val_loss: 0.0612 -
val_accuracy: 0.9767
Epoch 3/50
1333/1333 [=====] - 4s 3ms/step - loss: 0.0635 - accuracy: 0.9748 - val_loss: 0.0739 -
val_accuracy: 0.9718
Epoch 4/50
1333/1333 [=====] - 6s 4ms/step - loss: 0.0593 - accuracy: 0.9769 - val_loss: 0.0663 -
val_accuracy: 0.9738
Epoch 5/50
1333/1333 [=====] - 4s 3ms/step - loss: 0.0561 - accuracy: 0.9785 - val_loss:
.
.
.
Epoch 46/50
1333/1333 [=====] - 6s 4ms/step - loss: 0.0386 - accuracy: 0.9856 - val_loss: 0.0569 -
val_accuracy: 0.9788
Epoch 47/50
1333/1333 [=====] - 5s 4ms/step - loss: 0.0383 - accuracy: 0.9857 - val_loss: 0.0445 -
val_accuracy: 0.9832
Epoch 48/50
1333/1333 [=====] - 4s 3ms/step - loss: 0.0382 - accuracy: 0.9858 - val_loss: 0.0516 -
val_accuracy: 0.9797
Epoch 49/50
1333/1333 [=====] - 5s 4ms/step - loss: 0.0381 - accuracy: 0.9858 - val_loss: 0.0516 -
val_accuracy: 0.9803
Epoch 50/50
1333/1333 [=====] - 5s 4ms/step - loss: 0.0381 - accuracy: 0.9858 - val_loss: 0.0424 -
val_accuracy: 0.9845
<keras.src.callbacks.History at 0x7c16021837f0>

```

## بخش چهارم)

د. ماتریس درهمریختگی را روی قسمت آزمون داده‌ها رسم کنید و مقادیر Precision، Accuracy و Recall را گزارش کنید. فکر می‌کنید در مسائلی که توزیع برچسب‌ها نامتوابز است، استفاده از معیاری مانند f1score به تنها بی عملکرد مدل را به درستی نمایش می‌دهد؟ چرا؟ اگر نه، کدام معیار می‌تواند به عنوان مکمل Accuracy استفاده شود؟

در اینجا، به بررسی نتایج حاصل از کد زیر و تفسیر ماتریس درهمریختگی (Confusion Matrix) می‌پردازیم.

در واقع کد زیر معیارهای مختلف ارزیابی را روی داده‌های تست انجام می‌دهد.

```

from sklearn.metrics import accuracy_score, precision_recall_fscore_support, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the best classifier model
classifier.load_weights('best_classifier.h5')

# Predict on the denoised test set
y_test_pred = classifier.predict(X_test_denoised)
y_test_pred_classes = np.argmax(y_test_pred, axis=1)
y_test_true_classes = np.argmax(y_test, axis=1)

# Calculate metrics
accuracy = accuracy_score(y_test_true_classes, y_test_pred_classes)
precision, recall, f1, _ = precision_recall_fscore_support(y_test_true_classes, y_test_pred_classes, average='weighted')

# Print metrics
print(f'Accuracy: {accuracy*100:.2f}%')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1 Score: {f1:.2f}')

# Print metrics
print(f'Validation Set Metrics:')
print(f'Accuracy : {accuracy * 100:.2f}%')
print(f'F1 Score : {f1:.2f}')
print(f'Recall : {recall:.2f}')
print(f'Precision : {precision:.2f}')

# Plot confusion matrix
plt.figure(figsize=(7, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            annot_kws={"size": 14}, linewidths=.5, linecolor='black')
plt.title('Confusion Matrix', fontsize=16)
plt.xlabel('Predicted Class', fontsize=14)
plt.ylabel('Actual Class', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.tight_layout()
plt.show()

```

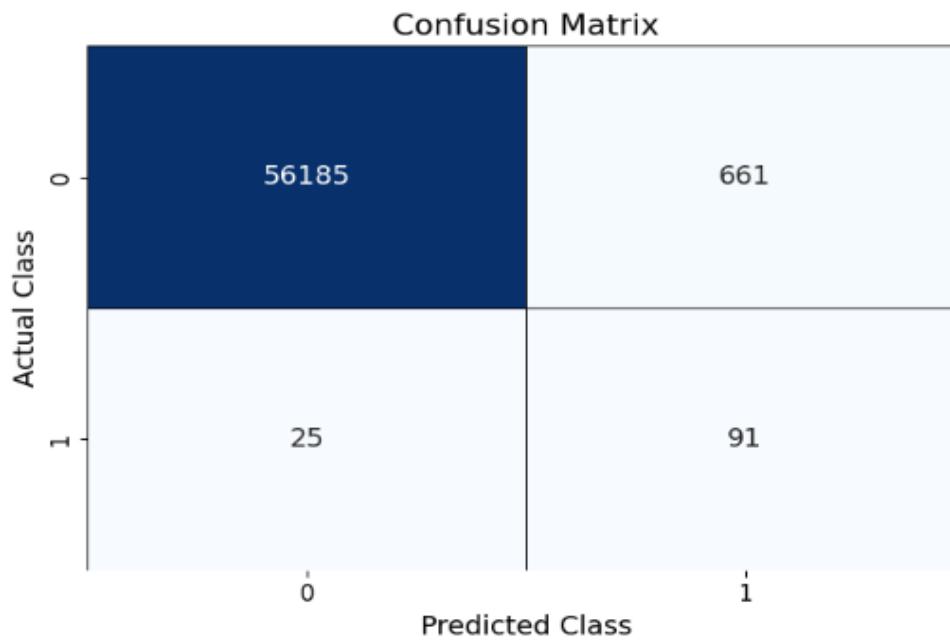
ما در ابتدا به دو نمای کلی نتایج کد فوق اشاره می کنیم که در زیر آمده اند : (شکل صفحه بعد نمایش دهنده ماتریس درهم ریختگی است که نشان می دهد چقدر مدل در تفکیک و دسته بندی دقیق نمونه ها توانایی دارد. این ماتریس به طور خاص برای مدل هایی که از **autoencoder** استفاده می کنند و در پردازش تصویر و ویدیو به کار می روند، اهمیت بیشتری دارد).

	print(classification_report(y_test_true_classes, y_test_pred_classes))				
	precision	recall	f1-score	support	
0	1.00	0.99	0.99	56846	
1	0.12	0.78	0.21	116	
accuracy			0.99	56962	
macro avg	0.56	0.89	0.60	56962	
weighted avg	1.00	0.99	0.99	56962	

```

1781/1781 [=====] - 3s 1ms/step
Accuracy: 98.80%
Precision: 1.00
Recall: 0.99
F1 Score: 0.99
Validation Set Metrics:
Accuracy : 98.80%
F1 Score : 0.99
Recall : 0.99
Precision : 1.00

```



با توجه به نتایج فوق به دقت حدود ۹۹ درصد رسیدیم و معیار دقت به تنها بی کارآمد نیست و نتایج حاصل از دیگر معیار ها هم در تصاویر فوق قابل روئت است.

در مواردی که داده ها نامتوازن هستند، مثل تعداد نمونه های هر کلاس که با یکدیگر متفاوت است، استفاده از دقت به تنها بی ممکن است به طور نادرستی عملکرد واقعی مدل را نشان دهد. به عبارت دیگر، دقت (Accuracy) ممکن است گمراه کننده باشد و اطلاعات کافی درباره کیفیت واقعی مدل را ارائه ندهد.

برای مثال، فرض کنید که بیشتر داده های ماشین آموز به یک کلاس از داده ها تعلق دارد و کمتر به کلاس دیگر. اگر مدل تمام نمونه ها را به عنوان این کلاس اصلی پیش بینی کند، دقت بالایی خواهد داشت زیرا بیشتر پیش بینی ها درست است. اما، این نشانگر بهترین عملکرد مدل نیست، زیرا ممکن است از نظر کاربردی مفید نباشد. به عبارت دیگر، ما ممکن است اهمیت بیشتری به تشخیص نمونه های واقعی مثبت (اعضای کلاس کمتر) دهیم.

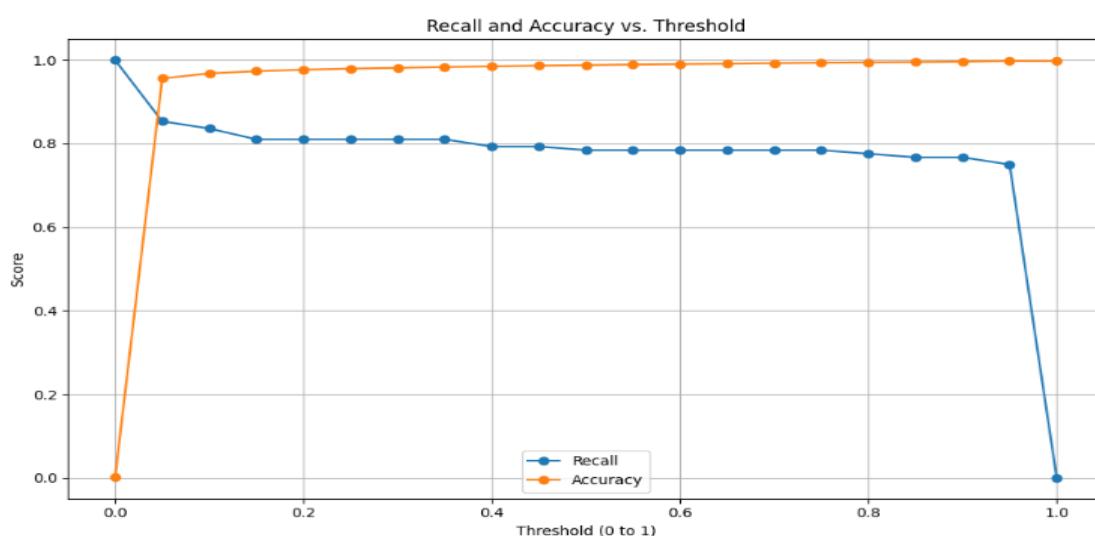
در اینجا، معیارهای دیگری مانند دقت در پیش‌بینی‌های مثبت (Precision) که نسبت نمونه‌های واقعی مثبت به کل نمونه‌هایی که به عنوان مثبت پیش‌بینی شده‌اند را نشان می‌دهد، و بازیابی (Recall) که نسبت نمونه‌های واقعی مثبت به کل نمونه‌هایی واقعی مثبت را نشان می‌دهد، اهمیت بیشتری پیدا می‌کنند. همچنین، F1 score که یک معیار ترکیبی از دقت و بازیابی است، نیز می‌تواند در شرایط نامتوازنی کلاس‌ها بهترین انتخاب باشد، زیرا از هر دو جنبه اهمیت می‌دهد.

پس به طور خلاصه، در صورتی که داده‌ها نامتوازن باشند، استفاده از معیارهای دیگر به جای دقت به تنها‌ی، بهترین عملکرد واقعی مدل را ارزیابی می‌کند و به تصمیم‌گیری در مورد ارزش و استفاده از مدل کمک می‌کند. همچنین ماتریس درهم‌ریختگی (Confusion Matrix) نیز یک ابزار مهم است که به تفکیک نتایج پیش‌بینی‌های مدل بر اساس واقعیت کمک می‌کند.

## بخش پنجم)

۱. با آستانه‌های مختلف برای Recall & Accuracy عمل کرد مدل را بررسی کرده و نمودار Oversampling را مانند شکل ۷ مقاله ترسیم کنید.

در اینجا ما به تأثیرگذاری آستانه‌ها در تصمیم‌گیری در طبقه‌بندی می‌پردازیم. آستانه‌ها به عنوان نقطه‌ای است که برای تصمیم‌گیری در مدل‌های طبقه‌بندی مورد استفاده قرار می‌گیرند. در اینجا ما آستانه‌های مختلف را در نظر می‌گیریم که با استفاده از 'thresholds = np.arange(0.0, 1.05, 0.05)' تعریف شده‌اند.



در نمودار بالا که بر اساس آستانه‌های مختلف ارائه شده است، دقت مدل در طول آستانه‌ها به خوبی قابل مشاهده است. با افزایش آستانه، دقت به طور کلی بهبود می‌یابد. اما، مقدار Recall در برخی موارد با افزایش آستانه به طور جزئی کاهش می‌یابد. این نشان می‌دهد که افزایش آستانه ممکن است منجر به کاهش تشخیص نمونه‌های مثبت (Recall) شود که ممکن است نیاز به تعديل داشته باشد.

پس همانطور که دیده می‌شود، نمودار ما تقریباً با نمودار مقاله یکسان است. همچنین، در آستانه‌ی ۰,۵ که مقدار قبلی مدنظر است، مقدار recall برابر با ۰,۸ و دقت (accuracy) برابر با ۰,۹۹ شده است. به طور نهایی، در مدل ما recall به مقدار صفر می‌رسد و دقت به ۱۰۰ درصد می‌رود که نشان‌دهنده‌ی این است که مدل نتوانسته است هیچ‌کدام از داده‌های کلاس تقلب را تشخیص دهد. به طور قطعی، اگر آستانه‌ی تصمیم‌گیری را کمتر کنیم، مدل بیشتر داده‌ها را به عنوان داده‌های کلاس تقلب شناسایی می‌کند، زیرا تعداد بیشتری از داده‌ها از آستانه بیشتری برخوردارند که این باعث می‌شود که به کلاس ۱ تعلق داده شوند و در نتیجه، داده‌های نرمال به درستی تشخیص داده نمی‌شوند که این موجب کاهش دقت مدل می‌شود.

کد زیر به ارزیابی استفاده از SMOTE با استراتژی‌های نمونه‌برداری مختلف می‌پردازد. هر مرحله از آموزش مدل بهبود یافته و نتایج عملکرد سیستم به صورت نمودار‌های زیر نمایش داده می‌شود.

(در مسائل داده‌کاوی و یادگیری ماشین، مشکل نمونه‌های نامتوازن معمولاً وجود دارد که یک کلاس نمونه‌های بیشتری دارد نسبت به کلاس دیگر. برای حل این مشکل، از روش‌هایی مانند SMOTE استفاده می‌شود که به کمک استراتژی‌های نمونه‌برداری متفاوت، نمونه‌های جدیدی برای کلاس اقلیتی تولید می‌کنند. مقادیر در `sampling\_strategies` نشان‌دهنده نسبت‌های مختلف اضافه کردن نمونه‌های جدید به کلاس اقلیتی نسبت به تعداد نمونه‌های اصلی آن کلاس هستند. این روش‌ها به کمک تولید نمونه‌های مصنوعی، عملکرد مدل‌های طبقه‌بندی را در مواجهه با داده‌های نامتوازن بهبود می‌بخشند و باعث می‌شود که مدل بهتری برای پیش‌بینی و تصمیم‌گیری داشته باشیم.)

```
sampling_strategies = [0.1, 0.25, 0.5, 0.75]

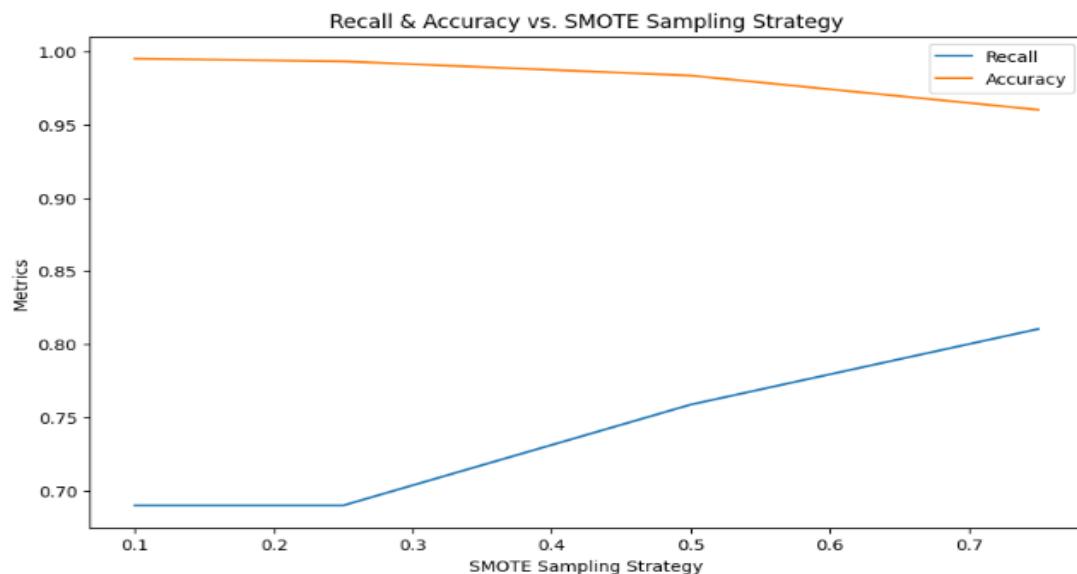
results = []

for strategy in sampling_strategies:
    # Apply SMOTE with the current sampling strategy
    smote = SMOTE(sampling_strategy=strategy, random_state=24)
    X_train_res, y_train_res = smote.fit_resample(X_train, np.argmax(y_train, axis=1))
    y_train_res = to_categorical(y_train_res, num_classes=2)

    print(f'Resampled dataset shape with strategy {strategy}:', X_train_res.shape, y_train_res.shape)
```

نمودار و نتایج زیر نشان می‌دهند که با افزایش آستانه SMOTE، عملکرد مدل بهبود می‌یابد، در حالی که نتایج به طور کلی با نتایج مقاله همخوانی دارند و این نشان می‌دهد که روش‌های ما در مواجهه با داده‌های نامتوازن مؤثر هستند.

SMOTE Threshold	Recall Rate	Accuracy
0	0.689655	0.995295
1	0.689655	0.993452
2	0.758621	0.983726
3	0.810345	0.960342
		--



## بخش ششم)

و. مدل را با استفاده از داده‌های نامتوازن و بدون حذف نویز، آموزش داده و موارد بخش قبلی را گزارش کنید و نتایج دو مدل را با هم مقایسه کنید.

در این قسمت، ما به بررسی عملکرد مدل در مواجهه با داده‌های نامتوازن و اثر استفاده از روش‌های خاص بر آن می‌پردازیم. ابتدا از روش autoencoder برای حذف نویز از داده‌ها صرف‌نظر می‌کنیم. این به این معناست که مدل شبکه عصبی برای تبدیل داده‌ها به فضای ویژگی‌های معنادار اجرا نمی‌شود و از این روی، هیچ عملیاتی مانند استفاده از روش SMOTE برای افزایش تعداد نمونه‌ها و بالанс کردن داده‌ها انجام نمی‌شود.

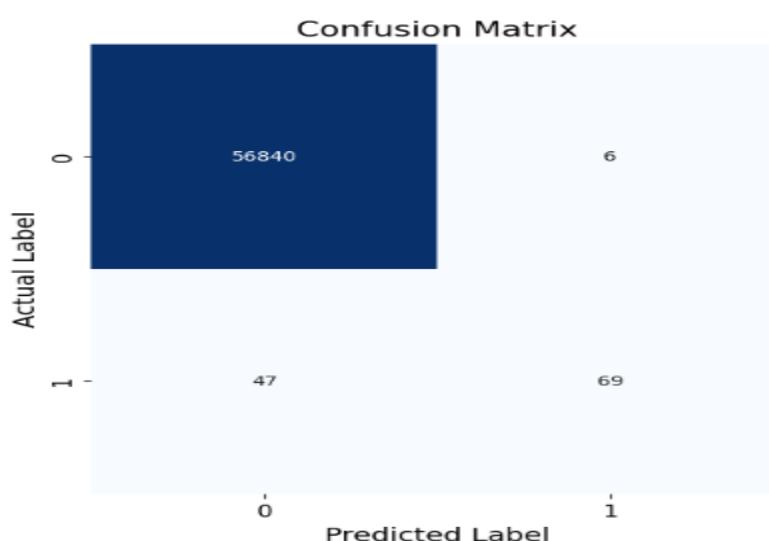
این تغییرات منجر به بهبود قابل توجهی در عملکرد مدل می‌شود که در ماتریس در هم ریختگی و گزارش طبقه بندهی که در ادامه ارائه می‌شود، قابل مشاهده است. به وضوح دیده می‌شود که دقت کلی مدل به طور قبل توجهی افزایش یافته است و این نشان می‌دهد که مدل دچار گمراهی کمتری در پیش‌بینی کلاس‌های مختلف نمی‌شود، به خصوص در مورد کلاس ۱ که با استفاده از SMOTE بالанс شده است.

همچنین، با مقایسه ماتریس در هم ریختگی در این حالت با حالت قبلی، مشاهده می‌شود که تعداد نمونه‌هایی که مدل به طور دقیق تشخیص داده است، افزایش یافته است. این نشان می‌دهد که استفاده از روش SMOTE در اینجا بهبود معناداری در عملکرد مدل داشته است و ما می‌توانیم با تنظیم آستانه‌های مدل، به نتایج بهتری دست پیدا کنیم که نمایان‌گر تأثیر مثبت استفاده از SMOTE برای طبقه‌بندی است.

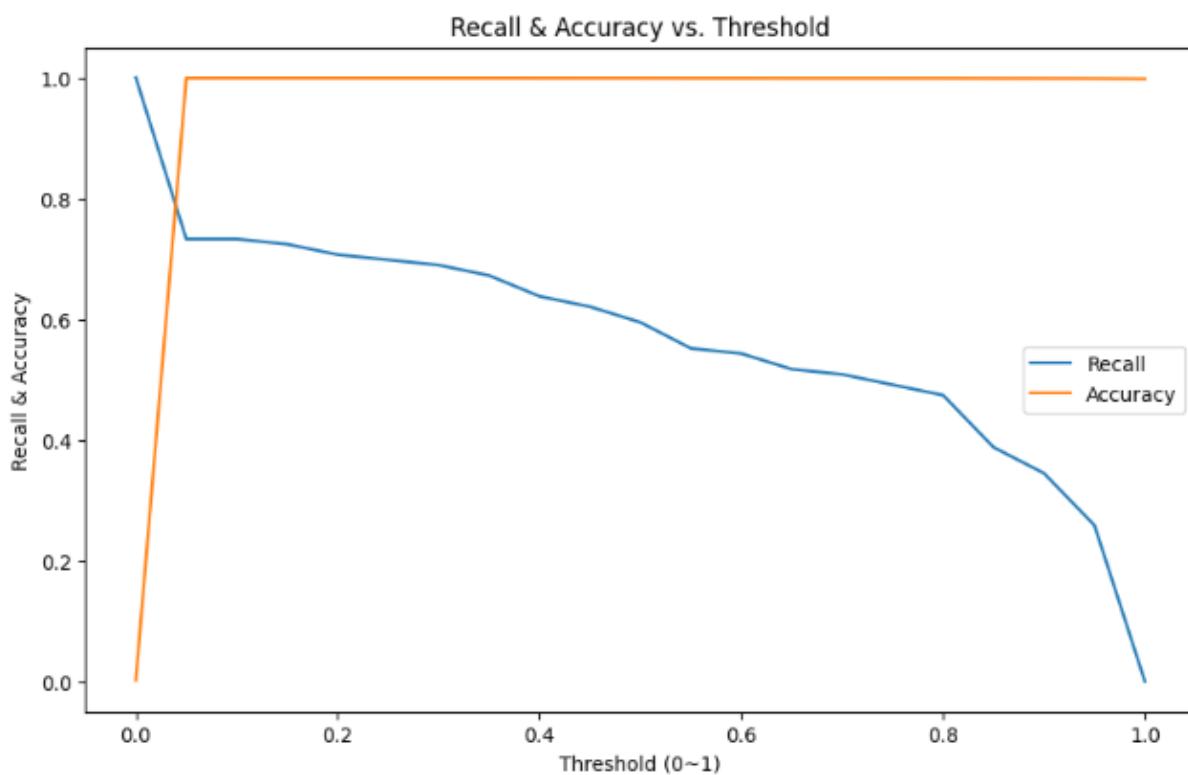
در نهایت، از طریق نمودار تاثیر تغییرات recall و دقت (accuracy) بدون استفاده از auto encoder و SMOTE ملاحظه می‌شود که مدل با استفاده از SMOTE، به تعمیم‌پذیری بیشتری دست یافته است که نشان می‌دهد فرآیند یادگیری مدل به طور سریع‌تر انجام شده است و این تأثیر مستقیمی بر کاهش عملکرد مدل در طول زمان ندارد.

نتایج بدون استفاده از SMOTE به صورت زیر هستند:

Accuracy: 99.91%
F1 Score: 0.72
Recall: 0.59
Precision: 0.92
precision      recall      f1-score      support
0      1.00      1.00      1.00      56846
1      0.92      0.59      0.72      116
accuracy      1.00      56962
macro avg      0.96      0.80      0.86      56962
weighted avg      1.00      1.00      1.00      56962



پس در کل در اینجا، ما دیگر از روش‌های oversampling و DEA استفاده نمی‌کنیم و به جای آن، تمام داده‌ها را به classifier خود ارائه می‌دهیم. این به این معنی است که تمام داده‌ها به صورت اصلی به مدل داده می‌شوند. به دلیل این تغییر، در مورد پیش‌فرض قبلی که اشتباه مطرح شده بود، دیگر مسئله oversampling وجود ندارد. با این رویکرد، دقت مدل پس از تجاوز از یک آستانه مشخص به ۱۰۰ درصد افزایش می‌یابد، اما recall به طور قابل توجهی کاهش می‌یابد تا به طور نهایی به مقدار صفر برسد. دلیل این اتفاق این است که مدل نمی‌تواند داده‌های تقلب را به درستی تشخیص دهد. همچنین، در مقایسه با مقاله مرجع، مدل ما بهبود قابل توجهی داشته است. (نمودار زیر هم صحت این مطلب را می‌رساند.)



پیان