به نام خدا

MP3

دانشجو : مصطفی نبی پور

شماره دانشجویی : ۴۰۱۱۲۸۶۴

لینک google colab :

https://drive.google.com/drive/folders/1AELZoWP33m9PRNRl2z_U4-RTUXsKTVj_?usp=sharing

لینک github :

https://github.com/mostafanb77/ML_4022_MP3

سوال ۱)آ)

در مرحله اول داده ها را اد میکنیم (گزارش کد در آخر سوال قرار داده شده است) سپس خواسته های مسئله را بدست می آوریم که بدین ترتیب میباشند :

اول از همه داده های مسئله در حالت dataframe بدین شکل میباشند :

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

150 rows × 5 columns

که داده های target ما بدین شکل میباشند :

```
target
0    50
1    50
2    50
Name: count, dtype: int64
```

۳ کلاس داریم که تعداد هر کدام از کلاس ها ۵۰ تا میباشد.

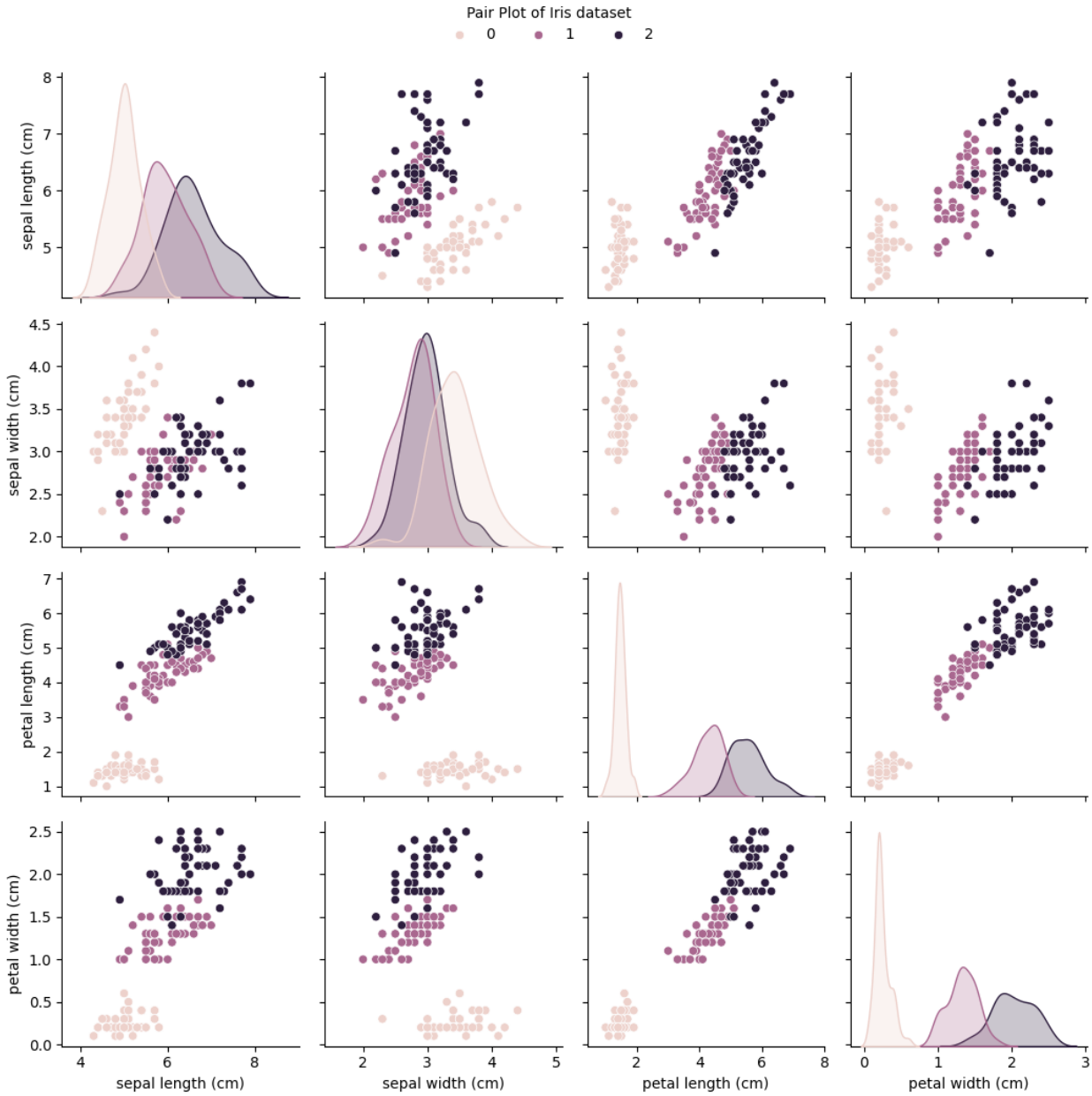
ابعاد داده ها همانطور که در بالا معلوم است دارای ۴ ویژگی و ۱۵۰ سطر میباشد :

```
((150, 5), 150)
```

میانگین و واریانس ویژگی ها بدین شکل میباشد : (البته target جزو ویژگی ها نمیباشد بلکه کلاس داده ها است.)

```
Mean of features:
 sepal length (cm)    5.843333
sepal width (cm)      3.057333
petal length (cm)     3.758000
petal width (cm)      1.199333
target                1.000000
dtype: float64

Variance of features:
 sepal length (cm)    0.685694
sepal width (cm)      0.189979
petal length (cm)     3.116278
petal width (cm)      0.581006
target                0.671141
dtype: float64
```



تصویر بالا در واقع پراکندگی داده ها را به ما نشان میدهد. عدد و رنگ کلاس ها در بالا آورده شده است.

برای اینکه بهتر بتوان این تصویر را تحلیل کرد که آیا نیازی به کاهش بعد دارد یا خیر میتوان بدین شکل گفت که به عنوان مثال اگر به ردیف petal length و sepal length نگاه کنید خواهید دید که میتوان تقریبا ۳ کلاس را از هم جدا کرد (کلاس زرد رنگ که خیلی راحت تر است.)همینجور که در تصویر سازی داده ها در همان ردیف مشاهده میکنید هم کلاس زرد رنگ را به راحتی میتوان جدا کرد ولی دو کلاس دیگر از آنجایی که با هم کمی همپوشانی دارند نمیتوان این کار را کرد.

برعکس این را میتوان در sepal width با sepal length مشاهده کرد هیمنطور که میبینید از آنجایی که داده ها خیلی با هم overlap دارند نمیتوان به راحتی از هم جدا کرد.همینطور اگر پراکندگی داده ها را در sepal width مشاهده کنیم متوجه میشویم که کلاس های داده ها باهم همپوشانی زیادی دارند پس کار ما برای طبقه بندی با این ویژگی سخت میباشد.

برای اینکه بتوان راحت تر تصمیم گرفت که نیازی به کاهش بعد داریم یا خیر یکی از راه ها استفاده از ماتریس همبستگی میباشد زیرا در واقع کاهش بعد مواقعی به کار ما می آید که ویژگی هایی وجود داشته باشند که همبستگی بالایی با هم داشته باشند (در واقع یکی از ویژگی ها اضافی به شمار می آید.) :

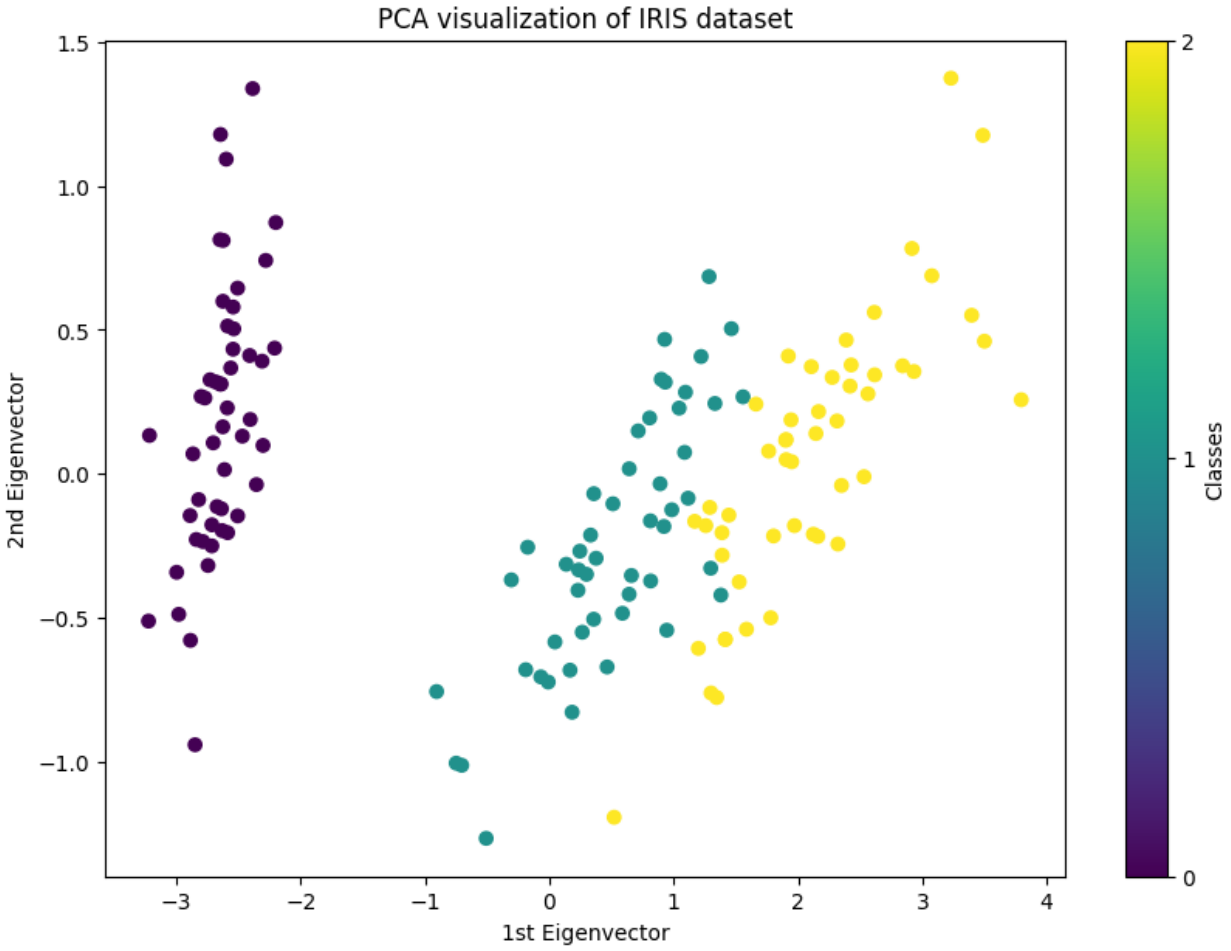
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941	0.782561
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126	-0.426658
petal length (cm)	0.871754	-0.428440	1.000000	0.962865	0.949035
petal width (cm)	0.817941	-0.366126	0.962865	1.000000	0.956547
target	0.782561	-0.426658	0.949035	0.956547	1.000000

همینجور که میبنید مانند قبل که توضیح دادیم sepal length با petal length همبستگی بالایی دارند برای همین همانطور که در تصویر قبل دیدیم میتوانستیم راحت تر آنها را جدا کنیم.

ولی برای sepal width با sepal length همبتسگی خیلی پایینی وجود دارد به همین دلیل نمیتوان این ۲ را از هم به راحتی تفکیک کرد(نمیتوان کاهش بعد را بر روی این ها انجام داد.)

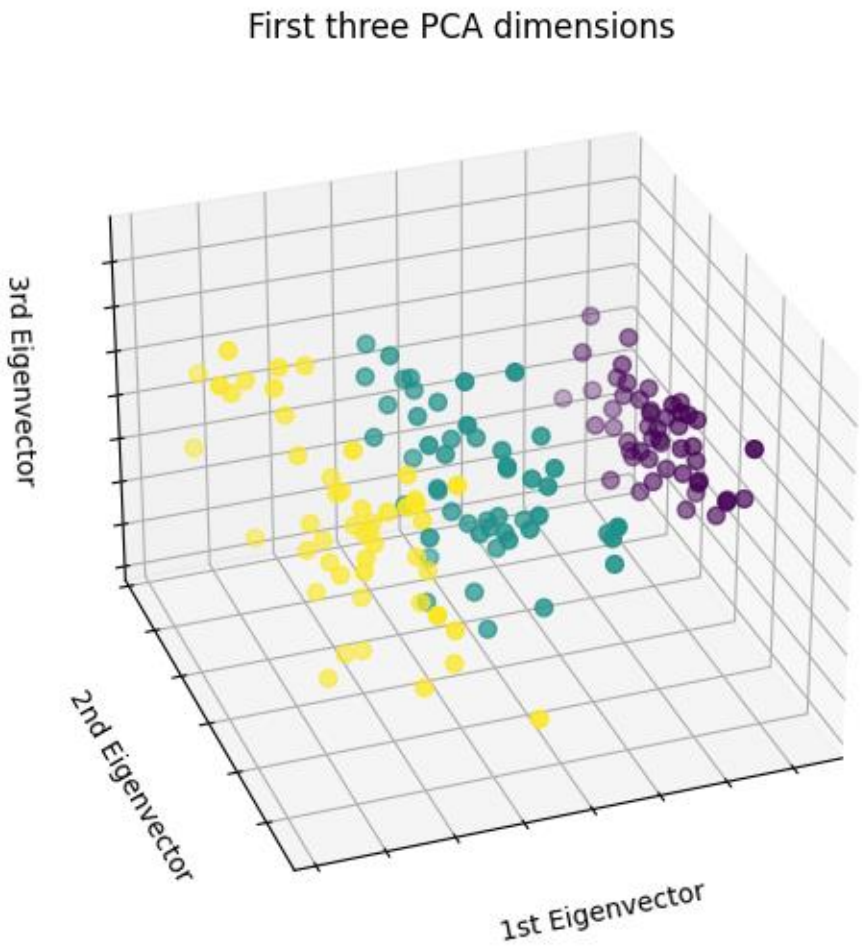
خوب پس حدس اولیه که میزنیم این است که میتوان از ۴ ویژگی که داریم ۳ تای آنها را به ۱ کاهش دهیم در اینصورت ما ۲ ویژگی خواهیم داشت. این عمل را در ابتدا با PCA با ۲ مولفه انجام میدهیم تا نتیجه را ببینیم :

(PCA در واقع یک تبدیل کننده خطی میباشد که داده ها را بر روی متخصات orthogonal پخش میکند که به آنها principal component گفته میشود که این مختصات بدین صورت است که بر روی مختصات اول بیشترین واریانس داده ها یافت میشود و بر روی بقیه مختصات به واریانس به صورت نزولی میباشد.)



خوب هیمنجورکه میبینید کلاس داده های ۰ به راحتی قابل جدا شدن هستند ولی کلاس های داده های ۱ و۲ کمی مزدیک به هم میباشند ولی بازهم راحت میتوانند جدا شوند.

برای ۳ مولفه هم انجام میدهیم که بدین شکل میشود :



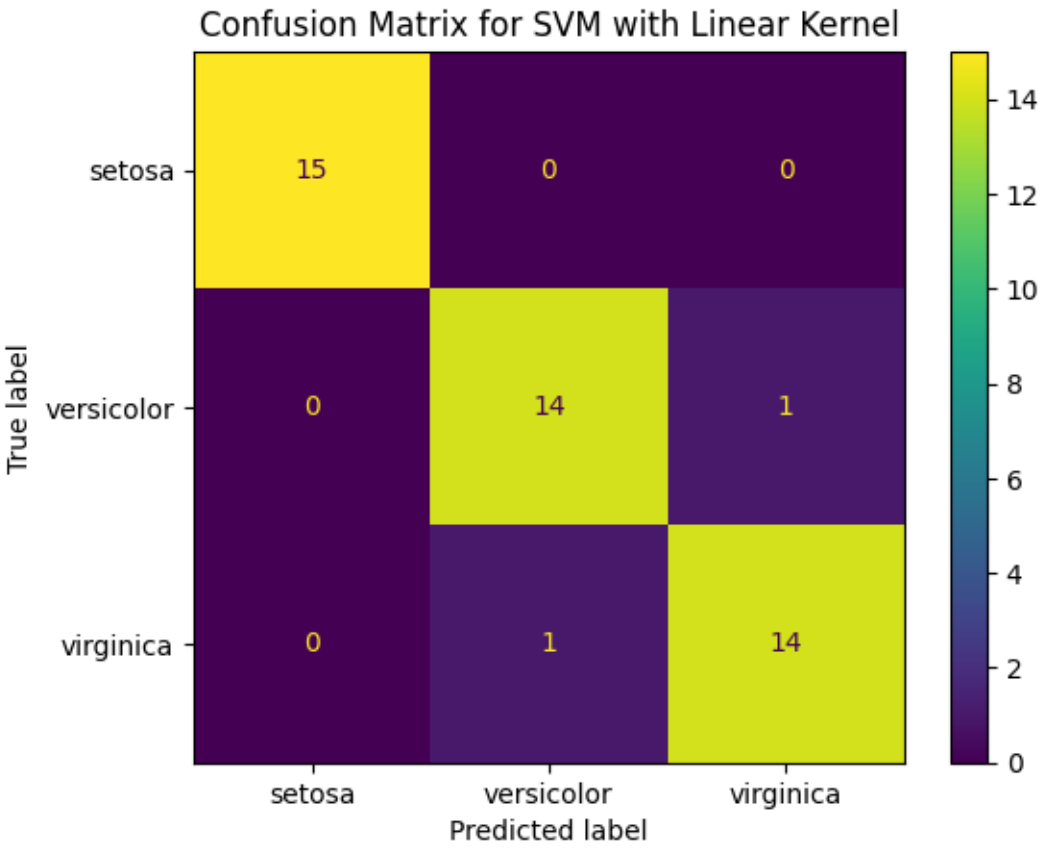
خوب همینجور که میبنید نتیجه مانند ۲ مولفه شده پس میتوان با ۳ مولفه نیز این کار را انج

ب)

ما در اینجا از LDA استفاده میکنیم در واقع دلیل استفاده ما از LDA این است که این روش برای مسائل supervised به کار میرود ولی PCA برای مسائل unsupervised به کار میرود.دلیل دیگر آن نیز اینست که در LDA تلاش برای این است که فاصله کلاس ها از هم زیاد و واریانس آنها کم باشند یعنی کلاس ها بهتر میتوانند جدا شوند ولی در PCA تلاش اینست که محور هایی را پیدا کند که بیشترین واریانس داده ها را بوجود آورد پس با این حال، این مؤلفه‌ها ممکن است لزوماً برای تمایز بین کلاس‌ها بهترین نباشند.

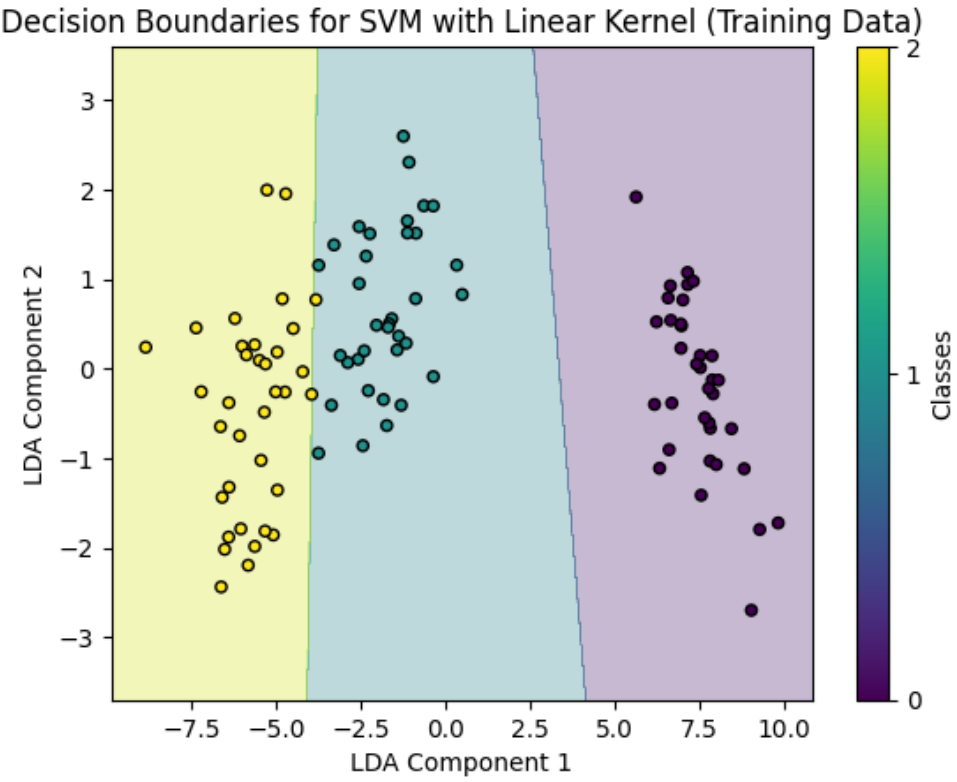
در واقع در LDA تمرکز بر ویژگی هایی است که به بهترین نحو کلاس ها را از هم جدا می کنند.

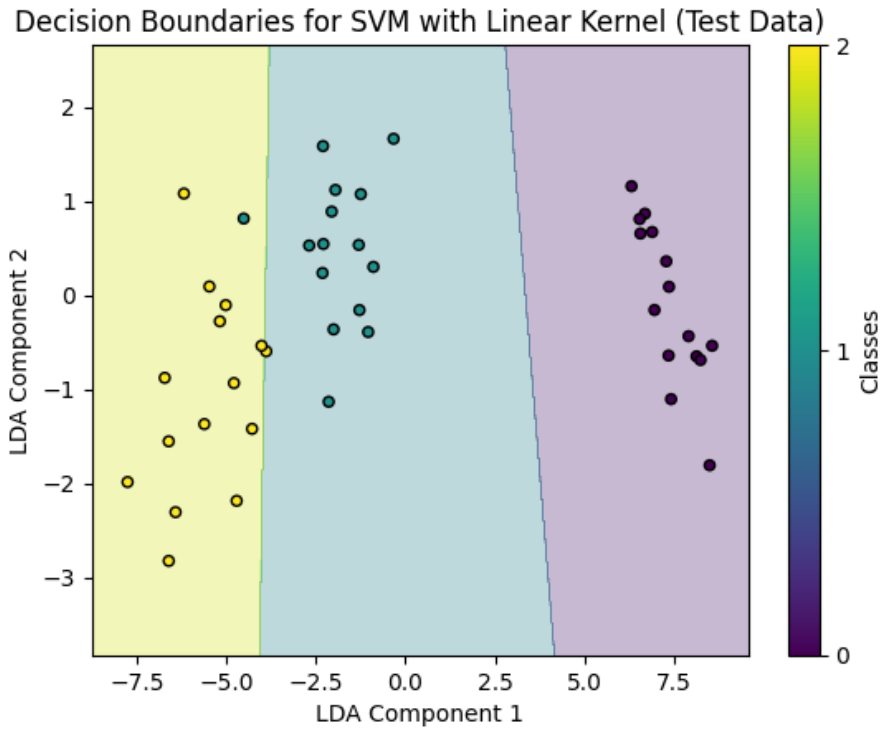
نتایج :(با random state = 64)



	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	15
versicolor	0.93	0.93	0.93	15
virginica	0.93	0.93	0.93	15
accuracy			0.96	45
macro avg	0.96	0.96	0.96	45
weighted avg	0.96	0.96	0.96	45

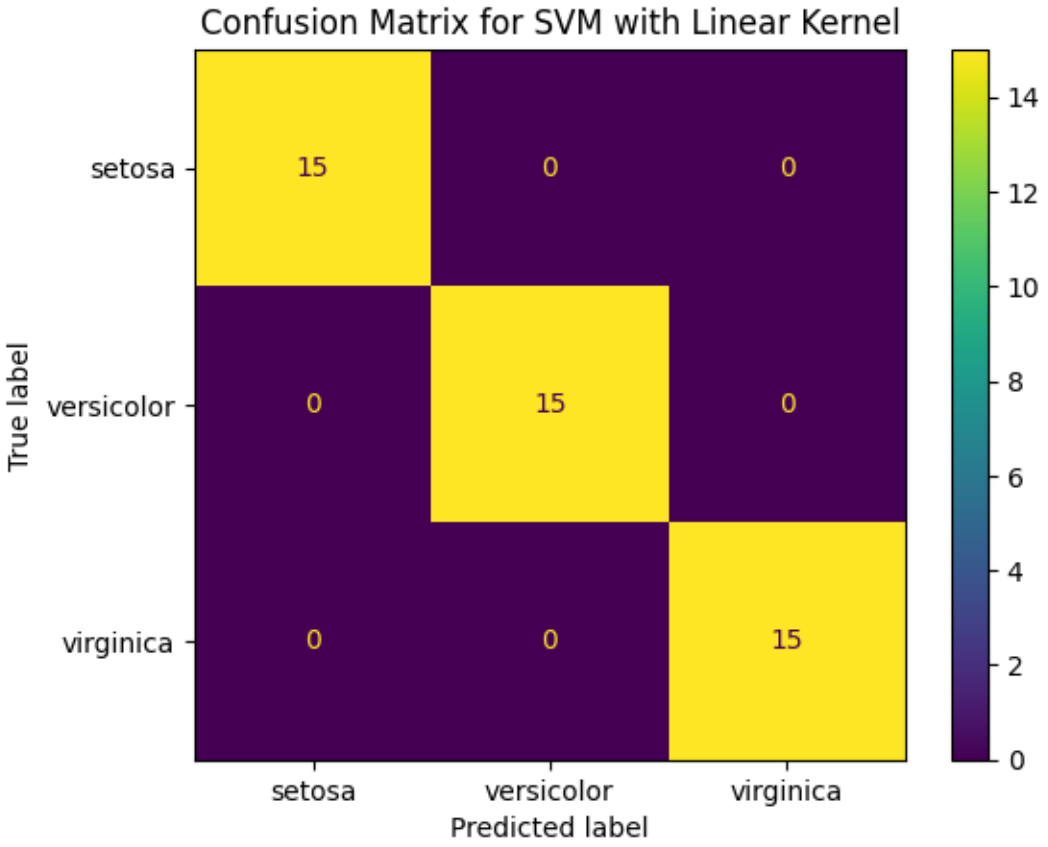
همینطور که میبنید دقت مدل بر روی داده های تست ۹۶ درصد میباشد که دقت خیلی خوبی میباشد.



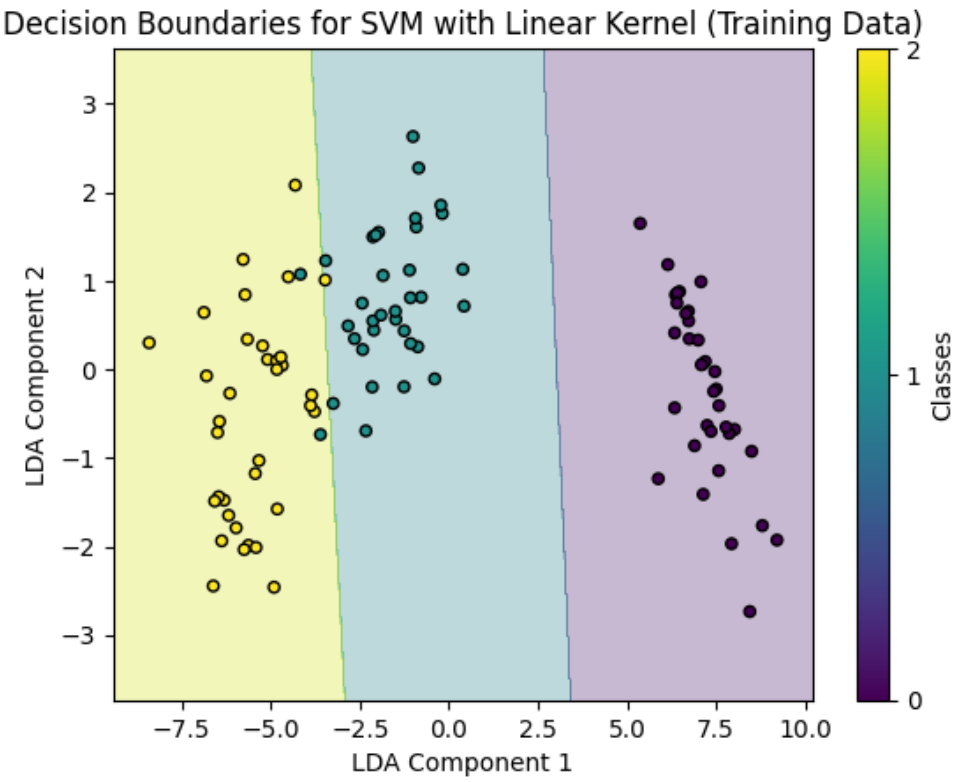


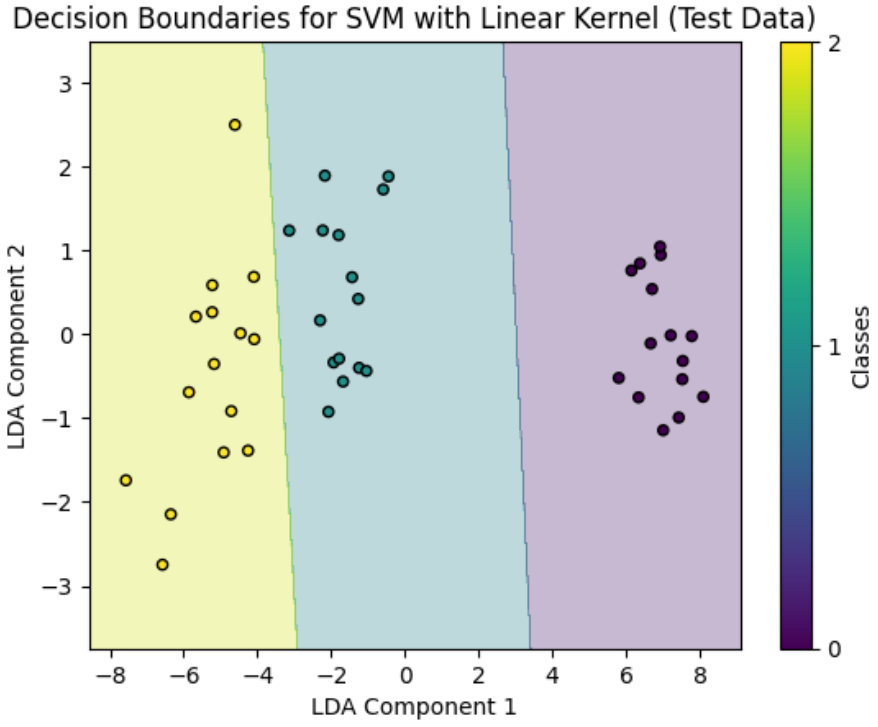
همینجور که میبینید مدل ما نتوانسته کاملا داده های تست و آموزش کلاس هایشان را به خوبی از هم جدا کند.(کلاس ۰ را به خوبی توانسته جدا کند.)

با random state = 0 :



	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	15
versicolor	1.00	1.00	1.00	15
virginica	1.00	1.00	1.00	15
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45


Decision Boundaries for SVM with Linear Kernel (Training Data)
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

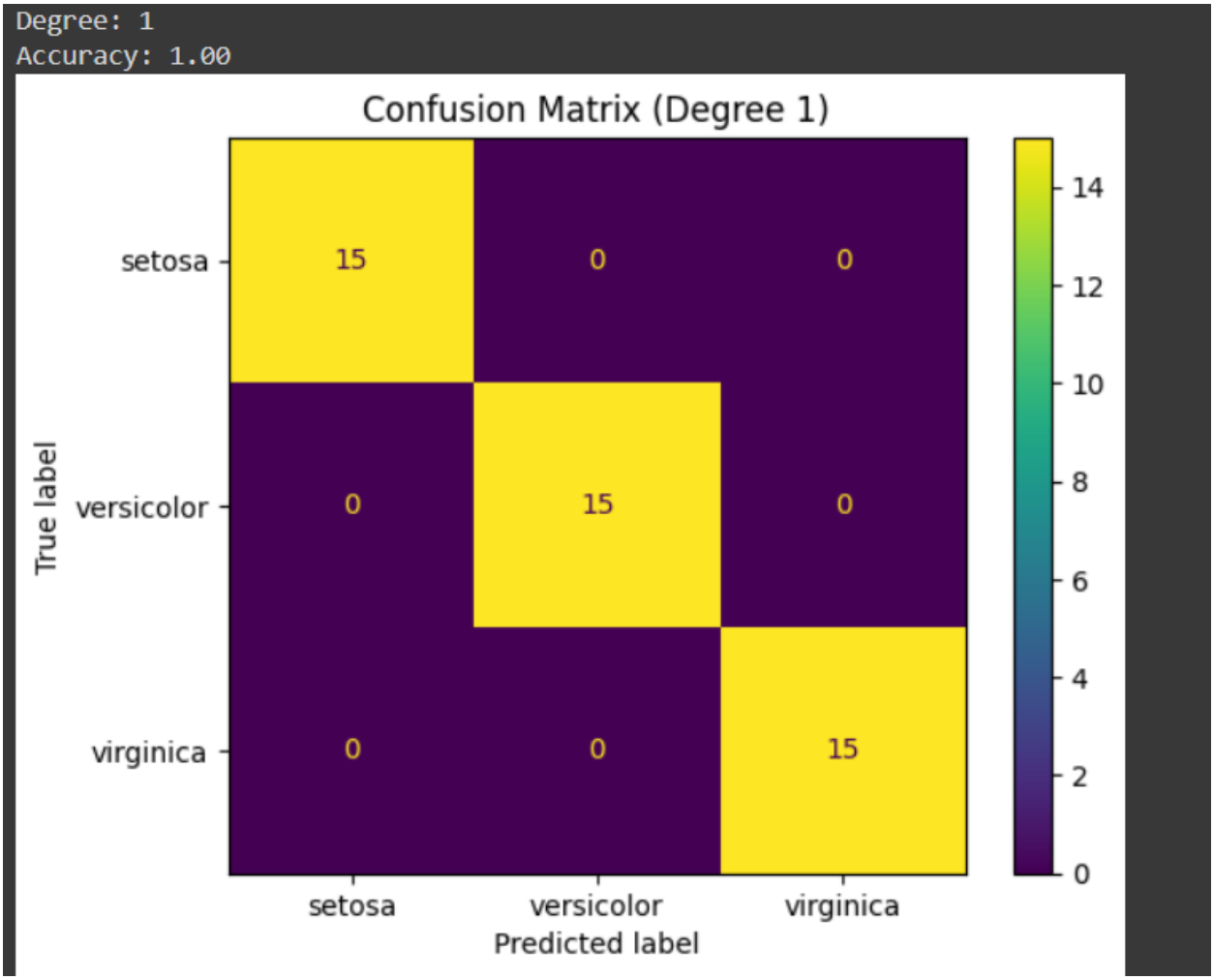

Decision Boundaries for SVM with Linear Kernel (Test Data)
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−8
−6
−4
−2
0
2
4
6
8
2
1
0

خوب همینجور از نتایجی که در بالا گذاشته ام میتوانید متوجه شوید که مدل به خوبی آموزش دیده است یعنی overfitting بر روی داده های آموزش نداشته است و توانسته است که به خوبی داده های تست را از هم جدا کند.این عمل را نتوانسته بر روی داده های آموزش انجام دهد.دقت مدل ما ۱۰۰ درصد شده یعنی توانسته به خوبی همه داده ها را پیش بینی کند.

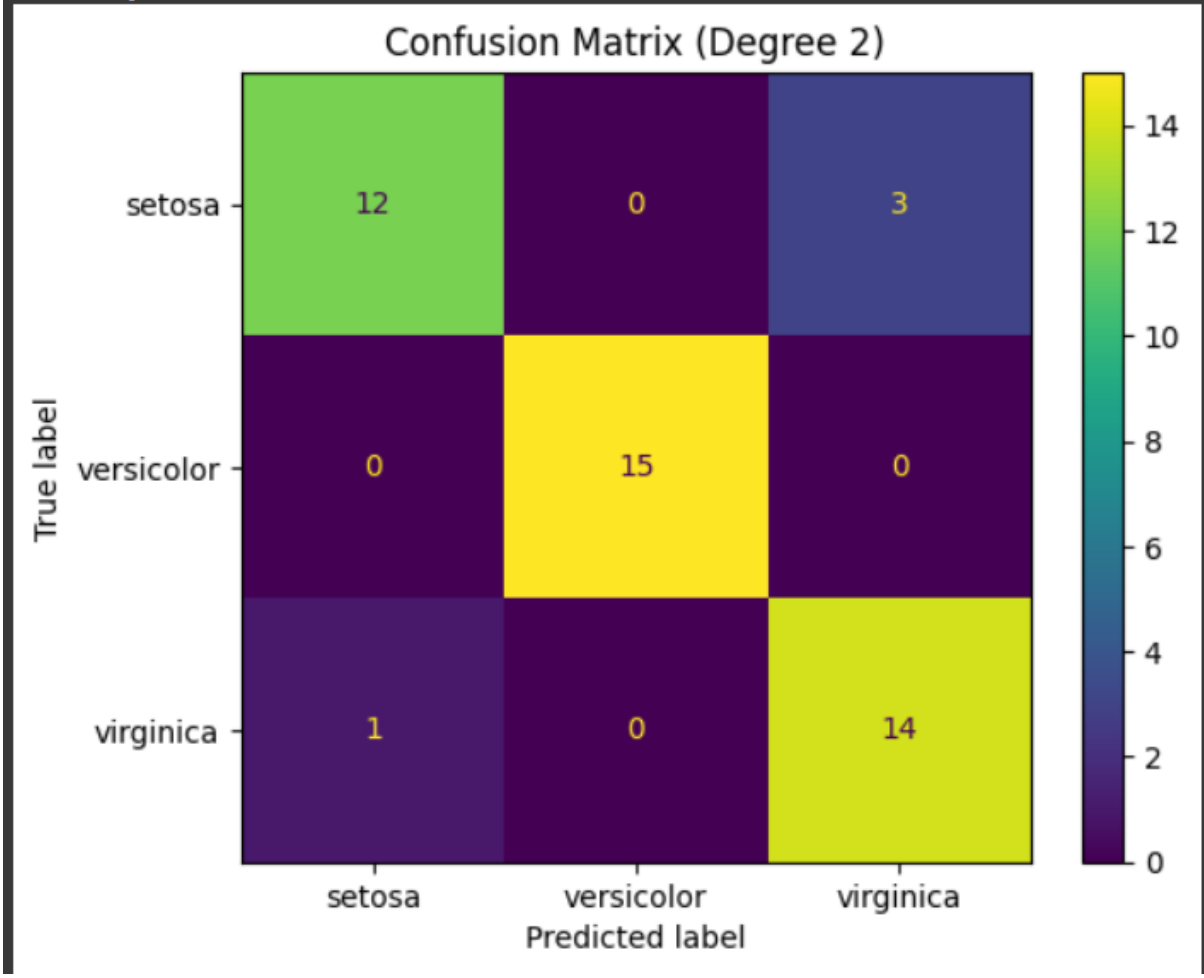
ج)

این قسمت را نیز با random state = 0 انجام میدهیم و از روش LDA با ۲ مولفه استفاده میکنیم که دلایل آن در بخش قبل گفته شده است.(بهترین دلیل که میتوان گفت این است که LDA برای ماکسیمم کردن فاصله کلاس ها به کار میرود. معمولا PCA برای داده هایی با ابعاد بالا به کار میرود که در اینجا ابعاد کم میباشد.)

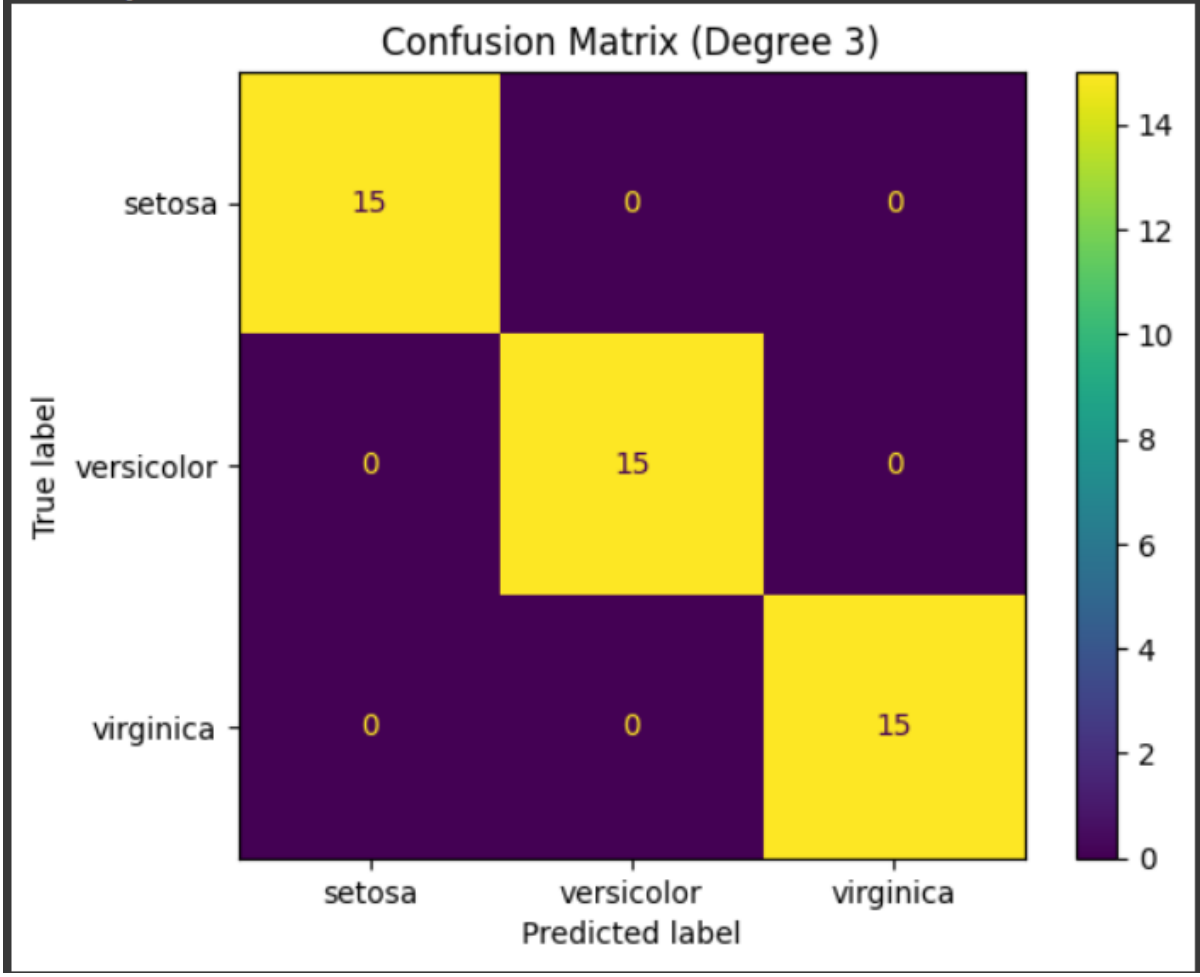
ما برای هسته چند جمله ای از ۱ تا ۱۰ مدل خود را آموزش دادیم و نتایجی که از این مدل بدست آوردیم accuracy,confusion matrix میباشد :



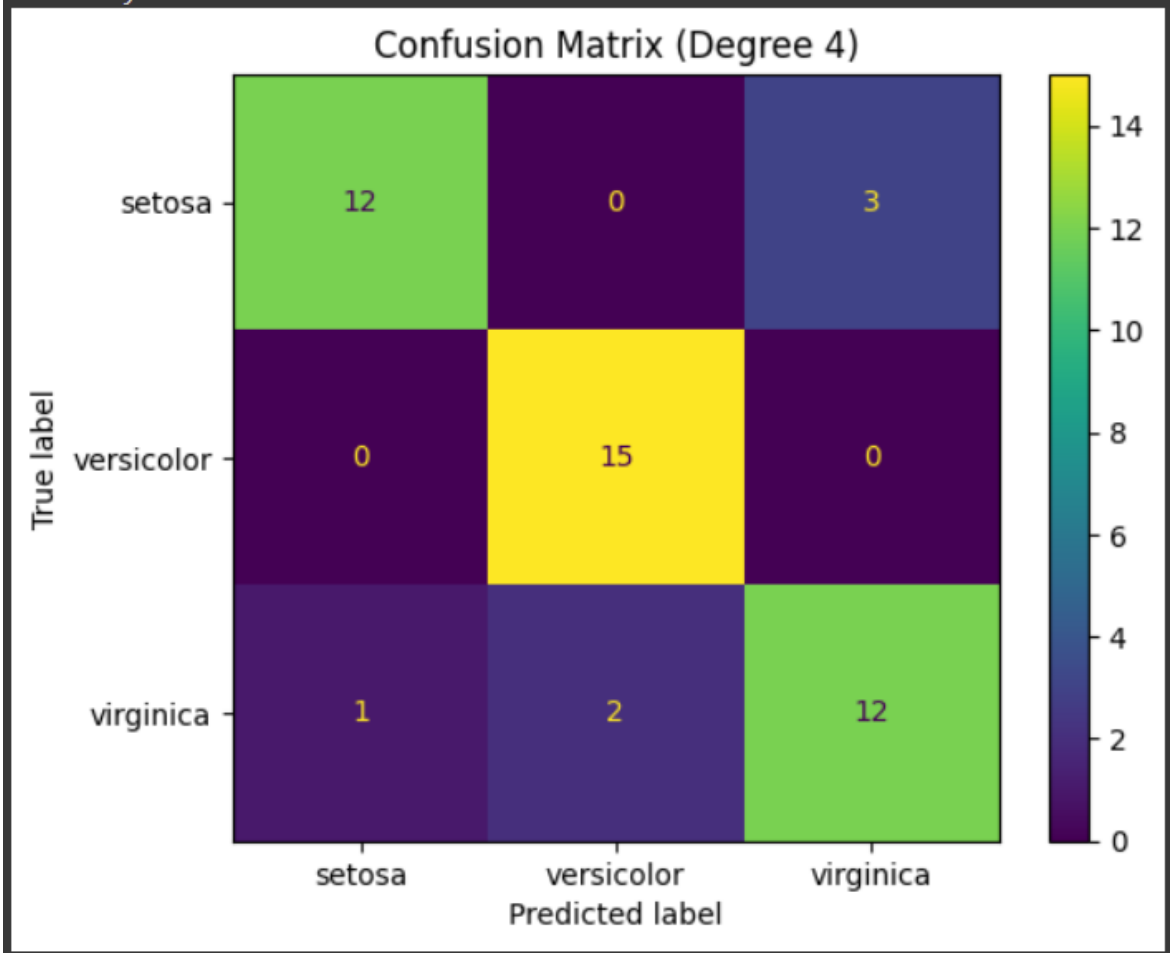
```
Degree: 2
Accuracy: 0.91
```



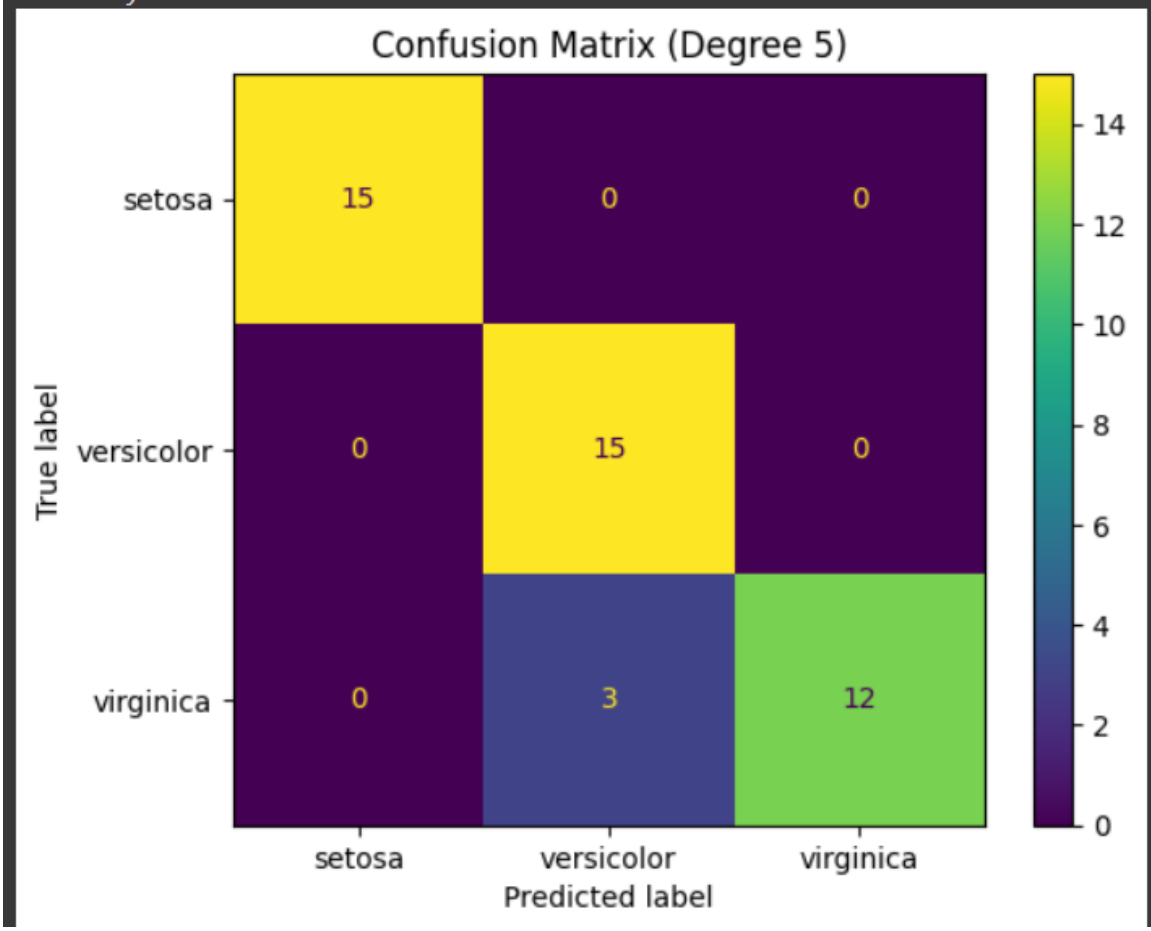
```
Degree: 3
Accuracy: 1.00
```



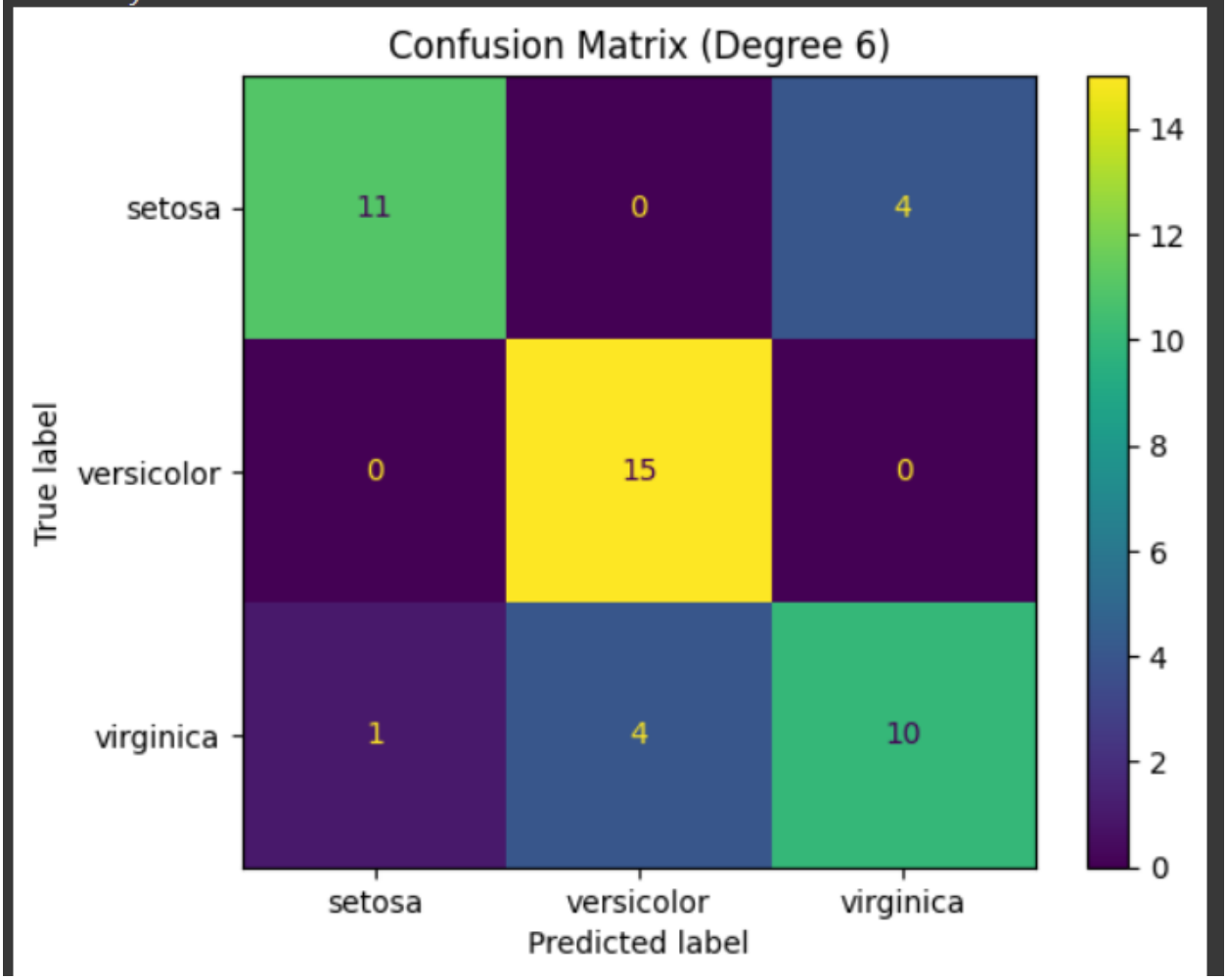
```
Degree: 4
Accuracy: 0.87
```



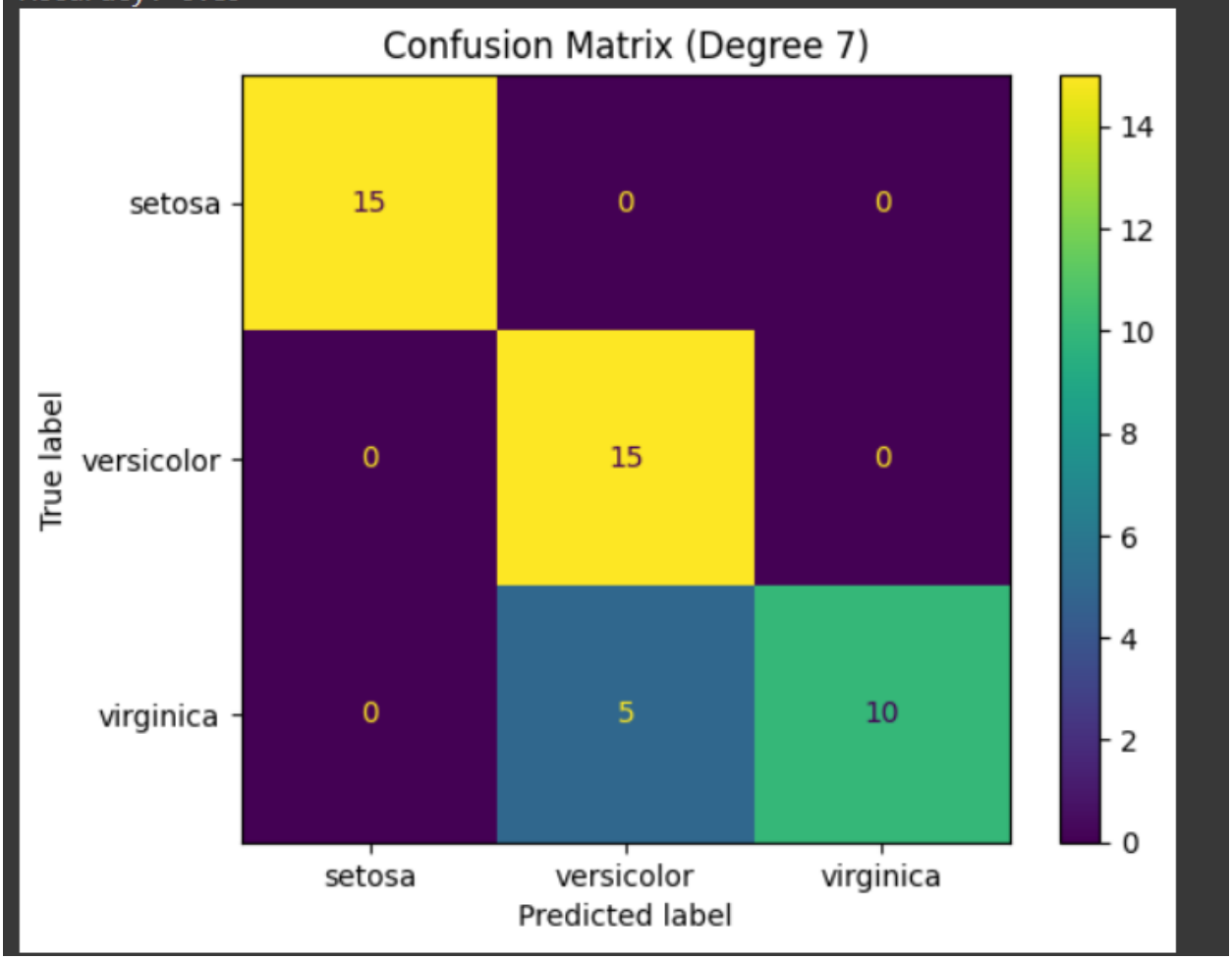
```
Degree: 5
Accuracy: 0.93
```



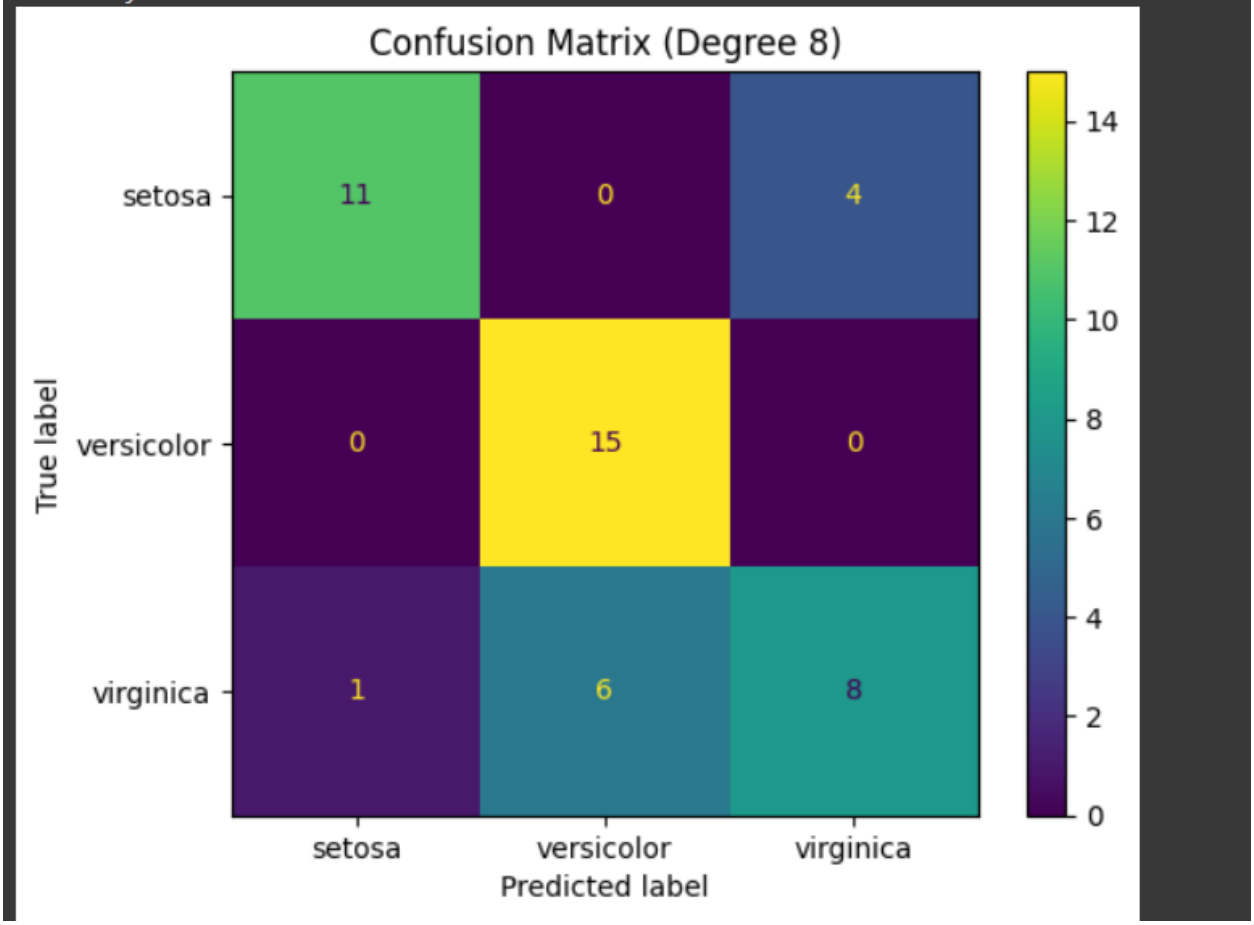
```
Degree: 6
Accuracy: 0.80
```



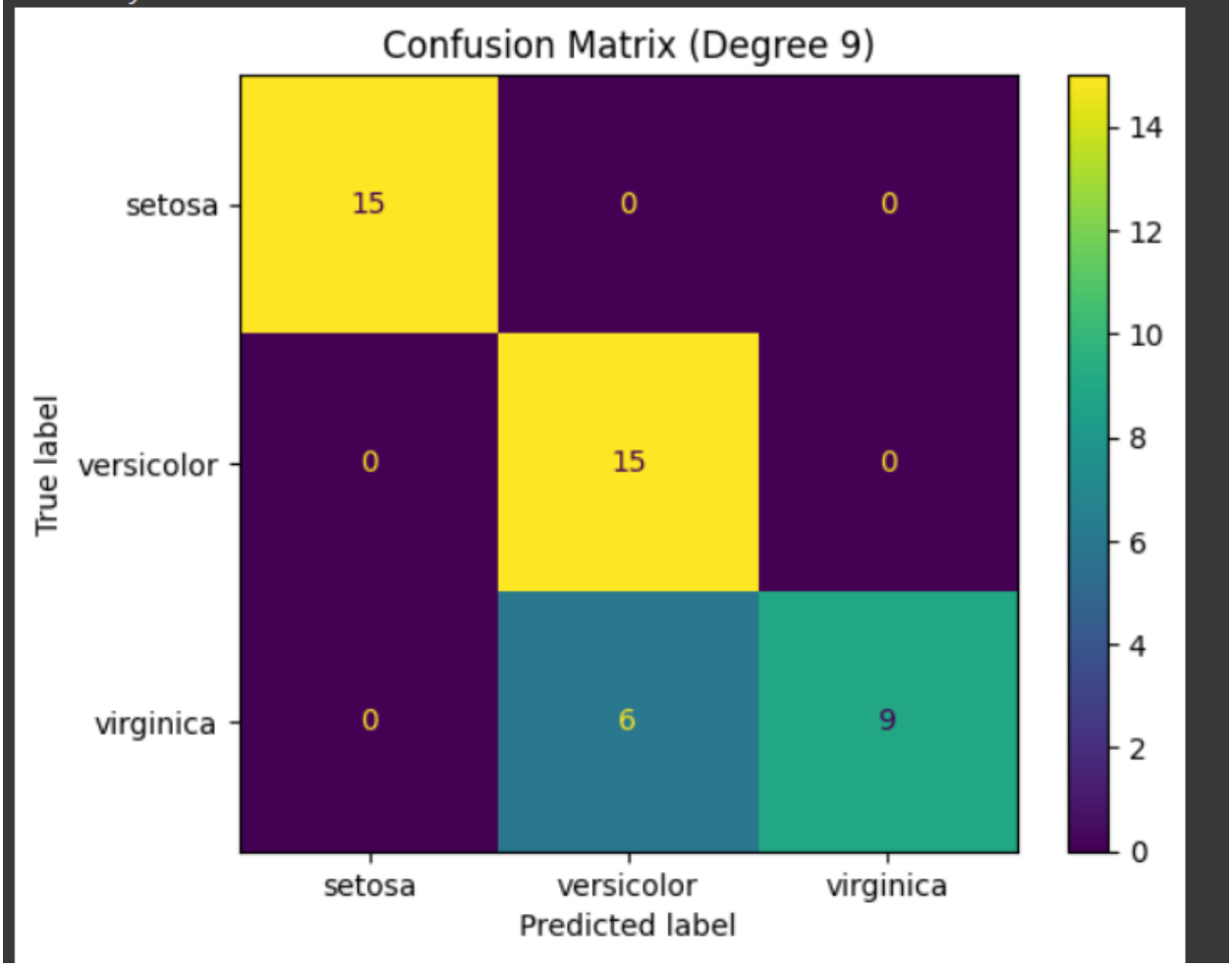
```
Degree: 7
Accuracy: 0.89
```

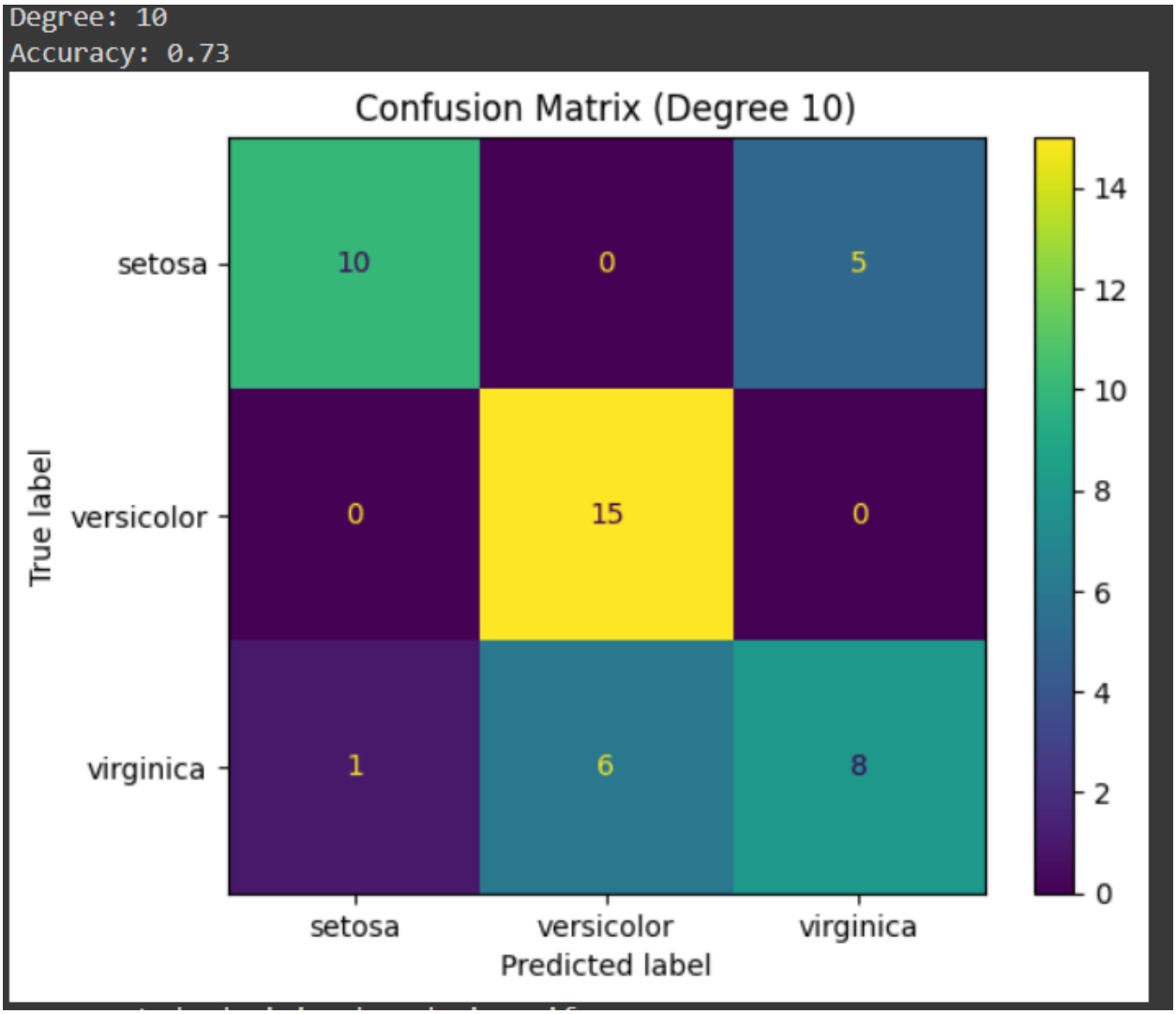


```
Degree: 8
Accuracy: 0.76
```



```
Degree: 9
Accuracy: 0.87
```





خوب همینجور که میبینید بهترین نتیجه برای مدل اول میباشد این به این دلیل است که مدل ما وقتی درجات آن بالا برود در واقع داده های آموزش را به خوبی میتواند یاد بگیرد یا به معنای دیگر در اینجا overfitting اتفاق میفتد.از آنجایی که داده های ما نیز کم میباشند پس نیازی به مدل پیچیده وجود ندارد. بد ترین نتیجه در واقع متعلق به درجه ۱۰ و بهترین نیز متعلق به درجه ۱ (که همان linear است) میباشد.

یکی از دلایل نوسانی بودن نتایج نیز میتواند کم بودن داده ها باشد همچنین تعداد ویژگی های ما نیز ۲ میباشد که داشتن درجه زیاد برای هسته چند جمله ای اشتباه است.

لینک gif (هم برای داده های تست و هم برای آموزش):

https://drive.google.com/file/d/1rUZDBCqJeFFkfimlSIeBeGTjDV5Akmmg/view?usp=sharing

د)

گزارش کد های هر بخش را در آخر سوال آورده ام ولی کد SVM خود را در اینجا توضیح میدهم :

```
class SVM(object):
    def __init__(self, kernel='polynomial', C=0, gamma=1, degree=3):
        self.C = float(C)
        self.gamma = float(gamma)
        self.degree = int(degree)
        self.kernel = kernel

    def polynomial_kernel(self, x, y, C=1, d=3):
        return (np.dot(x, y) + C) ** d

    def fit(self, X, y):
        n_samples, n_features = X.shape
        K = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                K[i, j] = self.polynomial_kernel(X[i], X[j], self.C,
self.degree)

        P = cvxopt.matrix(np.outer(y, y) * K + 1e+5 *
np.identity(n_samples))
        q = cvxopt.matrix(np.ones(n_samples) * -1)
        A = cvxopt.matrix(y.astype(np.double), (1, n_samples), tc='d')
        b = cvxopt.matrix(0.0)

        if self.C == 0:
            G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
            h = cvxopt.matrix(np.zeros(n_samples))
        else:
            tmp1 = np.diag(np.ones(n_samples) * -1)
            tmp2 = np.identity(n_samples)
            G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
            tmp1 = np.zeros(n_samples)
            tmp2 = np.ones(n_samples) * self.C
            h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

        cvxopt.solvers.options['show_progress'] = False
        cvxopt.solvers.options['abstol'] = 1e-10
        cvxopt.solvers.options['reltol'] = 1e-10
        cvxopt.solvers.options['feastol'] = 1e-10
```

```
        try:
            solution = cvxopt.solvers.qp(P, q, G, h, A, b)
        except ValueError as e:
            print("Solver failed due to rank deficiency.")
            return False

        alphas = np.ravel(solution['x'])

        sv = alphas > 1e-10
        ind = np.arange(len(alphas))[sv]
        self.alphas = alphas[sv]
        self.sv = X[sv]
        self.sv_y = y[sv]

        if len(self.alphas) > 0:
            self.b = 0
            for n in range(len(self.alphas)):
                self.b += self.sv_y[n]
                self.b -= np.sum(self.alphas * self.sv_y * K[ind[n], sv])
            self.b = self.b / len(self.alphas)
        else:
            self.b = 0

        if self.kernel == 'linear':
            self.w = np.zeros(n_features)
            for n in range(len(self.alphas)):
                self.w += self.alphas[n] * self.sv_y[n] * self.sv[n]
        else:
            self.w = None
        return True

    def project(self, X):
        if self.w is not None:
            return np.dot(X, self.w) + self.b
        else:
            y_predict = np.zeros(len(X))
            for i in range(len(X)):
                s = 0
                for a, sv_y, sv in zip(self.alphas, self.sv_y, self.sv):
                    s += a * sv_y * self.polynomial_kernel(X[i], sv,
self.C, self.degree)
                y_predict[i] = s
            return y_predict + self.b
```

```
    def predict(self, X):
        return np.sign(self.project(X))
```

class SVM(object): کلاسی به نام SVM که مخفف Support Vector Machine است را تعریف می کند.

__init__(self, kernel='polynomial', C=0, gamma=1, grade=3): متد سازنده شی SVM را با پارامترهای مشخص شده مقداردهی اولیه می کند:

kernel: نوع تابع هسته، پیش فرض چند جمله ای است.

C: پارامتر Regularization.

gamma: ضریب برای تابع هسته.

degree: درجه هسته چند جمله ای.

self.C، self.gamma، self.degree، self.kernel: پارامترهای ورودی را به عنوان متغیرهای نمونه ذخیره کنید.

polynomial_kernel(self, x, y, C=1, d=3): تابع هسته چند جمله ای را تعریف می کند.

x، y: بردارهای ورودی.

C, d: پارامترهای هسته چند جمله ای.

return (np.dot(x, y) + C) ** d: مقدار هسته چند جمله ای را محاسبه و برمی گرداند.

Fitting the Model :

fit(self, X, y): روش برازش مدل SVM.

X: ماتریس ویژگی.

y: بردار هدف.

n_samples, n_features = X.shape: تعداد نمونه ها و ویژگی ها را بدست آورید.

K = np.zeros((n_samples, n_samples)): ماتریس کرنل K را راه اندازی کنید.

برای i در محدوده(n_samples): برای محاسبه مقادیر هسته هر نمونه را حلقه بزنید.

K[i, j] = self.polynomial_kernel(X[i], X[j], self.C, self.degree): مقدار هسته را برای هر جفت نمونه محاسبه کرده و در K ذخیره کنید.

P = cvxopt.matrix(np.outer(y,y) * K + 1e+5 * np.identity(n_samples)): ماتریس P را برای مسئله برنامه نویسی درجه دوم ایجاد میکنیم.

q = cvxopt.matrix(np.ones(n_samples) * -1): بردار q را ایجاد میکنیم.

A = cvxopt.matrix(y.atype(np.double)، 1)، (n_samples، tc='d'): ماتریس A را ایجاد میکنیم.

b = cvxopt.matrix(0.0): اسکالر b را ایجاد میکنیم.

if self.C == 0:: بررسی میکنیم که آیا پارامتر تنظیم C 0 است (حاشیه سخت SVM).

G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1)): ماتریس G را ایجاد میکنیم.

h = cvxopt.matrix(np.zeros(n_samples)): بردار h را ایجاد کنید.

else:: برای C غیر صفر (SVM حاشیه نرم).

tmp1 = np.diag(np.ones(n_samples) * -1): ایجاد ماتریس موقت tmp1.

tmp2 = np.identity(n_samples): ایجاد ماتریس موقت tmp2.

tmp1 : G = cvxopt.matrix(np.vstack((tmp1, tmp2))) و tmp2 را به صورت عمودی برای ایجاد G وصل میکنیم.

tmp1 = np.zeros(n_samples): بردار موقت tmp1 ایجاد میکنیم.

tmp2 = np.ones(n_samples) * self.C: بردار موقت tmp2 ایجاد میکنیم.

tmp1 : h = cvxopt.matrix(np.hstack((tmp1, tmp2))) و tmp2 را به صورت افقی برای ایجاد h وصل میکنیم.

cvxopt.solvers.options[...]: گزینه هایی را برای حل کننده cvxopt تنظیم کنید تا دقت و نمایش فرآیند بهینه سازی را کنترل کند.

try:: تلاش برای حل مسئله برنامه نویسی درجه دوم.

solution = cvxopt.solvers.qp(P, q, G, h, A, b): حل مسئله QP.

به جز ValueError به عنوان e:: هر گونه خطا را در حین حل بررسی میکنیم.

print print("Solver failed due to rank deficiency."): اگر حل کننده با مشکل مواجه شود، پیغام خطای چاپ.

return False: Return False نشان دهنده شکست است.

alphas = np.ravel(solution['x']): آلفا ها را مسطح میکند.

sv = alphas > 1e-5: بردارهای پشتیبانی را شناسایی میکنیم (که در آن آلفاها از یک آستانه بزرگتر هستند).

ind = np.arange(len(alphas))[sv]: شاخص های بردارهای پشتیبان را بدست می آوریم.

self.alphas = alphas[sv]: ذخیره ضرایب بردار پشتیبانی.

self.sv = X[sv]: ذخیره بردارهای پشتیبانی.

self.sv_y = y[sv]: برچسب های بردارهای پشتیبانی را ذخیره میکنیم.

if len(self.alphas) > 0:: بررسی میکنیم که آیا بردارهای پشتیبانی وجود دارد یا خیر.

self.b = 0: اصطلاح سوگیری را آغاز میکنیم.

برای n در محدوده(len(self.alphas)):: از میان بردارهای پشتیبانی حلقه میزنیم.

self.b += self.sv_y[n]: برچسب بردار پشتیبانی را اضافه میکنیم.

self.b -= np.sum(self.alphas * self.sv_y * K[ind[n], sv]): از مجموع مقادیر کرنل وزن شده با آلفاها و برچسب ها کم میکنیم.

self.b = self.b / len(self.alphas): میانگین تعصب را محاسبه میکنیم.

else:: اگر هیچ بردار پشتیبانی یافت نشد.

self.b = 0: عبارت bias را روی 0 قرار میدهیم.

if self.kernel == 'linear':: بررسی میکنیم که آیا هسته خطی است یا خیر.

self.w = np.zeros(n_features): بردار وزن w را مقداردهی میکنیم.

برای n در محدوده(len(self.alphas)): از میان بردارهای پشتیبانی حلقه میزنیم.

self.w += self.alphas[n] * self.sv_y[n] * self.sv[n]: بردار وزن را به روز میکنیم.

else: برای هسته های غیر خطی.

self.w = None: بردار وزن را روی None قرار میدهیم.

بازگشت True: Return True نشان دهنده برازش موفقیت آمیز است.

Making Predictions :

Project(self,X): روشی برای پروژه داده های ورودی X بر روی مرز تصمیم.

if self.w is not None: بررسی میکنیم که آیا بردار وزن w تعریف شده است یا خیر.

return np.dot(X, self.w) + self.b: تابع تصمیم خطی را محاسبه میکنیم.

else: برای هسته های غیر خطی.

y_predict = np.zeros(len(X)): آرایه پیش بینی را مقداردهی میکنیم.

برای i در محدوده(len(X)): از طریق هر نمونه ورودی حلقه میزنیم.

s = 0: مقدار اولیه s.

برای a, sv_y, sv در zip(self.alphas, self.sv_y, self.sv): حلقه از میان بردارهای پشتیبانی.

s += a * sv_y * self.polynomial_kernel(X[i], sv, self.C, self.degree): مجموع را با مقدار کرنل به روز میکنیم.

y_predict[i] = s: جمع را در آرایه پیش بینی ذخیره میکنیم.

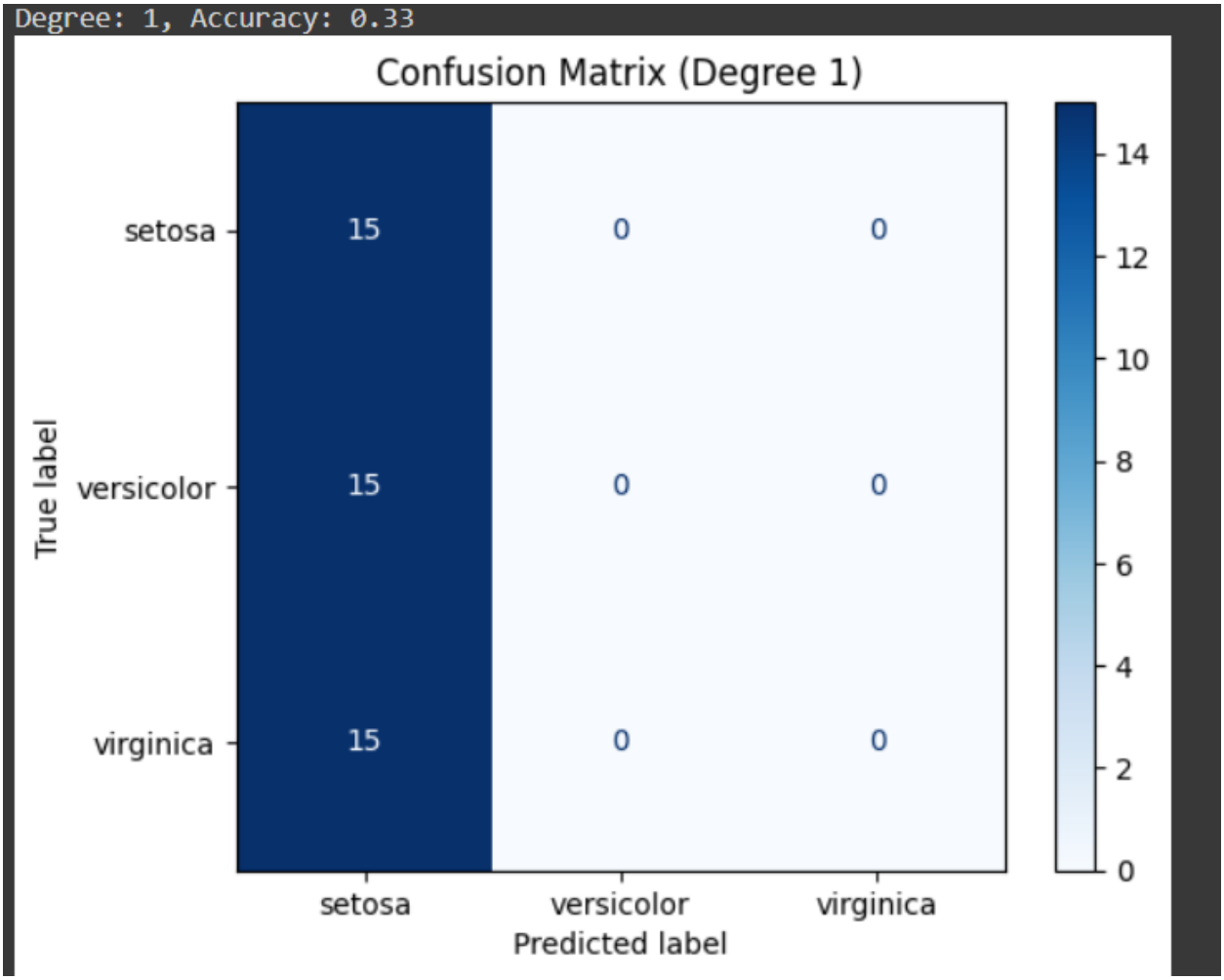
return y_predict + self.b: آرایه پیش بینی نهایی را برمی گرداند.

Predict(self,X): روشی برای پیش بینی داده های ورودی X.

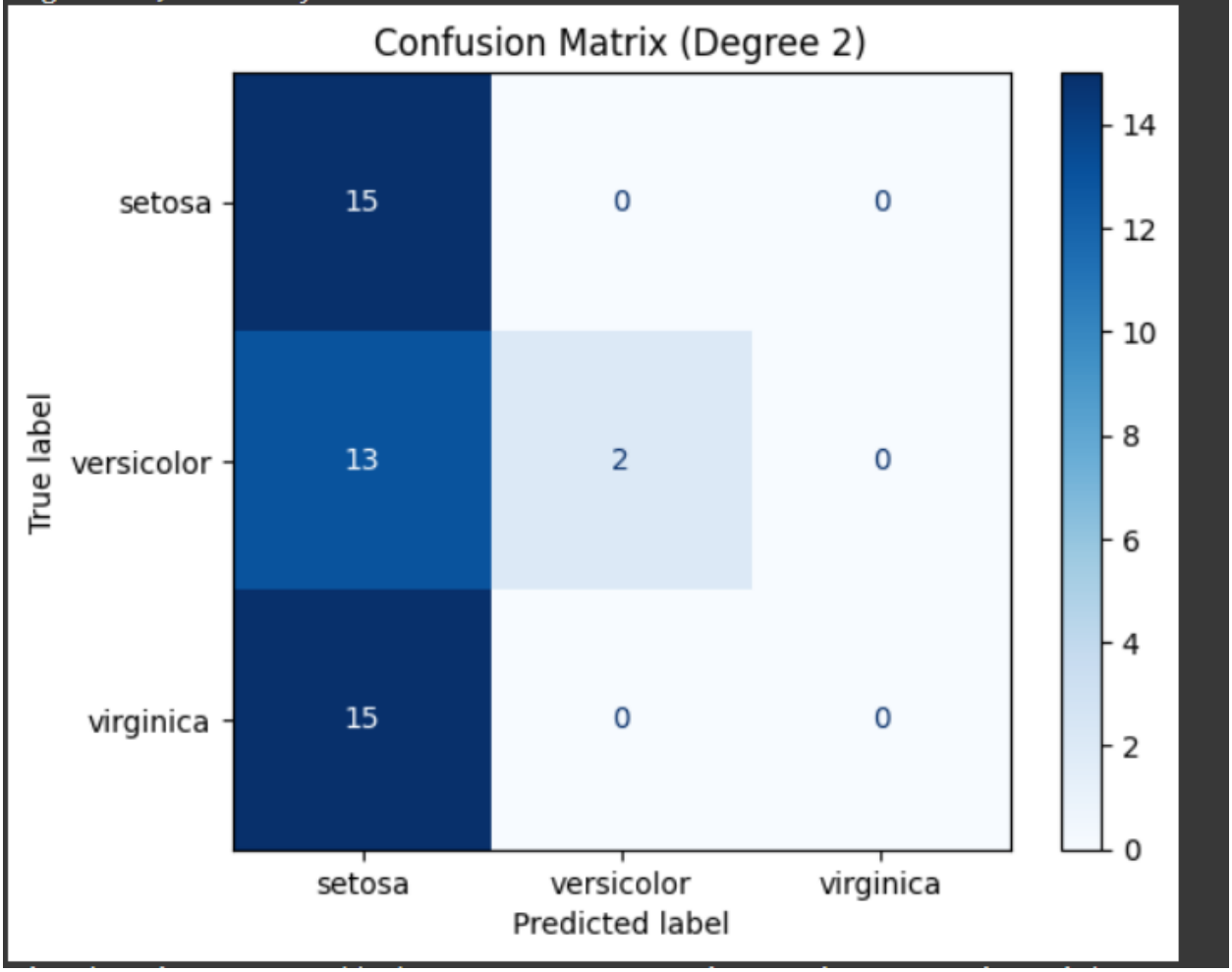
return np.sign(self.project(X)): علامت مقادیر پیش بینی شده (یعنی برچسب های کلاس) را برمیگرداند.

نتایج :

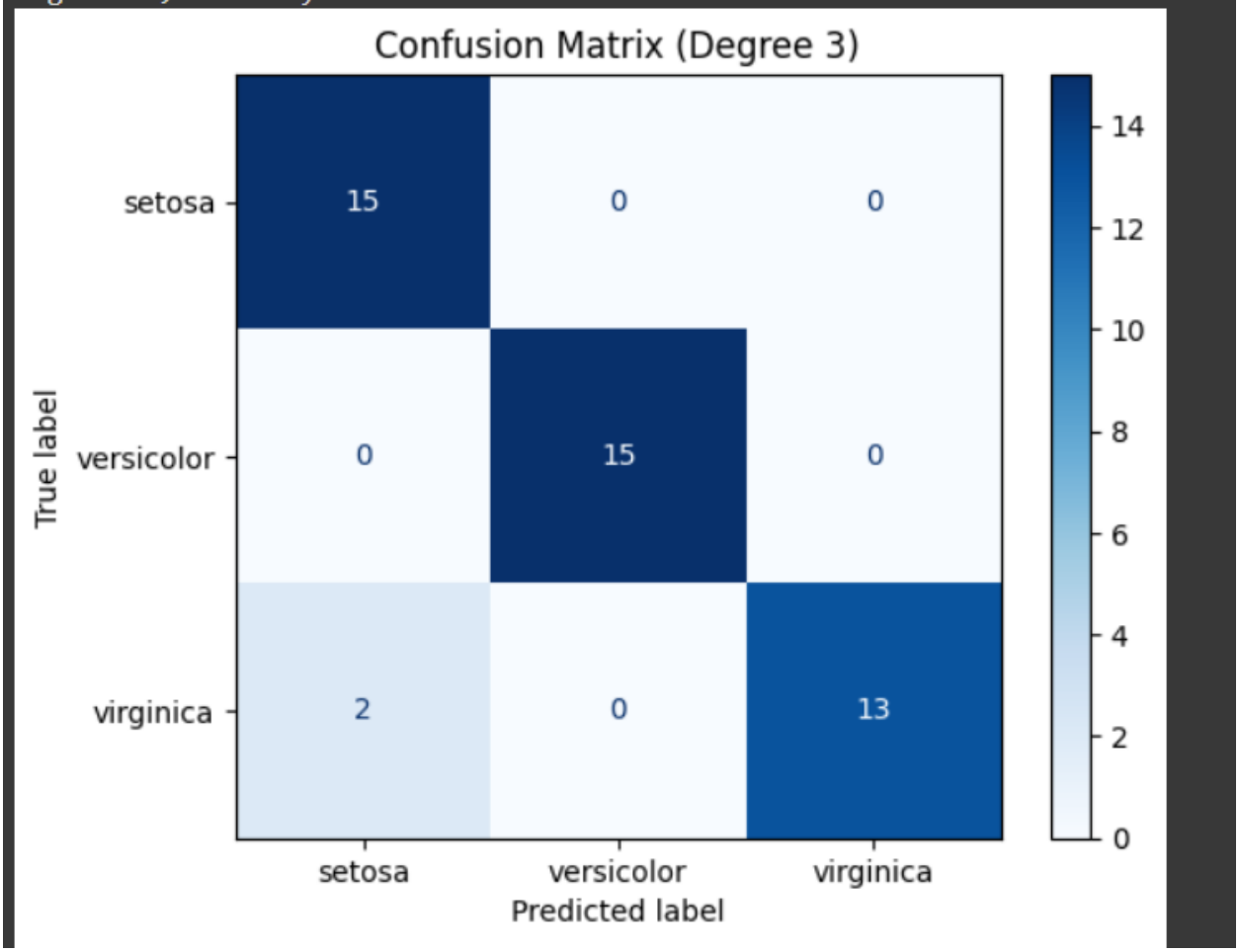
ما در این قسمت نیز از LDA برای کاهش ابعاد استفاده کردیم بعد از آموزش و تست , confusion matrix و دقت مدل را نمایش دادیم و در آخر gif را درست کردیم که نتایج بدین شکل شد :



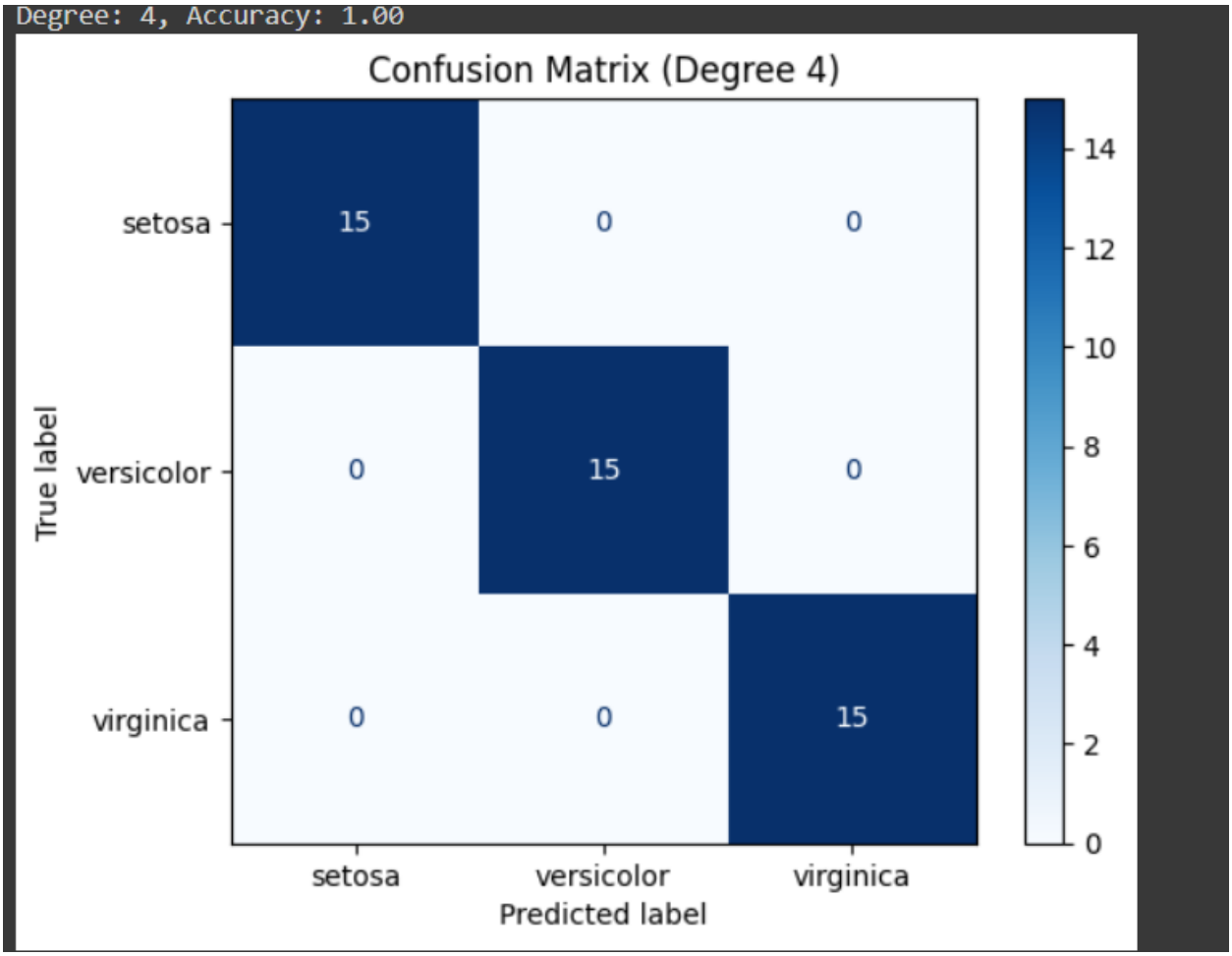
```
Degree: 2, Accuracy: 0.38
```



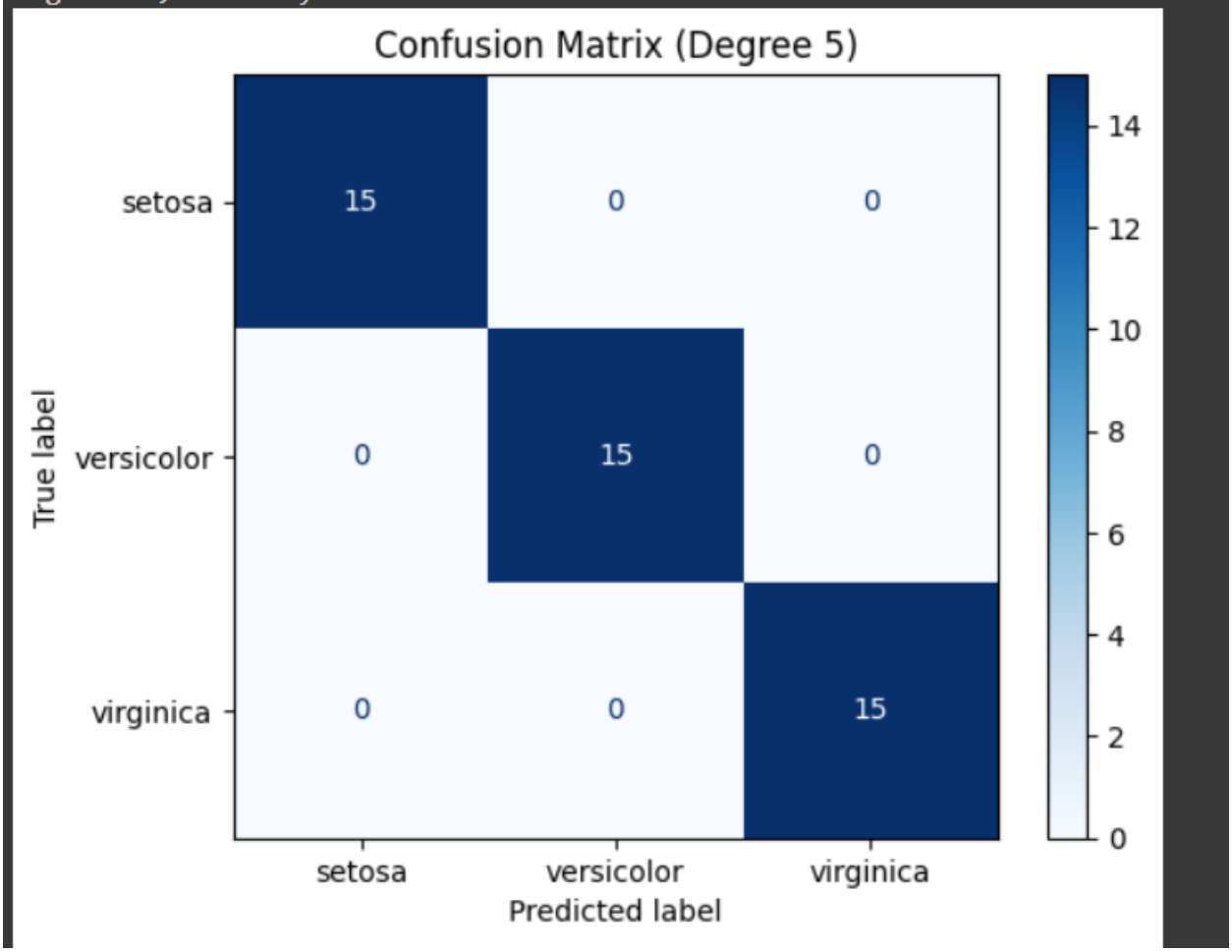
```
Degree: 3, Accuracy: 0.96
```



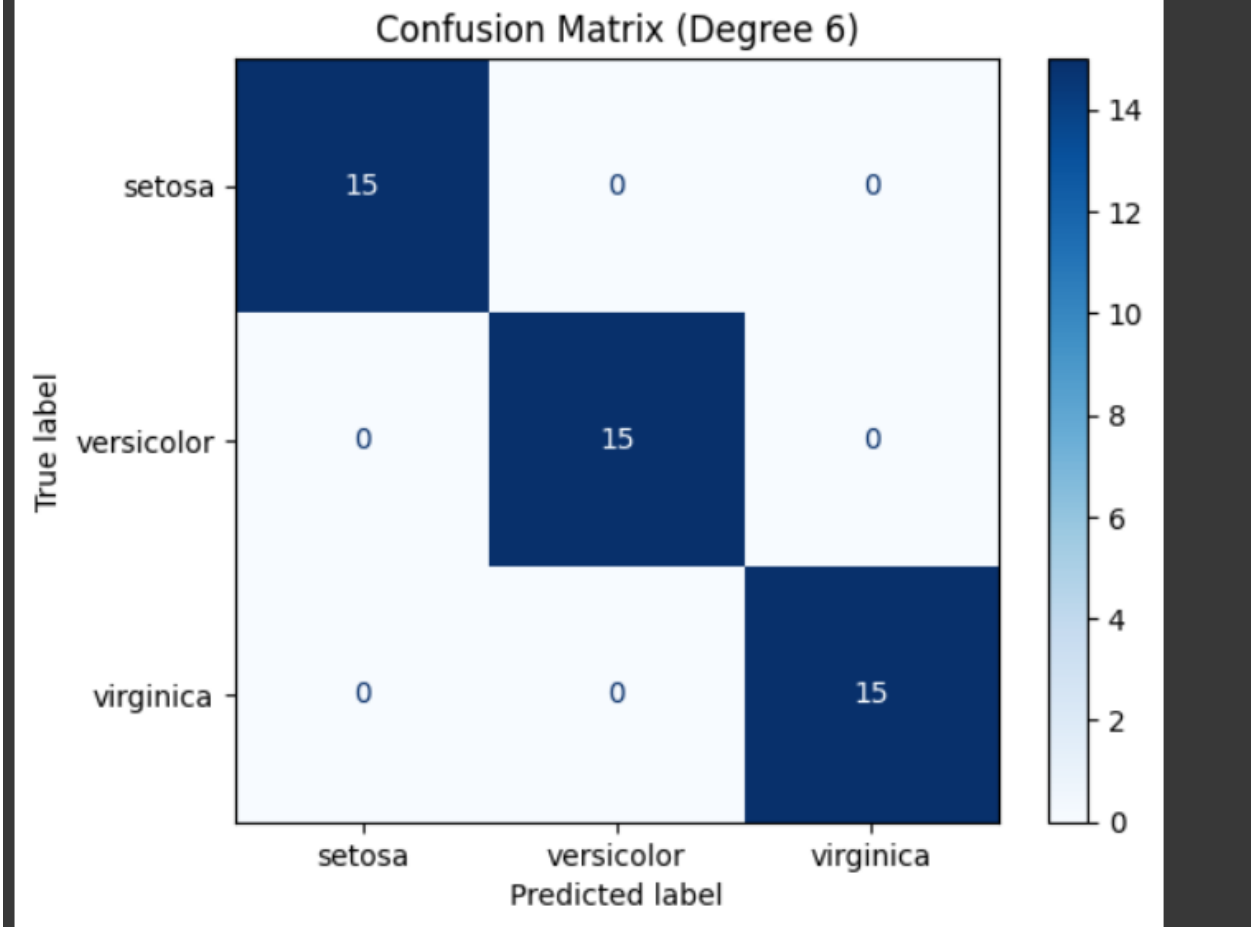
```
Degree: 4, Accuracy: 1.00
```



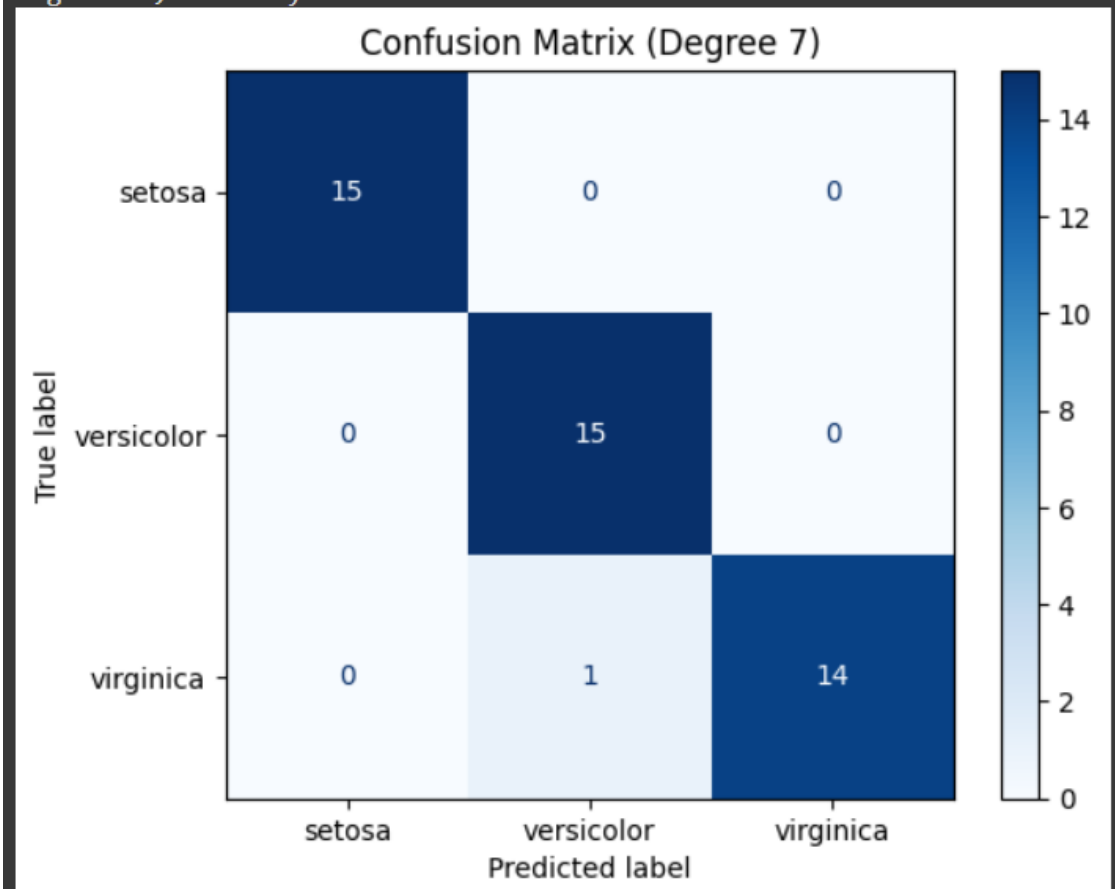
```
Degree: 5, Accuracy: 1.00
```



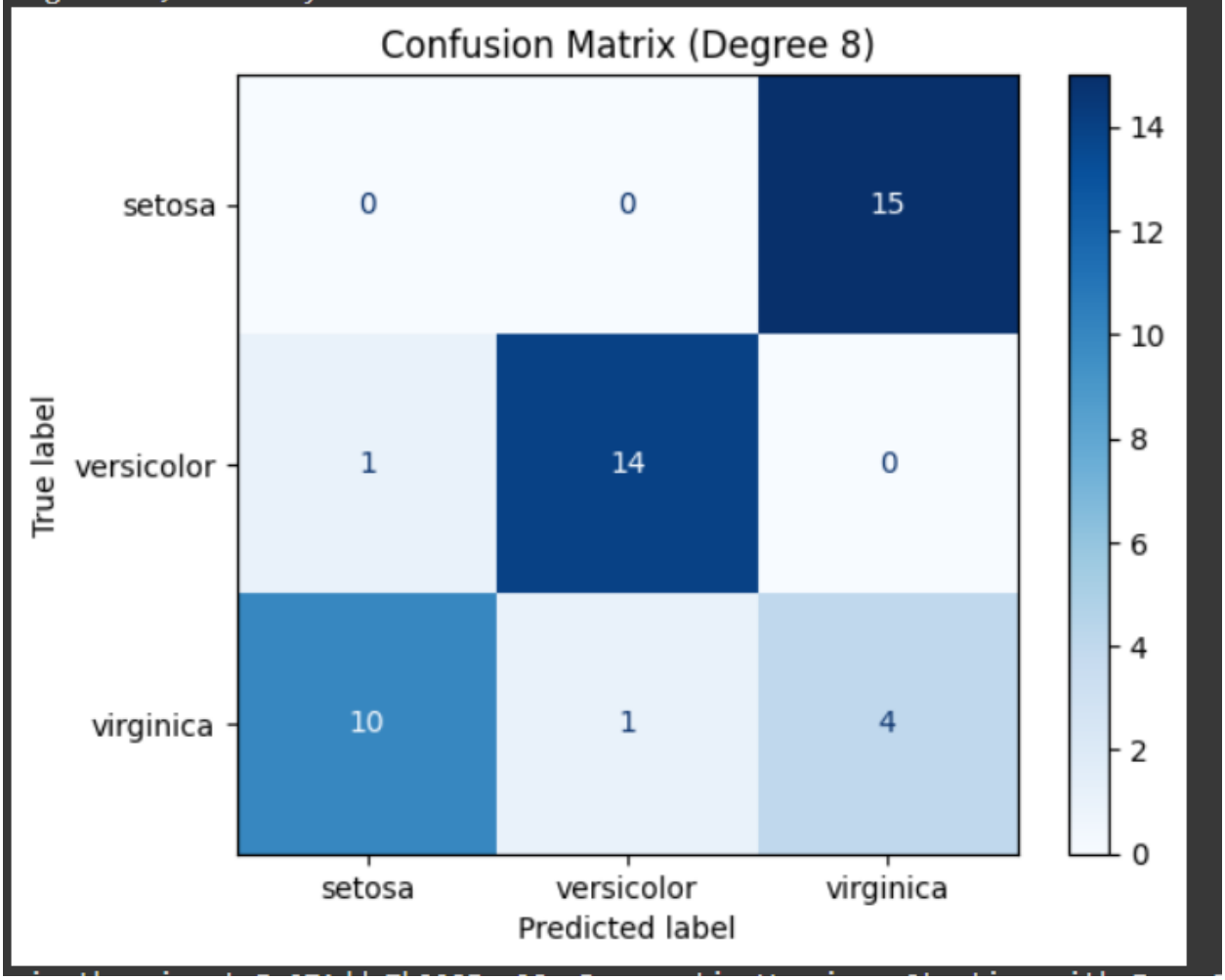
```
Degree: 6, Accuracy: 1.00
```



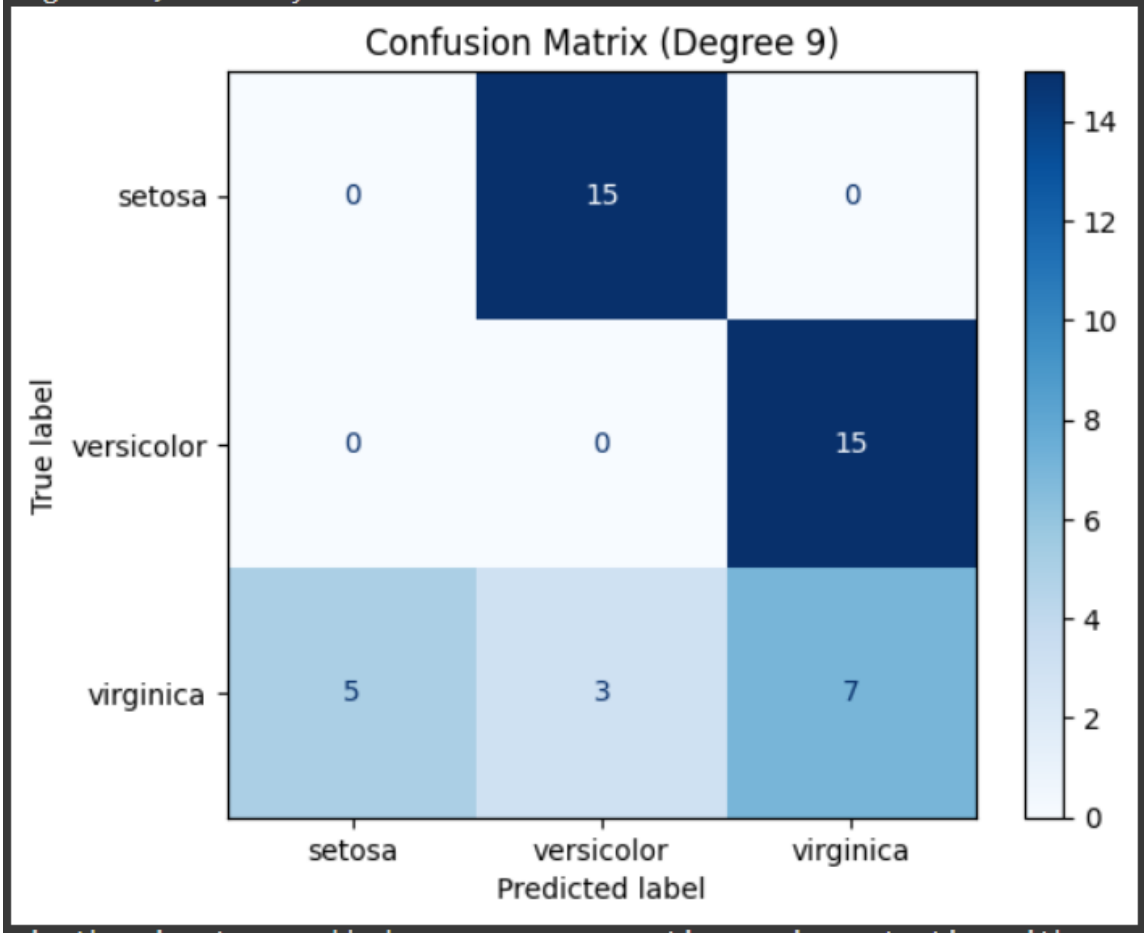
Degree: 7, Accuracy: 0.98

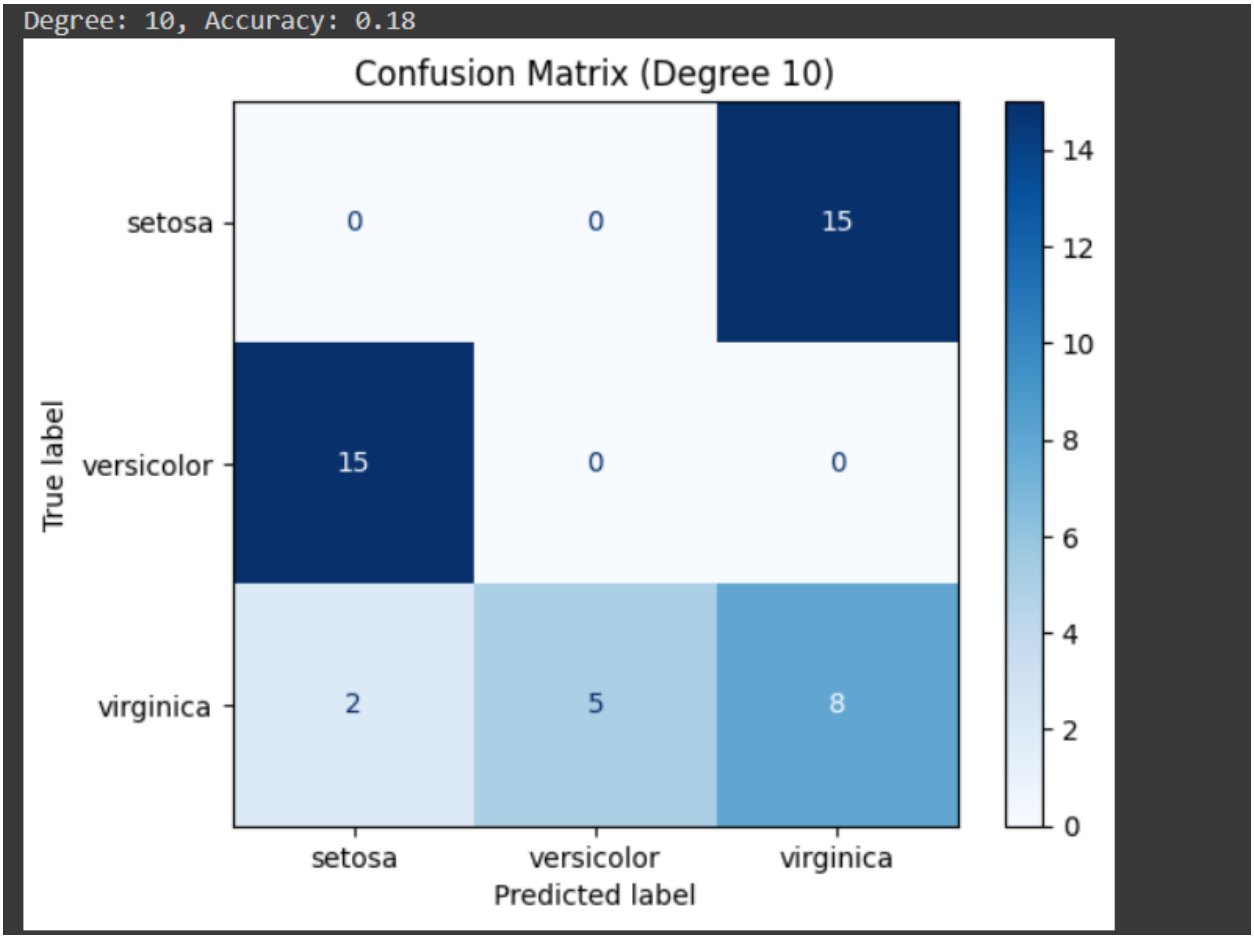


Degree: 8, Accuracy: 0.40



Degree: 9, Accuracy: 0.16





خوب همینطور که میبنید دقت مدل دستی ما با مدل سایکیت تفاوت زیادی دارد در واقع در سایکیت مدل ما در درجه ۱ دقت آن ۱۰۰ درصد بود ولی در اینجا ۳۳ درصد میباشد با افزایش درجه دقت مدل بهتر شده تا به درجه ۸ رسیدیم همانطور که معلوم است دقت مدل نسبت به درجه قبلی خیلی کم تر شده که نشان دهنده overfitting میباشد.

چالش اصلی در اینجا این بود که مدل نمیتوانست وزن های w را تشکیل دهد زیرا نمیتوانست Q.P. را حل کند به همین منظور من ماترس P را بزرگ تر کردم تا بتواند درجات بالا را تشکیل دهد:

```
P = cvxopt.matrix(np.outer(y, y) * K + 1e+5 * np.identity(n_samples))
```

لینک GIF(هم برای داده های تست و هم برای آموزش) :

https://drive.google.com/file/d/1tS85CD0ZjGCCeA3eWgbHBhiDsjJSNgyR/view?usp=sharing

گزارش کد : آ)

```
# Load the iris dataset
iris = load_iris()
data = iris.data
target = iris.target
feature_names = iris.feature_names
target_names = iris.target_names

# Convert to DataFrame for easier analysis
df = pd.DataFrame(data, columns=feature_names)
df['target'] = target
df
```

load_iris: مجموعه داده Iris را بارگذاری می کند، مجموعه داده معروفی برای یادگیری ماشین.

data: شامل ویژگی ها (طول کاسبرگ، عرض کاسبرگ، طول گلبرگ، عرض گلبرگ).

target: حاوی برچسب های هدف (گونه های عنبیه).

feature_names: نام ویژگی ها.

target_names: نام طبقات (گونه) هدف.

pd.DataFrame: داده ها را برای دستکاری و تجزیه و تحلیل آسان تر به DataFrame پاندا تبدیل می کند.

df['target']: برچسب های هدف را به DataFrame اضافه می کند.

```
df['target'].value_counts()
# Basic information
dimensions = df.shape
num_samples = len(df)

dimensions, num_samples

# Descriptive statistics: mean and variance
means = df.mean()
variances = df.var()

print("\nMean of features:\n", means)
print("\nVariance of features:\n", variances)
```

df.shape: ابعاد DataFrame را نمایش می دهد.

df['target'].value_counts(): تعداد نمونه ها را برای هر کلاس هدف شمارش می کند.

len(df): تعداد کل نمونه ها.

df.mean(): میانگین هر ویژگی را محاسبه می کند.

df.var(): واریانس هر ویژگی را محاسبه می کند.

```
ax = sns.pairplot(df , hue = 'target')
sns.move_legend(
    ax,"lower center",
    bbox_to_anchor=(.5,1) , ncol=3,title="Pair Plot of Iris
dataset",frameon = False
)
plt.tight_layout()
plt.show()
# Correlation matrix
correlation_matrix = df.corr()

correlation_matrix
```

sns.pairplot: طرح‌های زوجی از تمام ترکیب‌های ویژگی ایجاد می‌کند که با کلاس هدف رنگ‌بندی می‌شوند.

sns.move_legend: افسانه را به مرکز پایین طرح منتقل می کند.

plt.tight_layout(): طرح را برای جلوگیری از همپوشانی تنظیم می کند.

df.corr(): ماتریس همبستگی ویژگی ها را محاسبه می کند.

```
# Perform t-SNE
pca = PCA(n_components=2, random_state=64)
pca_result = pca.fit_transform(data)

# Plot t-SNE result
plt.figure(figsize=(10, 7))
scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1], c=target,
cmap='viridis', marker='o')
plt.colorbar(scatter, ticks=range(len(target_names)), label='Classes')
plt.title('PCA visualization of IRIS dataset')
plt.xlabel('1st Eigenvector')
plt.ylabel('2nd Eigenvector')
plt.show()
```

```
fig = plt.figure(1, figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d", elev=-150, azim=110)

X_reduced = PCA(n_components=3 , random_state =
64).fit_transform(iris.data)
ax.scatter(
    X_reduced[:, 0],
    X_reduced[:, 1],
    X_reduced[:, 2],
    c=iris.target,
    s=40,
)

ax.set_title("First three PCA dimensions")
ax.set_xlabel("1st Eigenvector")
ax.xaxis.set_ticklabels([])
ax.set_ylabel("2nd Eigenvector")
ax.yaxis.set_ticklabels([])
ax.set_zlabel("3rd Eigenvector")
ax.zaxis.set_ticklabels([])

plt.show()
```

PCA: تجزیه و تحلیل اجزای اصلی را برای کاهش ابعاد به 2 جزء انجام می دهد.

plt.scatter: دو جزء اصلی اول را با رنگ هایی که کلاس های هدف را نشان می دهند ترسیم می کند.

plt.colorbar: یک نوار رنگی به طرح اضافه می کند.

fig.add_subplot: یک طرح فرعی سه بعدی به شکل اضافه می کند.

PCA(n_components=3): داده ها را به 3 جزء کاهش می دهد.

ax.scatter: یک نمودار پراکندگی سه بعدی از سه جزء اصلی اول ایجاد می کند.

ax.set_title: عنوان طرح را تعیین می کند.

ax.set_xlabel, ax.set_ylabel, ax.set_zlabel: محورها را برچسب گذاری می کند.

ax.xaxis.set_ticklabels([]): برچسب های تیک را از محورها حذف می کند.

کد بخش ب :

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=0, stratify=target)

scalar = StandardScaler()
scalar.fit_transform(X_train)
scalar.fit(X_test)

# Reduce dimensionality using LDA on the training data
lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)

# Train SVM classifier with a linear kernel
svm = SVC(kernel='linear')
svm.fit(X_train_lda, y_train)

# Predict using the trained classifier
y_test_pred = svm.predict(X_test_lda)

# Obtain and print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_test_pred)
cmd = ConfusionMatrixDisplay(conf_matrix, display_labels=target_names)
cmd.plot()
plt.title('Confusion Matrix for SVM with Linear Kernel')
plt.show()

print(classification_report(y_test, y_test_pred,
target_names=target_names))


# Function to plot decision boundaries
def plot_decision_boundaries(X, y, model, title='Decision Boundaries'):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
    scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis',
edgecolor='k', s=20)
```

```
    plt.colorbar(scatter, ticks=range(len(target_names)), label='Classes')
    plt.title(title)
    plt.xlabel('LDA Component 1')
    plt.ylabel('LDA Component 2')
    plt.show()

# Draw decision boundaries on the training data
plot_decision_boundaries(X_train_lda, y_train, svm, title='Decision
Boundaries for SVM with Linear Kernel (Training Data)')

# Draw decision boundaries on the test data
plot_decision_boundaries(X_test_lda, y_test, svm, title='Decision
Boundaries for SVM with Linear Kernel (Test Data)')
```

train_test_split: این تابع مجموعه داده را به مجموعه های آموزشی و آزمایشی تقسیم می کند.

data: ویژگی های مجموعه داده.

target: برچسب های مجموعه داده.

test_size=0.3: 30 درصد از داده ها برای تست، 70 درصد برای آموزش اختصاص داده شده است.

random_state=0: تکرارپذیری تقسیم را تضمین می کند.

stratify=target: اطمینان حاصل می کند که نسبت کلاس ها در مجموعه های آموزشی و آزمایشی مانند مجموعه داده اصلی است.

StandardScaler: ویژگی ها را با حذف میانگین و مقیاس بندی به واریانس واحد استاندارد می کند.

fit_transform(X_train): مقیاس‌کننده را با داده‌های آموزشی متناسب می‌کند و آن را تبدیل می‌کند.

fit(X_test): مقیاس‌کننده را با داده‌های تست منطبق می‌کند (برای جلوگیری از نشت داده‌ها باید به جای تناسب تبدیل شود).

LDA (تحلیل تشخیص خطی): ابعاد داده ها را کاهش می دهد و در عین حال اطلاعات تبعیض آمیز طبقاتی را تا حد امکان حفظ می کند.

n_components=2: داده ها را به 2 بعد کاهش می دهد.

LDA: fit_transform(X_train, y_train) را با داده های آموزشی متناسب می کند و آن را تبدیل می کند.

transform(X_test): داده های تست را با استفاده از مدل LDA از قبل برازش شده تبدیل می کند.

SVC: پشتیبانی از دسته بندی بردار از sklearn.svm.

kernel='linear': از یک هسته خطی برای SVM استفاده می کند.

fit(X_train_lda، (y_train: مدل SVM را با استفاده از داده های آموزشی تبدیل شده توسط LDA آموزش می دهد.

predict(X_test_lda): از مدل SVM آموزش‌دیده برای پیش‌بینی برچسب‌های داده‌های آزمون تبدیل شده توسط LDA استفاده می‌کند.

confusion_matrix(y_test, y_test_pred): ماتریس سردرگمی را برای ارزیابی دقت طبقه بندی محاسبه می کند.

ConfusionMatrixDisplay: ماتریس سردرگمی را تجسم می کند.

classification_report: گزارشی را ایجاد می کند که معیارهای طبقه بندی اصلی را نشان می دهد.

plot_decision_boundaries: تابعی برای رسم مرزهای تصمیم مدل SVM آموزش دیده.

X, y: داده ها و برچسب ها برای رسم.

مدل: مدل SVM آموزش دیده.

عنوان: عنوان طرح.

meshgrid: شبکه ای از مقادیر را در فضای ویژگی ایجاد می کند.

predict: برچسب ها را برای هر نقطه از شبکه پیش بینی می کند.

contourf: مرزهای تصمیم را ترسیم می کند.

scatter: نقاط داده واقعی را رسم می کند.

کد بخش د :

کد بخش (ج) مانند بخش (د) هست ولی از آنجایی که این بخش بیشتر کار برده همین بخش را توضیح میدهیم:

```
def plotSVC(X, y, models, degree, dataset_type='Training'):
    # create a mesh to plot in
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    h = (x_max - x_min)/100
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
    plt.subplot(1, 1, 1)

    Z = np.zeros((xx.ravel().shape[0], len(models)))
    for i, model in enumerate(models):
        Z[:, i] = model.predict(np.c_[xx.ravel(), yy.ravel()])

    Z = np.argmax(Z, axis=1)
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
    scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis',
edgecolor='k', s=20)
    plt.colorbar(scatter, ticks=range(len(target_names)), label='Classes')
    plt.title(f'Decision Boundaries for SVM with Polynomial Kernel (Degree
{degree}) ({dataset_type} Data)')
    plt.xlabel('LDA Component 1')
    plt.ylabel('LDA Component 2')
    filename = f'decision_boundary_degree_{degree}_{dataset_type}.png'
    plt.savefig(filename)
    plt.close()
    return filename
```

تنها فرق این decision با قبلی در این قسمت کد است :

```
h = (x_max - x_min)/100
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))
    plt.subplot(1, 1, 1)
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, target,
test_size=0.3, random_state=0, stratify=target)

# Reduce dimensionality using LDA on the training data
lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)


# Prepare to store results and images for the GIF
results = []
images = []


# One-vs-rest classification using polynomial kernels of degrees 1 to 10
for degree in range(1, 11):
    svm_models = []

    for i in range(len(target_names)):
        y_train_binary = np.where(y_train == i, 1, -1)
        svm = SVM(kernel='polynomial', degree=degree, C=1.0)
        svm.fit(X_train_lda, y_train_binary)
        svm_models.append(svm)


    # Plot and save decision boundaries for the training data
    filename = plotSVC(X_train_lda, y_train, svm_models, degree,
'Training')
    images.append(imageio.imread(filename))
    # Plot and save decision boundaries for the testing data
    filename = plotSVC(X_test_lda, y_test, svm_models, degree, 'Testing')
    images.append(imageio.imread(filename))

    y_test_pred = np.zeros(len(y_test))

    for i in range(len(target_names)):
        y_test_pred_binary = svm_models[i].predict(X_test_lda)
        y_test_pred[y_test_pred_binary == 1] = i

    # Compute accuracy
    accuracy = accuracy_score(y_test, y_test_pred)
    print(f'Degree: {degree}, Accuracy: {accuracy:.2f}')
    # Plot the confusion matrix
```

```
    cm = confusion_matrix(y_test, y_test_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=target_names)
    disp.plot(cmap=plt.cm.Blues)
    plt.title(f'Confusion Matrix (Degree {degree})')
    plt.show()

# Save the images as a GIF
gif_filename = 'decision_boundaries.gif'
imageio.mimsave(gif_filename, images, duration=10)

# Print the GIF filename
print(f'GIF saved as {gif_filename}')
```

:One-vs-Rest Classification with Polynomial Kernels

حلقه های چند جمله ای از 1 تا 10 را طی می کند.

یک لیست خالی را برای ذخیره مدل های SVM برای هر کلاس راه اندازی می کند.

One-vs-Rest Strategy: یک SVM جداگانه برای هر کلاس آموزش می دهد.

y_train_binary: برچسب های چند کلاسه را به برچسب های باینری تبدیل می کند (1 برای کلاس فعلی، -1 برای بقیه).

SVM(kernel='polynomial', grade=degree, C=1.0): SVM را با هسته چند جمله ای درجه فعلی راه اندازی می کند.

fit(X_train_lda، y_train_binary): SVM را بر روی داده های آموزشی تبدیل شده توسط LDA آموزش می دهد.

svm_models.append(svm): مدل SVM آموزش دیده را به لیست اضافه می کند.

plotSVC: تابعی برای ترسیم مرزهای تصمیم گیری برای مدل های SVM (فرض می شود در جای دیگری تعریف شود).

imageio.imread (نام فایل): تصویر نمودار ذخیره شده را می خواند.

images.append: تصویر را به لیست اضافه می کند.

آرایه ای را برای ذخیره پیش بینی های آزمایشی راه اندازی می کند.

برای هر کلاس، برچسب های باینری را برای داده های تست پیش بینی می کند و پیش بینی های نهایی را بر اساس آن به روز می کند.

accuracy_score(y_test, y_test_pred): دقت مدل را محاسبه می کند.

confusion_matrix(y_test, y_test_pred): ماتریس سردرگمی را محاسبه می کند.

ConfusionMatrixDisplay: ماتریس سردرگمی را تجسم می کند.

سوال ۳)آ)

الف. بزرگترین چالش ها در توسعه مدل های کشف تقلب و روش های مورد استفاده برای حل این چالش ها:

چالش ها :

پروفایل های رفتار متقلبانه پویا: تراکنش های متقلبانه اغلب شبیه تراکنش های قانونی هستند و تشخیص ناهنجاری ها را دشوار می کند.

مجموعه داده های نامتعادل: تراکنش های متقلبانه در مقایسه با موارد قانونی بسیار نادرتر هستند که منجر به ایجاد مجموعه داده های بسیار منحرف(skewed) می شود.

انتخاب ویژگی بهینه: شناسایی مرتبط ترین ویژگی ها برای کشف تقلب بسیار مهم اما چالش برانگیز است.

معیارهای ارزیابی مناسب: معیارهای استاندارد مانند دقت(accuracy) برای مجموعه داده های نامتعادل موثر نیستند.

روش های حل این چالش ها:

نمونه برداری بیش از حد(oversampling): برای رفع عدم تعادل، مقاله از تکنیک های نمونه برداری بیش از حد استفاده می کند تا نمونه های بیشتری از طبقه اقلیت تولید کند، به عنوان مثال، تراکنش های تقلبی.

Denoising Autoencoder (DAE): این مقاله از یک DAE برای نمونه برداری و حذف نویز مجموعه داده استفاده می کند. این به حفظ اطلاعات طبقه بندی کمک می کند و در عین حال نویز وارد شده در طول فرآیند نمونه برداری بیش از حد را کاهش می دهد.

معیارهای ارزیابی: این مقاله مدل ها را با استفاده از معیارهایی مانند نرخ فراخوان(recall) و دقت در آستانه های مختلف، با تمرکز بر بهبود نرخ تشخیص تراکنش های جعلی ارزیابی می کند.

ب)

ب. معماری شبکه

معماری شبکه ارائه شده در مقاله از دو بخش اصلی تشکیل شده است:

(DAE):

لایه ها: DAE دارای 7 لایه است که به شرح زیر است:

لایه ورودی با نویز گاوسی اضافه شده

لایه های کاملا متصل با اندازه های 29، 22، 15، 10، 15، 22، 29

تابع تلفات مربعی(MSE) که برای بهینه سازی استفاده می شود.

هدف: DAE برای حذف نویز مجموعه داده پس از نمونه برداری بیش از حد استفاده می شود و کیفیت داده را برای طبقه بندی کننده افزایش می دهد.

طبقه بندی:

لایه ها: طبقه بندی کننده دارای 6 لایه است:

لایه های کاملا متصل با اندازه های 29، 22، 15، 10، 5، 2

لایه SoftMax با تابع تلفات متقابل آنتروپی برای طبقه بندی استفاده می شود.(categorical cross entropy)

هدف: این شبکه عصبی کاملا متصل عمیق برای طبقه‌بندی مجموعه داده حذف‌شده با تمرکز بر تمایز بین تراکنش‌های مشروع و جعلی استفاده می‌شود.

ج)

برای اینکه مدل خود را آموزش دهیم به یک سری پیش پردازش هایی نیاز داریم که بدین شکل هستند :

بعد از دانلود کردن دیتا و سیو کردن آن به صورت دیتافریم اول از همه تمام سطر هایی که داده ای در آن وجود ندارد حذف میکنیم سپس ویژگی Amount را نرمالایز میکنیم (با دستور standardscalar)سپس ویژگی class را جدا کرده و آنرا به صورت label ذخیره میکنیم و بقیه ویژگی ها را به صورت features ذخیره میکنیم و ویژگی time را نیز از آنها حذف میکنیم .تعداد کلاس ها در labels بدین شکل است :

```
Class
0    284315
1       492
Name: count, dtype: int64
```

سپس داده ها را با دستور train test split از هم جدا میکنیم با نسبت 2. سپس روش smote را بر روی داده های train استفاده میکنیم.که بدین شکل میشود :

```
Class
0    227464
1       381
Name: count, dtype: int64
Class
0    227464
1    227464
Name: count, dtype: int64
```

همینجور که میبینید اولی قبل از oversampling و بعدی بعد از آن میباشد.

سپس بخش DAE است که بدین شکل میباشد :

```
dae = Sequential([
    GaussianNoise(0.1, input_shape=(X_train_resampled.shape[1],)),
    Dense(29, activation='relu'),
    Dense(22, activation='relu'),
    Dense(15, activation='relu'),
    Dense(10, activation='relu'),
    Dense(15, activation='relu'),
    Dense(22, activation='relu'),
    Dense(29, activation='relu')
])
```

```
dae.compile(optimizer='adam', loss='mse')
```

ورودی آن داده های oversampling با نویزگوسی با انحراف معیار 1. میباشد و optimizer و تابع اتلاف نیز معلوم هستند.

سپس داده های را فیت کرده و predict نیز میکنیم :

```
# Train the DAE
dae.fit(X_train_resampled, X_train_resampled, epochs=100, batch_size=256,
validation_split=0.2, callbacks=[
    EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
])

# Get the denoised output
X_train_denoised = dae.predict(X_train_resampled)
```

تعداد epoch یعنی تعداد دفعاتی که مدل آموزش میبیند و batch size نیز بدین معنی است که داده ها را به ۲۵۶ قسمت تقسیم میکنند و وزن ها بعد از هر بار که batch ها آموزش داده شده اند آپدیت میشوند.

در قسمت earlystopping در واقع ما داده های validation (۲۰ درصد داده های train هستند) خود را معیاری برای توقف آموزش مدل قرار دادیم بدین صورت که اگر تا 10 epoch , validation loss ما بهتر نشود مدل متوقف میشود و وزن های مدل ها نیز save شده اند تا بهترین مدل نیز انتخاب شود:

```
Epoch 98/100
1422/1422 [==============================] - 5s 3ms/step - loss: 8.6106 - val_loss: 21.6380
Epoch 99/100
1422/1422 [==============================] - 5s 4ms/step - loss: 8.6106 - val_loss: 21.6377
Epoch 100/100
1422/1422 [==============================] - 7s 5ms/step - loss: 8.6105 - val_loss: 21.6383
14217/14217 [==============================] - 24s 2ms/step
```

برای قسمت classification ما باید اول از همه داده های y_train را با تکنیک one hot درست کنیم سپس آنها را برای مدل خود استفاده کنیم زیرا categorical cross entropy تابع اتلاف ما میباشد (همانطور که مقاله گفته) و در واقع این تابع اتلاف برای چند کلاسه است و برای همین از این تکنیک استفاده میکنیم:

```
# One-hot encode the labels for the classifier
y_train_resampled = tf.keras.utils.to_categorical(y_train_resampled,
num_classes=2)
y_test_one_hot = tf.keras.utils.to_categorical(y_test, num_classes=2)
```

مدل ما بدین شکل میباشد :

```
# Classifier
classifier = Sequential([
    Dense(29, input_shape=(X_train_denoised.shape[1],),
activation='relu'),
    Dense(22, activation='relu'),
    Dense(15, activation='relu'),
    Dense(10, activation='relu'),
    Dense(5, activation='relu'),
    Dense(2, activation='softmax')
])
classifier.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Model checkpoints
checkpoint = ModelCheckpoint(filepath = 'best_model', monitor='val_loss',
save_best_only=True, mode='min')
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# Train the classifier
history = classifier.fit(X_train_denoised, y_train_resampled, epochs=100,
batch_size=256, validation_split=0.2, callbacks=[checkpoint,
early_stopping])
```

همینطور که میبینید برای آموزش مدل ورودی در واقع خروجی DAE میباشد.در اینجا مدلی save میشود که کم ترین(mode=min) validation loss را دارا باشد.(در فایل best_model ذخیره میشود)

برای آموزش مدل ورودی x_train_denoised و خروجی نیز با y_train_resampled مقایسه(تابع اتلاف) میشود :

```
Epoch 18/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0109 - accuracy: 0.9968 - val_loss: 0.0063 - val_accuracy: 0.9987
Epoch 19/100
1422/1422 [==============================] - 4s 3ms/step - loss: 0.0113 - accuracy: 0.9966 - val_loss: 0.0082 - val_accuracy: 0.9988
Epoch 20/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0102 - accuracy: 0.9970 - val_loss: 0.0070 - val_accuracy: 0.9987
```

برای فراخانی مدلی که بهترین وزن دارد بیاد بدین شکل کد آنرا بنویسیم :

```
from tensorflow.keras.models import load_model

# Load the saved model
model = load_model('best_model')

# Iterate through the layers and display their weights
for layer in model.layers:
    print(f"Layer name: {layer.name}")
    print(f"Weights: {layer.get_weights()}")
    print("="*50)
```

خروجی این کد لایه و بهترین وزن های آنرا به ما میدهد.(که در آن کم ترین تابع اتلاف برای validation را دارا میباشد.)

د)

دقت در مجموعه داده های نامتعادل:

مسئله: دقت معیار مناسبی برای مجموعه داده های نامتعادل نیست زیرا می تواند گمراه کننده باشد. به عنوان مثال، اگر 99 درصد تراکنش‌ها مشروع باشند، مدلی که همه تراکنش‌ها را مشروع پیش‌بینی می‌کند، 99 درصد دقت را خواهد داشت.در اینجا معمولا از confusion matrix استفاده میکنند.

معیارهای جایگزین:

Recall (حساسیت): درصد موارد مثبت واقعی (معاملات متقلبانه) را که به درستی توسط مدل شناسایی شده اند اندازه گیری می کند.(در واقع در این معیار ما داده هایی که متقلب پیش بینی شده اند را نسبت به کل داده های متقلبانه حساب میکنیم)

مرحله بعدی این است که داده های تست که denoise شده اند را وارد مدل classifier خود کنیم و آنرا ارزیابی کنیم :

```
# Get the denoised output
X_test_denoised = dae.predict(X_test)
# Evaluate the model on the test set
y_pred_proba = classifier.predict(X_test_denoised)
y_pred = np.argmax(y_pred_proba, axis=1)
```

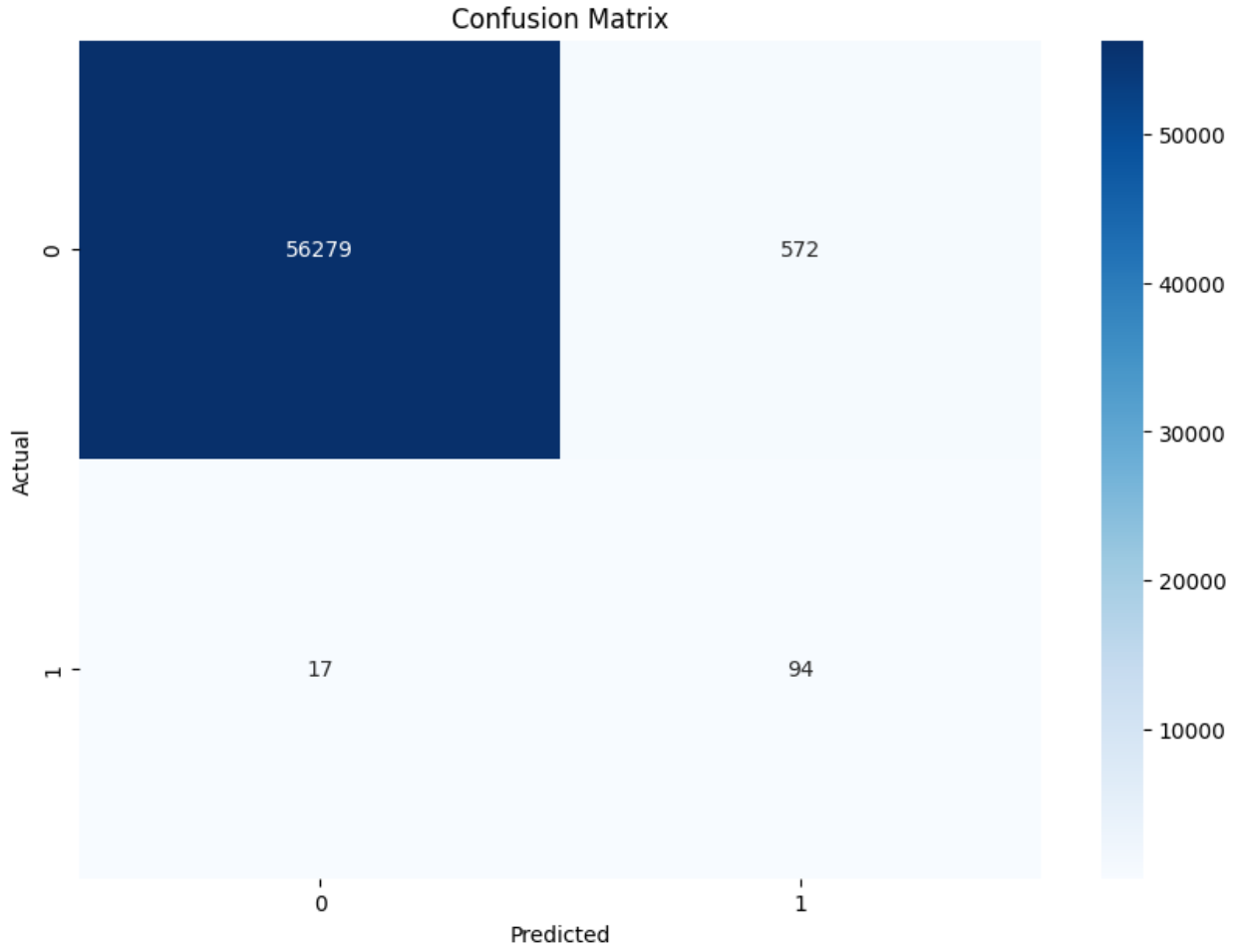
سپس classification report و confusion matrix را میکشیم :

```
1781/1781 [==============================] - 3s 2ms/step
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     56851
           1       0.14      0.85      0.24       111

    accuracy                           0.99     56962
   macro avg       0.57      0.92      0.62     56962
weighted avg       1.00      0.99      0.99     56962
```

خوب همینجور که میبینید مدل ما توانسته recall نسبتا خوبی داشته باشد.که در واقع نشان دهنده است که از بین داده هایی که کلاس ۱ میباشند توانسته 85. آنها را به درستی پیش بینی کند.

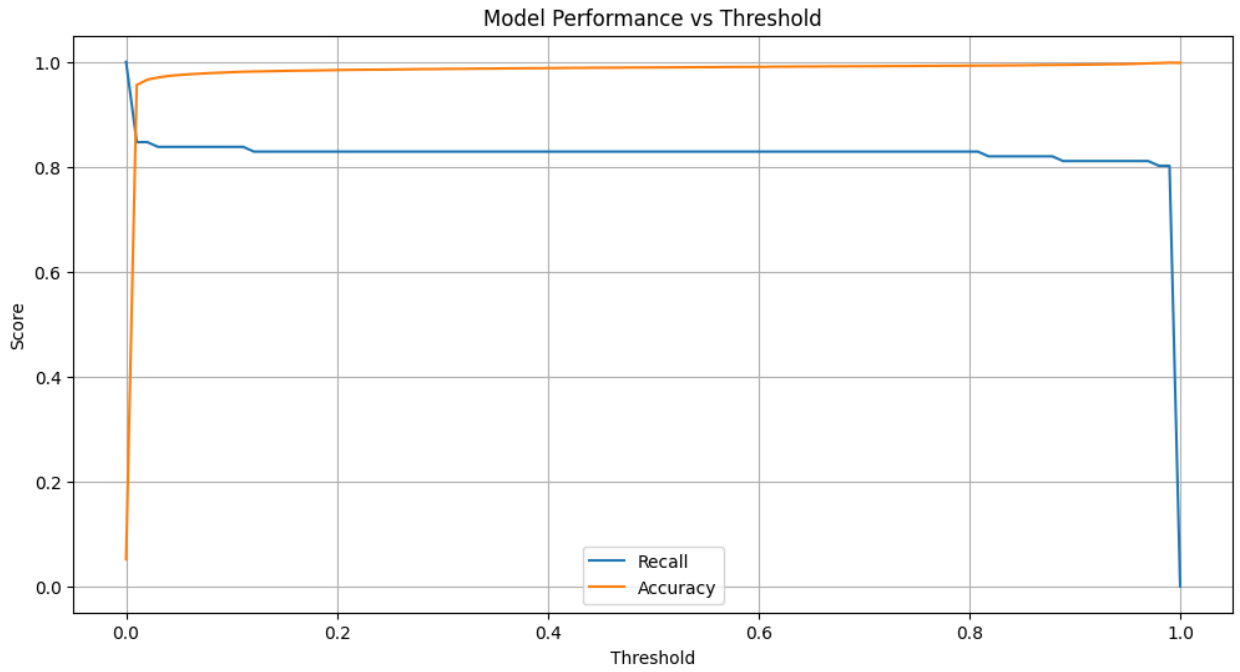


۱۱۱ داده کلاس ۱ (تقلب) داریم که با پیش بینی توانسته ایم ۹۴ تای آنهارا به درستی پیش بینی کنیم.(recall=.85)

۵)

(در واقع این قسمت از سوال که نویسنده خواسته مانند تصویر ۷ مقاله انجام بدیم اشتباه مطرح کرده زیرا در مقاله threshold را تغییر داده نه آستانه برای oversampling را ولی ما هر ۲ را انجام میدهیم.)

آستانه برای پیش بینی(threshold) :



همینطور که میبینید نمودار تقریبا با نمودار مقاله یکی است.همینطور که میبینید در threshold = .5 که مقدار بخش های قبلی میباشد recall = .85 و accuracy = .99 شده اند.در آخر نیز مدل recall آن برابر ۰ میشود و دقت برابر ۱۰۰ میشود که این یعنی مدل نتوانسته هیچ کدام از داده های کلاس تقلب را اندازه گیری کند.قاعدتا واضح است که اگر آستانه را کم تر کنیم مدل بیشتر کلاس ها را کلاس داده های تقلب در نظر میگیرد زیرا داده های بیشتری وجود دارند که از آستانه بیشتر باشند پس به کلاس ۱ تعلق میگیرند و در این صورت داده های سالم را درست تشخیص نمیدهد برای همین accuracy آن کم است.

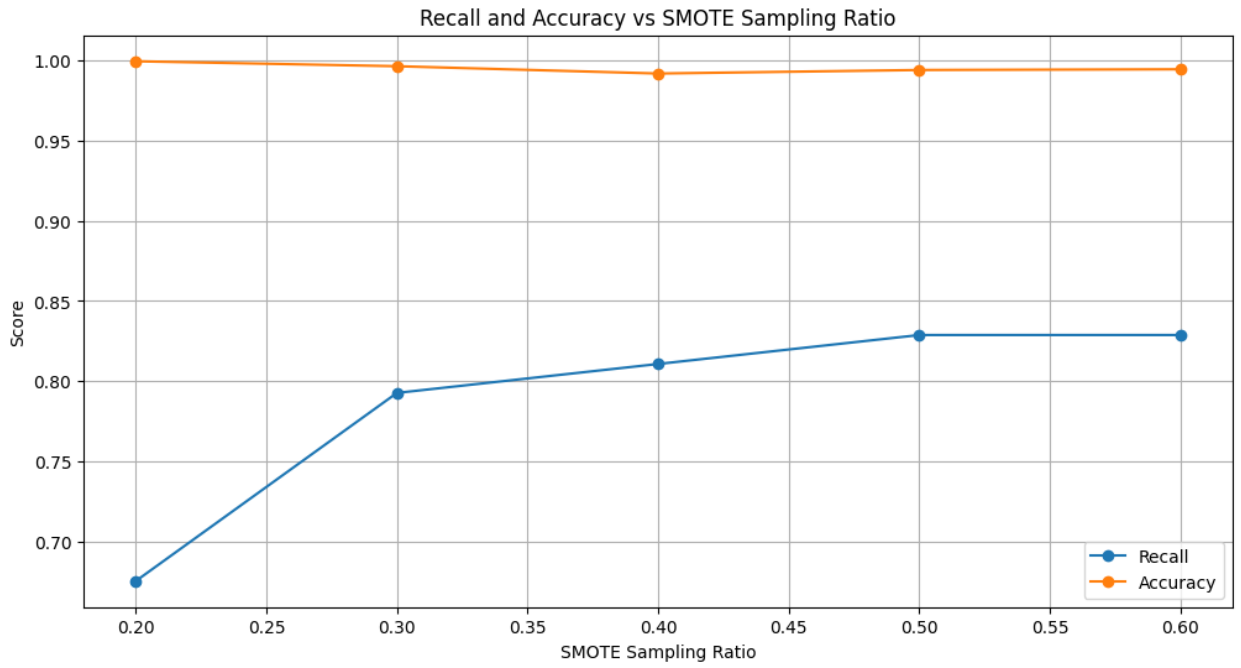
آستانه برای oversampling :

در اینجا تنها تفاوت آن این است که باید ratio های متفاوت معلوم کنیم و در یک حلقه for بنویسیم که مدل ها را بر روی ratio های مختلف درست کند و طبقه بندی کند:

```
# Store results
ratios = [0.2, 0.3, 0.4, 0.5 , .6]
recall_scores = []
accuracy_scores = []

for ratio in ratios:
    # Apply SMOTE with different ratios
    smote = SMOTE(sampling_strategy=ratio, random_state=64)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)
```

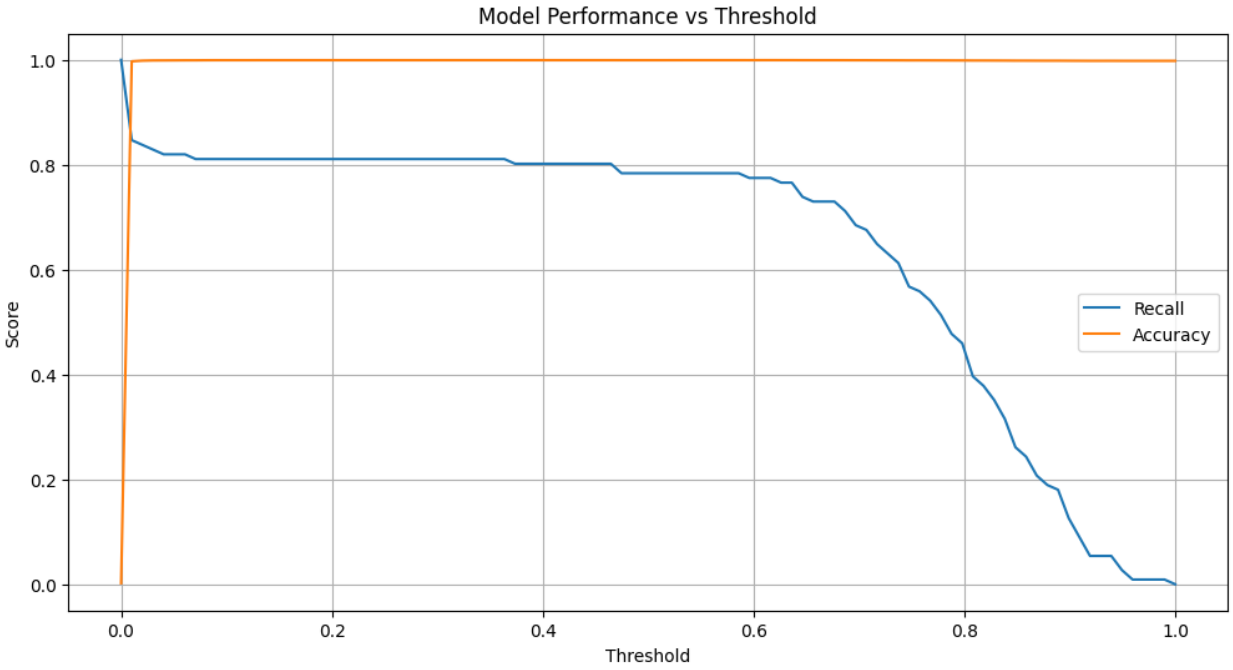
که نتیجه آن بدین شکل میشود :



همینجور که مشاهده میکنید با افزایش آستانه oversampling , recall بیتشر شده و accuracy کم تر میشود این به این دلیل است که مدل ما وقتی داده های کلاس ۱ بیشتر شوند بهتر میتواند این کلاس را یاد بگیرد به همین دلیل این اتفاق افتاده است.

و)

در اینجا دیگه از DEA و oversampling استفاده نمیکنیم یعنی در واقع تمام داده ها را به classifier خود میدهیم که بدین شکل میشود: (دلیل اینکه بخش قبلی اشتباه مطرح شده همین قسمت است زیرا این قسمت oversampling ندارد ولی خواسته های بخش قبلی را از ما خواسته است مانند شکل ۶ مقاله)



همینطور که میبینید accuracy از بعد از یک آستانه ای ۱۰۰ درصد شده است ولی با افزایش این آستانه recall کم تر شده تا در آخر ۰ میشود در واقع دلیل این اتفاق این است که مدل دیگر نمیتواند داده های تقلب را تشخیص دهد به همین دلیل به ۰ میرسد.همینطور که میبینید مدل ما بهتر از مقاله عمل کرده است.

در حالت معمول : چون معمولا threshold برابر 5. است میتوان از روی نمودار خواند که accuracy = 100 و recall = .78 میباشد.

خوب همینجور که میبینید نمودار نسبت به بخش قبلی recall آن نتوانسته به خوبی عمل کند جدا از اینکه Noise آن گرفته نشده میتوان بدین موضوع اشاره کرد که تعداد داده های کلاس ۱ خیلی کم تر از داده های نرمال است و این موضوع باعث میشود که نتواند این کلاس را به درستی یاد بگیرد به همین دلیل در این بخش recall نتوانسته به خوبی عمل کند.

همانطور که در بخش قبل دیدیم نمودار ما تا قبل از آستانه تقریبا 9. توانسته بالای ۸۰ درصد recallداشته باشد و accuracy آن نیز بالا است ولی در اینجا بعد از تقریبا آستانه 45. این مقدار کم تر از 8. شده است ولی

accuracy آن نیز همیشه (به جز آستانه های اول) برابر تقریبا ۱۰۰ بوده زیرا درصد داده های کلاس ۱ خیلی کم میباشد و مدل نتوانسته این کلاس را به خوبی یاد بگیرد و کلاس ۰ را به خوبی یاد گرفته به همین دلیل accuracy نمودار مانند بخش قبل نیست.

گزارش کد :

در اینجا فقط گزارش کد برای آستانه برای prediction آورده ایم :

```
# Define a function to calculate metrics at different thresholds
def evaluate_thresholds(model, X_test, y_test, thresholds):
    recall_scores = []
    accuracy_scores = []
    precision_scores = []
    f1_scores = []

    y_pred_proba = model.predict(X_test)

    for threshold in thresholds:
        y_pred = (y_pred_proba[:, 1] > threshold).astype(int)  # Get
binary predictions for the positive class
        recall = recall_score(y_test, y_pred)
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        recall_scores.append(recall)
        accuracy_scores.append(accuracy)
        precision_scores.append(precision)
        f1_scores.append(f1)

    return recall_scores, accuracy_scores, precision_scores, f1_scores

# Define thresholds to evaluate
thresholds = np.linspace(0, 1, 100)

# Assuming the classifier model is already trained and available as
'classifier'
recall_scores, accuracy_scores, precision_scores, f1_scores =
evaluate_thresholds(classifier, X_test, y_test, thresholds)

# Plot the results
```

```
plt.figure(figsize=(12, 6))
plt.plot(thresholds, recall_scores, label='Recall')
plt.plot(thresholds, accuracy_scores, label='Accuracy')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.title('Model Performance vs Threshold')
plt.legend()
plt.grid(True)
plt.show()
```

این در واقع functionمیباشد که ورودی آن مدل و داده های تست و آزمون و آستانه ها میباشد. تنها موضوع جدید این خط کد میباشد :

```
        y_pred = (y_pred_proba[:, 1] > threshold).astype(int)  # Get
binary predictions for the positive class
```

y_pred_proba آرایه ای از احتمالات پیش بینی شده خروجی توسط مدل است.

این آرایه معمولاً شکلی از (num_samples، num_classes) دارد. برای یک مسئله طبقه بندی باینری، شکلی از (num_samples، 2) دارد که در آن:

y_pred_proba [:,0] احتمال پیش بینی شده کلاس 0 (غیر تقلب) را می دهد.

y_pred_proba[:,1] احتمال پیش بینی شده کلاس 1 (تقلب) را می دهد.

با انتخاب y_pred_proba[:,1] روی احتمالات پیش بینی شده برای کلاس مثبت (تقلب) تمرکز می کنیم.

y_pred_proba[:,1] > آستانه:

این هر احتمال پیش بینی شده برای کلاس مثبت را با آستانه مشخص شده مقایسه می کند.

نتیجه این مقایسه یک آرایه بولی است که در آن هر عنصر اگر احتمال پیش‌بینی‌شده بیشتر از آستانه باشد True و در غیر این صورت False است.

.atype(int):

این آرایه بولی را به یک آرایه عدد صحیح تبدیل می کند.

True به 1 (نشان دهنده کلاس 1، تقلب) و False به 0 (نشان دهنده کلاس 0، غیر تقلبی) تبدیل می شود.