بسمه تعالی

مهدی وحیدمقدم

۴۰۲۱۳۰۷۴

مینی‌پروژه شماره۳ درس یادگیری ماشین

فهرست مطالب

لینک کد در گوگل کولب:

https://colab.research.google.com/drive/1tE4yVlZXpHqnCUS9ymDyg2qsvtwfiWRz?usp=sharing

لینک گیت‌هاب:

https://github.com/mvmoghadam1999/ML403_MP1_40213074

سوال ۱)

الف)

آ. در مرحلهٔ اول دیتاست را فراخوانی کنید و اطلاعاتی نظیر ابعاد، تعداد نمونه‌ها، میانگین، واریانس و همبستگی ویژگی‌ها را به‌دست 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.

کد به صورت زیر است:

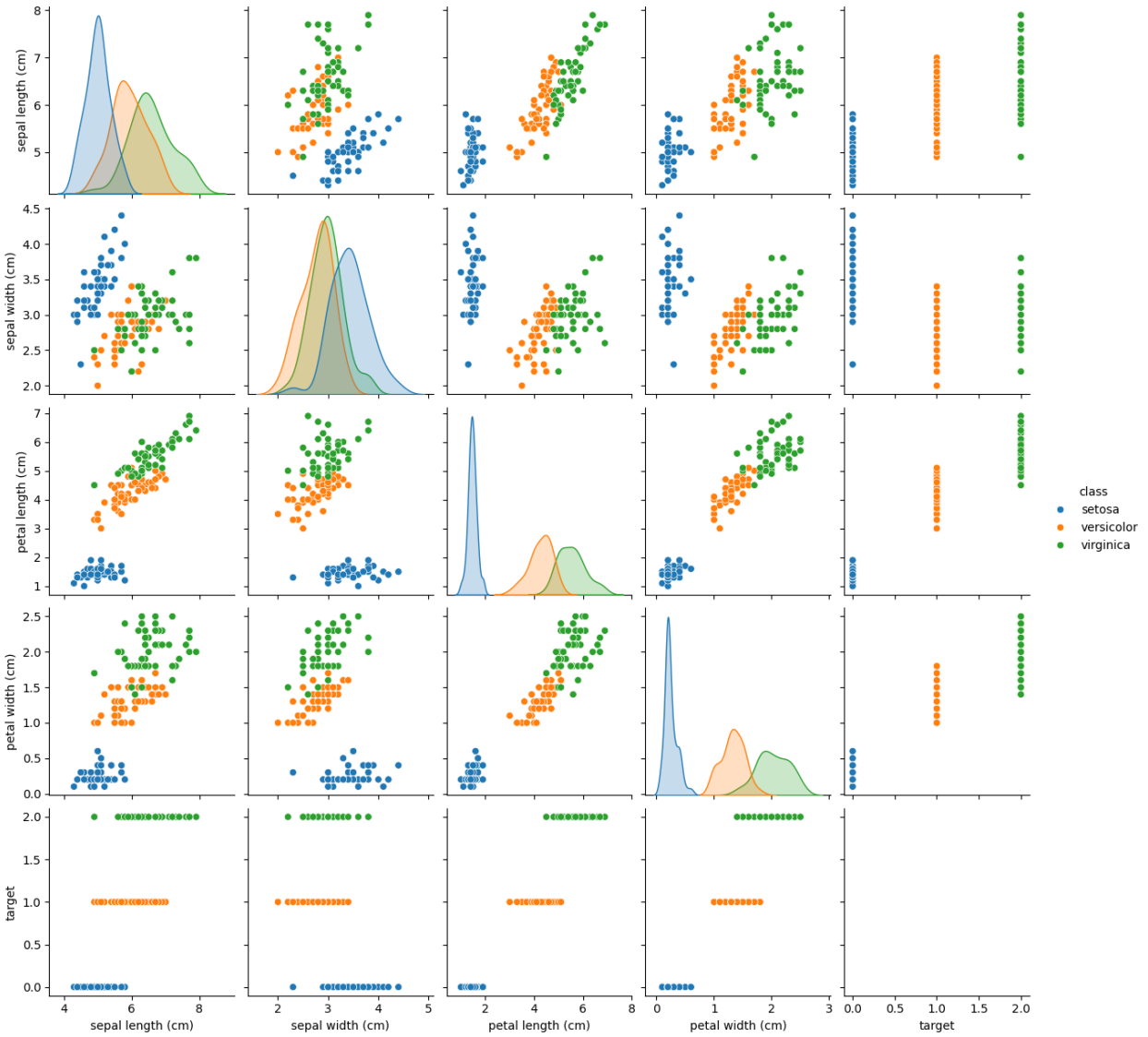
```
import pandas as pd
from sklearn.datasets import load_iris
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
import numpy as np


iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target


target_names = {0: 'setosa', 1: 'versicolor', 2: 'virginica'}
df['class'] = df['target'].map(target_names)


features = df.drop(columns=['target', 'class'])

print("Dimensions of dataset:", features.shape)
print("Number of samples:", len(df))
print("Mean of features:\n", features.mean())
print("Variance of features:\n", features.var())
print("Correlation of features:\n", features.corr())


sns.pairplot(df, hue='class', diag_kind='kde')
plt.show()


tsne = TSNE(n_components=2, random_state=42)
tsne_result = tsne.fit_transform(iris.data)

plt.figure(figsize=(10, 7))
```

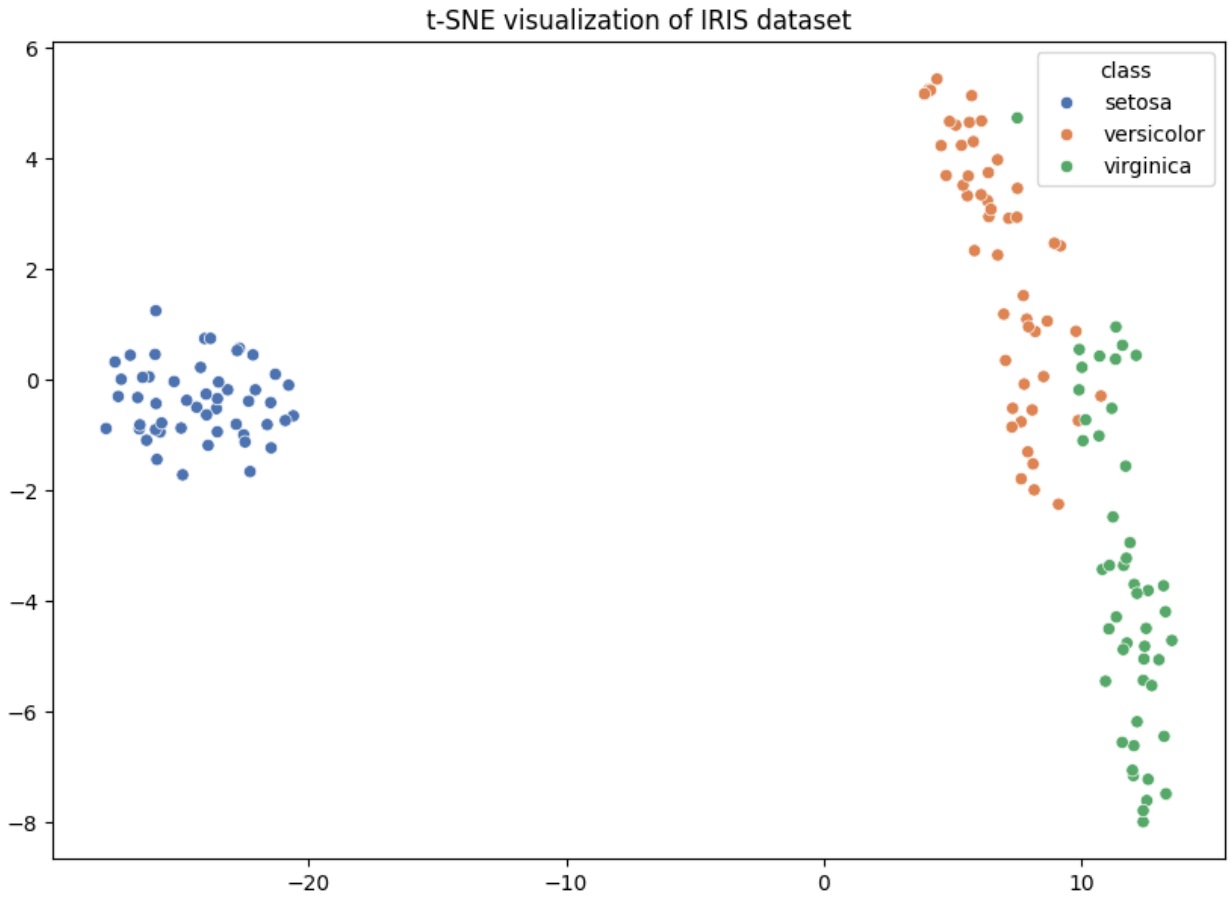
```
sns.scatterplot(x=tsne_result[:, 0], y=tsne_result[:, 1], hue=df['class'],
palette="deep", legend="full")
plt.title("t-SNE visualization of IRIS dataset")
plt.show()
```

در این کد ابتدا دیتاست iris فراخوانی می‌شود.سپس برای میانگین‌گیری و سایر کارها ستون‌های ویژگی‌ها از ستون برچسب‌ها جدا می‌شوند.همچنین نام کلاس‌ها به برچسب‌های موردنظر اختصاص داده می‌شوند.

در قسمت بعد اطلاعاتی نظیر تعداد داده‌ها،تعداد نمونه‌های هر کلاس،میانگین داده‌های هر ویژگی و واریانس آن‌ها و همبستگی ویژگی‌های مختلف نمایش داده می‌شوند.همچنین پس از آن در یک شکل نمونه‌های دیتاست به تصویر کشیده شده‌اند. نتیجه به صورت زیر است:

```
Dimensions of dataset: (150, 4)
Number of samples: 150
Mean of features:
 sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
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
dtype: float64
Correlation of features:
                   sepal length (cm)  sepal width (cm)  petal length (cm)  \
sepal length (cm)           1.000000         -0.117570           0.871754
sepal width (cm)           -0.117570          1.000000          -0.428440
petal length (cm)           0.871754         -0.428440           1.000000
petal width (cm)            0.817941         -0.366126           0.962865

                   petal width (cm)
sepal length (cm)          0.817941
sepal width (cm)          -0.366126
petal length (cm)          0.962865
petal width (cm)           1.000000
```


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
target
class
setosa
versicolor
virginica



همان‌طور که مشاهده می‌شود،اگر به ماتریس همبستگی دقت کنیم،با توجه به همبستگی نسبتا زیاد برخی ویژگی‌ها،کاهش بعد در این مسئله می‌تواند به طبقه‌بندی بهتر کمک کند.پس در این دیتاست کاهش بعد خواهیم داشت.

ب)

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

کد به صورت زیر است:

```
from sklearn.svm import SVC
from sklearn.decomposition import PCA
```

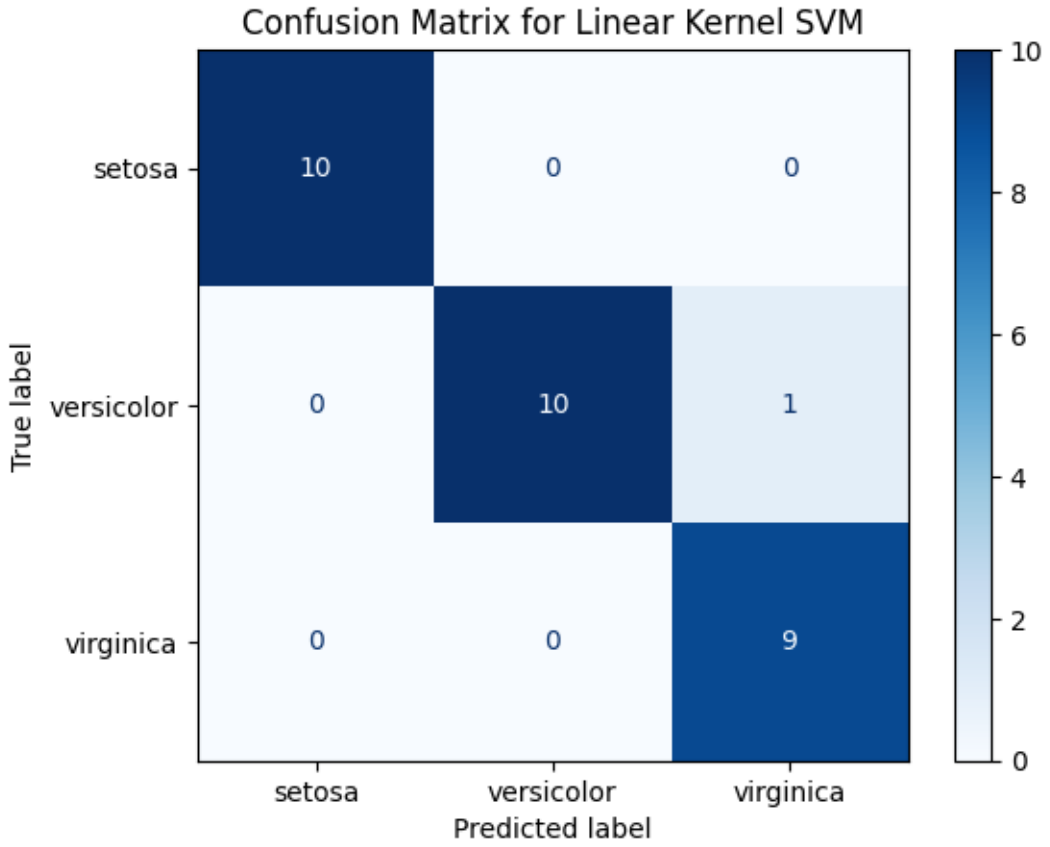
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(iris.data)
X_reduced_train ,X_reduced_test  , y_train , y_test =
train_test_split(X_reduced , iris.target , test_size = 0.2 , random_state
= 74)

svm_linear = SVC(kernel='linear')
svm_linear.fit(X_reduced_train, y_train)


y_pred = svm_linear.predict(X_reduced_test)
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=iris.target_names)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix for Linear Kernel SVM")
plt.show()


def plot_decision_boundary(clf, X, y, title , save_path):
    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, 0.02), np.arange(y_min,
y_max, 0.02))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.8)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o')
    plt.title(title)
    plt.savefig(save_path)
    plt.show()

plot_decision_boundary(svm_linear, X_reduced_test, y_test, "Decision
Boundary for Linear Kernel SVM" , save_path = 'svc_linear.png')
```

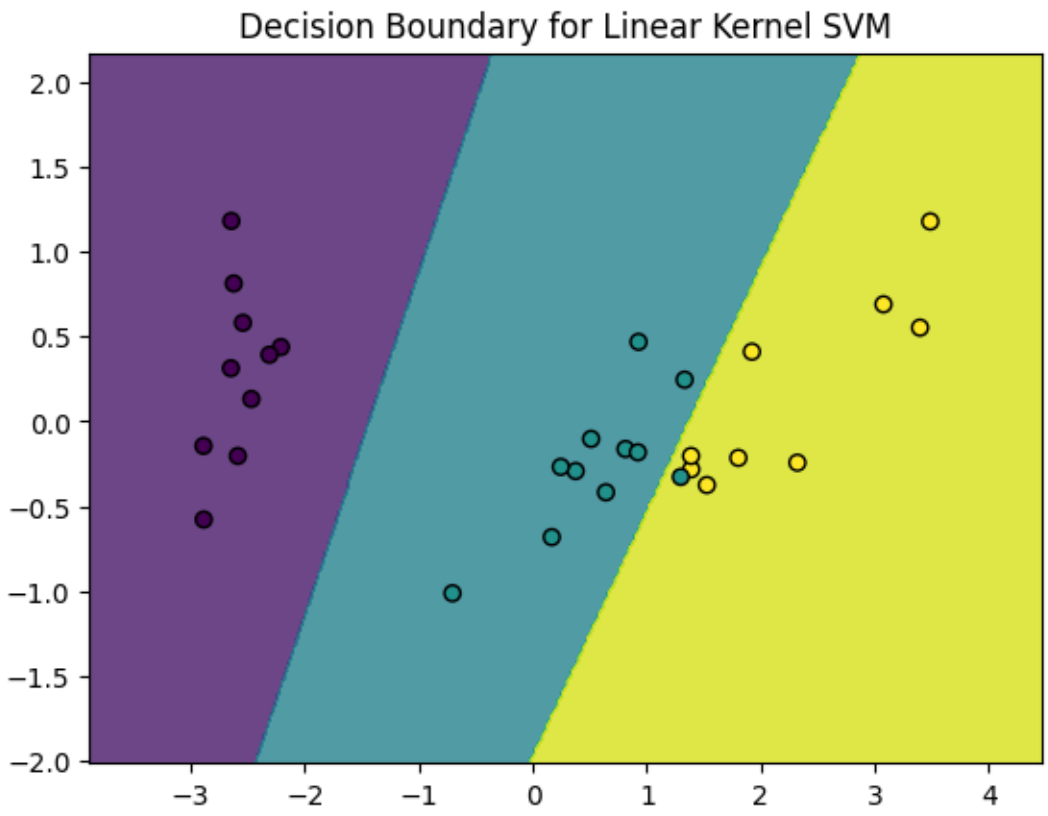
در این کد ابتدا از الگوریتم PCA برای کاهش بعد به دو بعد استفاده می‌شود.همان‌طور که گفته شد این کاهش بعد به این دلیل است که همبستگی‌های برخی ویژگی‌ها به یکدیگر زیاد است و تقریبا دو ویژگی هستند که همبستگی دارند.

پس از کاهش بعد،داده‌ها به دو دسته‌ی آموزش و ارزیابی تقسیم می‌شوند.سپس مدل SVM با هسته خطی تعریف می‌شود و این مدل روی داده‌های آموزش،آموزش می‌بیند.سپس با داده‌های ارزیابی،پیش‌بینی با مدل انجام می‌شود.سپس،ماتریس درهم‌ریختگی تعریف و در یک شکل نمایش داده می‌شود.

در قسمت بعد،یک تابع برای رسم مرزهای تصمیم‌گیری در نظر گرفته شده است.این تابع علاوه بر رسم مرزهای تصمیم‌گیری،تصویر مربوط به آن را در آدرس داده شده ذخیره خواهد کرد.

خروجی به صورت زیر است:





همان‌طور که مشاهده می‌شود،دقت این طبقه‌بندی بالا بوده و SVM خطی عملکرد خوبی داشته است.

ج)

ج. بخش قبلی را با استفاده از هسته‌های چند جمله‌ای و با استفاده از کتابخانهٔ 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 کتابخانهٔ imageio جداسازی ویژگی‌های اصلی را (کاهش بُعد از طریق یکی از روش‌های آموخته‌شده 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.

کد برای این بخش به صورت زیر است:

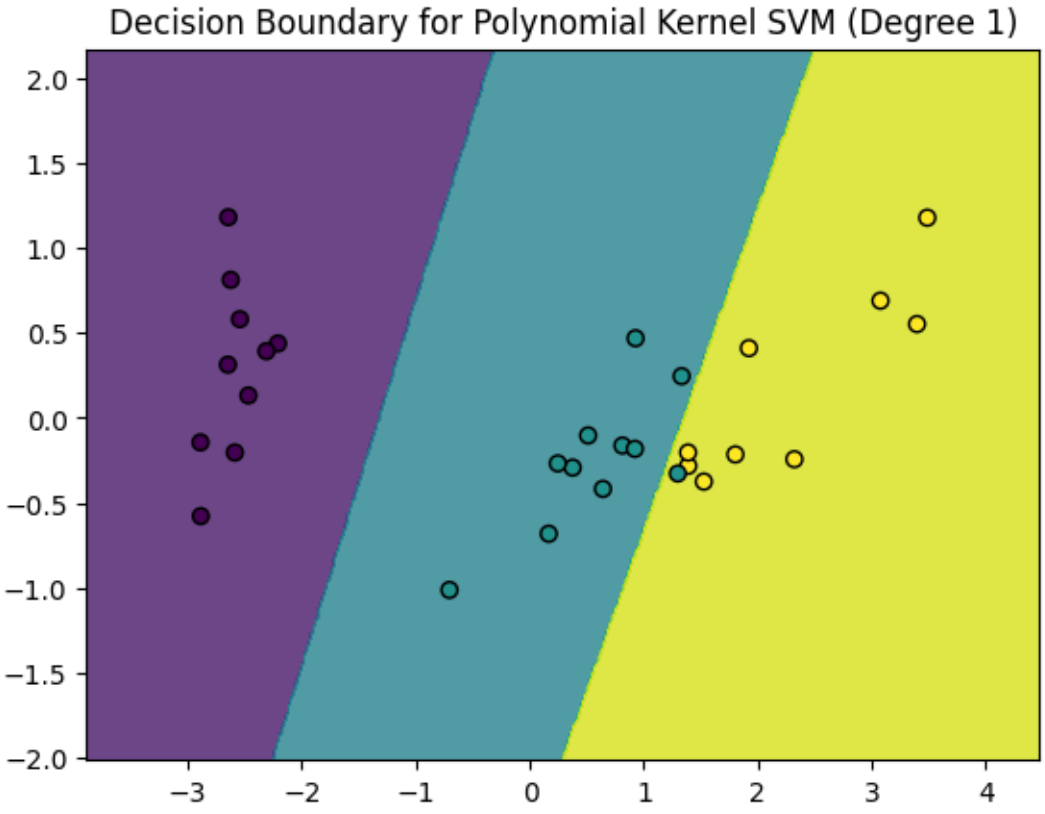
```
import os
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
import imageio
```

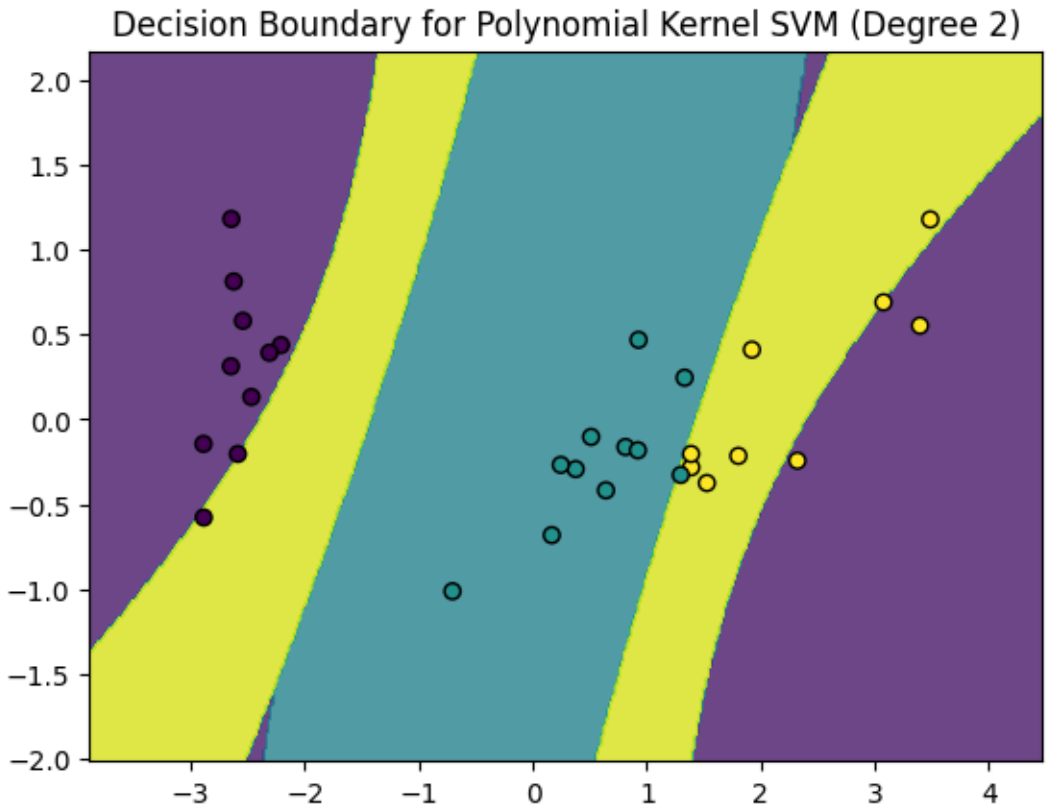
```
results = []
accuracy = []


for degree in range(1, 11):

    svm_poly = SVC(kernel='poly', degree=degree)
    svm_poly.fit(X_reduced, iris.target)

    y_pred_poly = svm_poly.predict(X_reduced)
    acc = accuracy_score(iris.target, y_pred_poly)
    accuracy.append(acc)


    plt.figure()
    plot_decision_boundary(svm_poly, X_reduced_test, y_test, f"Decision
Boundary for Polynomial Kernel SVM (Degree {degree})" , save_path =
f'svm_poly_degree_{degree}.png')


    results.append((degree, acc))


images = []
for degree in range(1, 11):
    images.append(imageio.imread(f'svm_poly_degree_{degree}.png'))
imageio.mimsave('svm_poly_kernels.gif', images, duration=1)


for degree, acc in results:
    print(f"Degree {degree}: Accuracy = {acc}")

print("GIF saved as 'svm_poly_kernels.gif'")
```

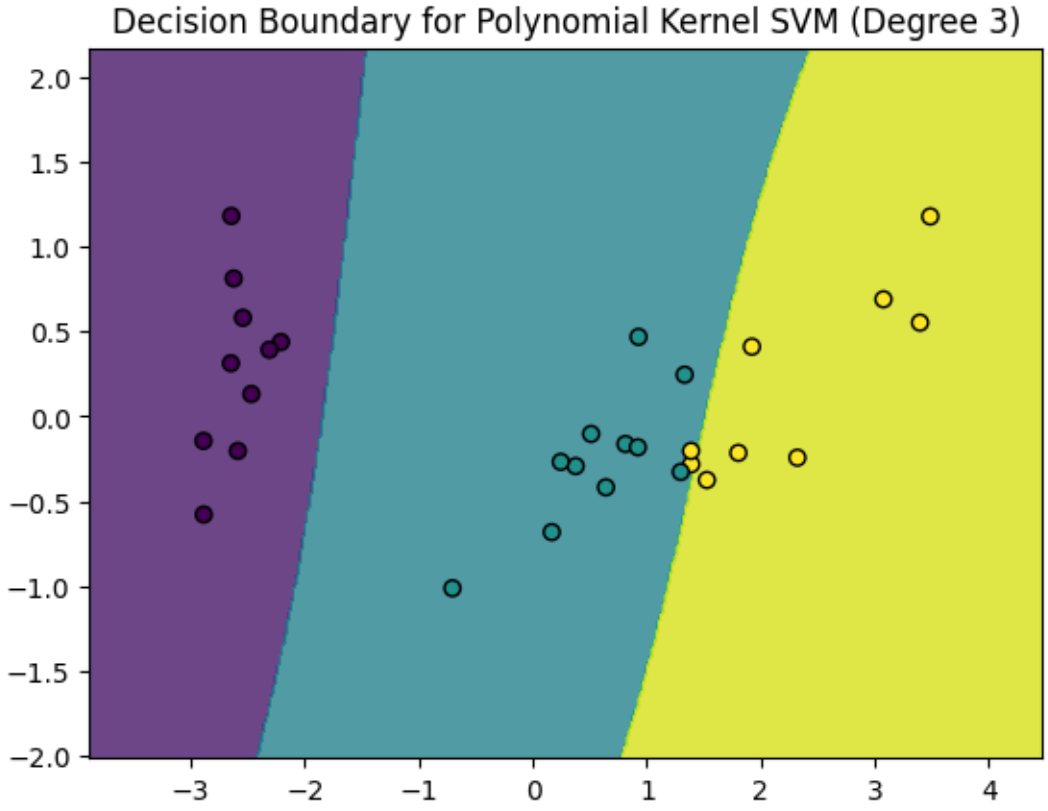
در این قسمت،به ازای درجه‌های ۱ تا ۱۰ مدل‌های SVM با هسته‌های چندجمله‌ای ایجاد می‌شود.به ازای هر یک از درجه‌ها،مدل آموزش می‌بیند و روی داده‌های تست پیش‌بینی انجام می‌شود.همچنین در هر حالت مرزهای تصمیم‌گیری رسم شده و ذخیره می‌شود تا در فایل گیف مورداستفاده قرار گیرد.دقت در هر مرحله هم در یک لیست ذخیره می‌شود تا در نهایت دقت به ازای هر یک از درجه‌ها،نمایش داده شود.

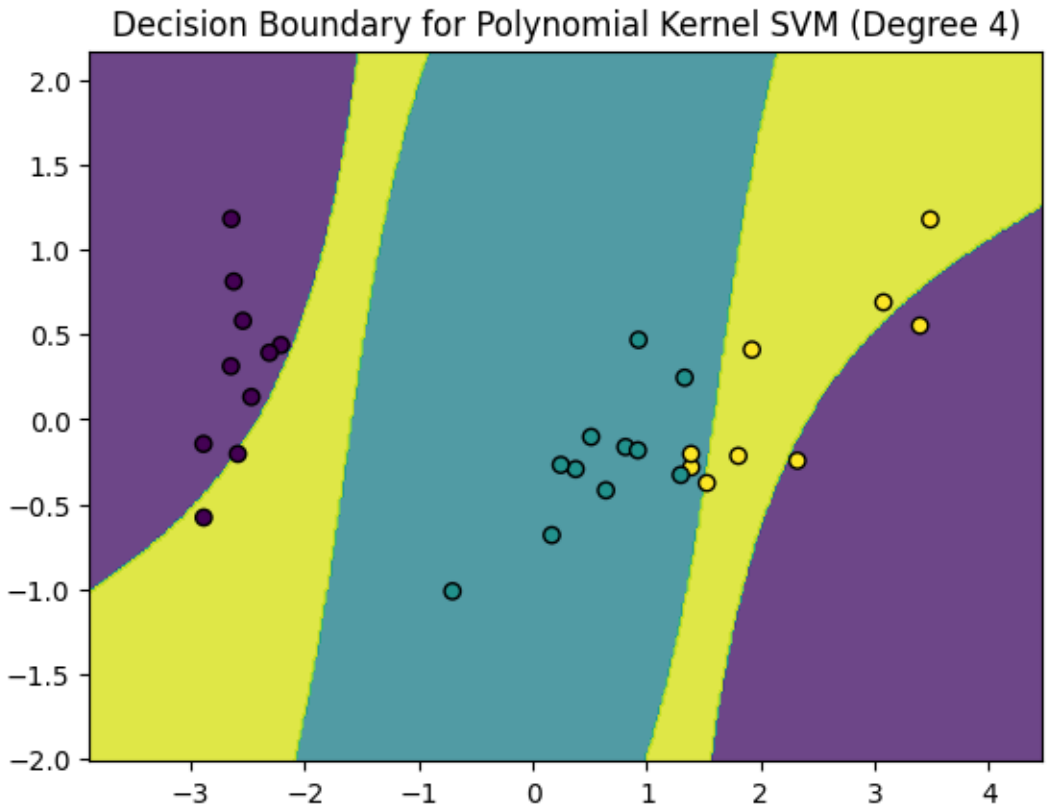
در قسمت آخر هم تصاویر مرزهای تصمیم‌گیری که از درجه‌های مختلف ذخیره شده بودند،به صورت یک گیف با زمان یک ثانیه ذخیره می‌شوند.

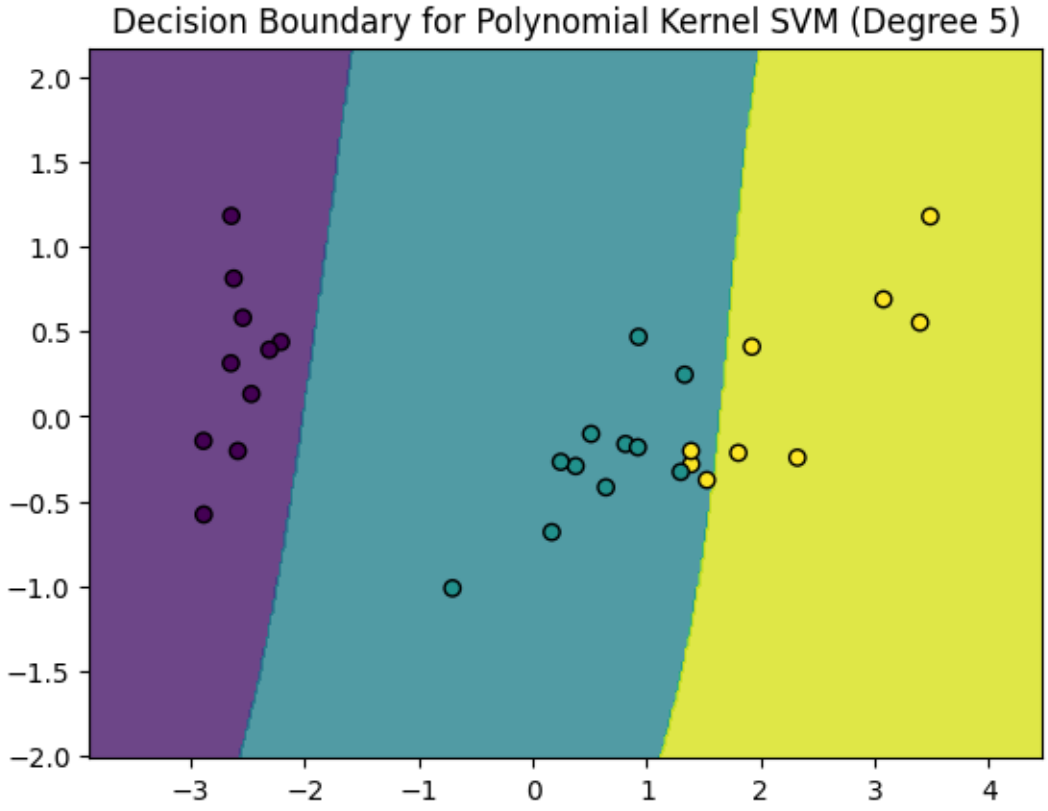
نتیجه به صورت زیر است:

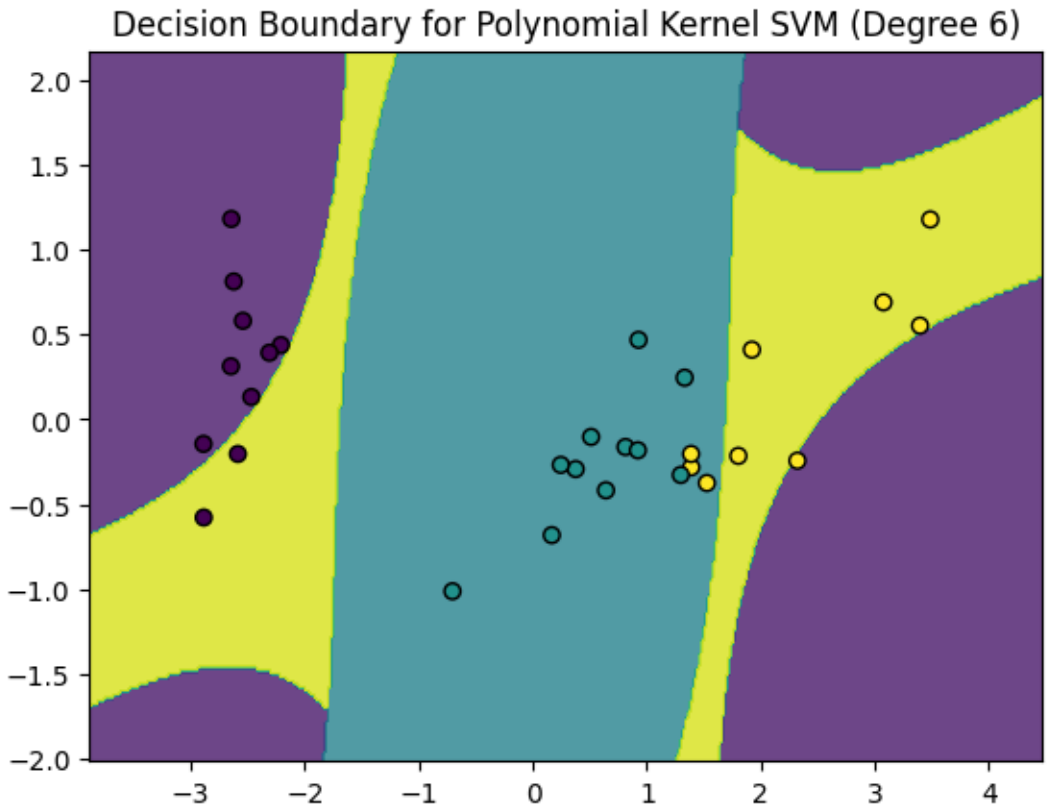


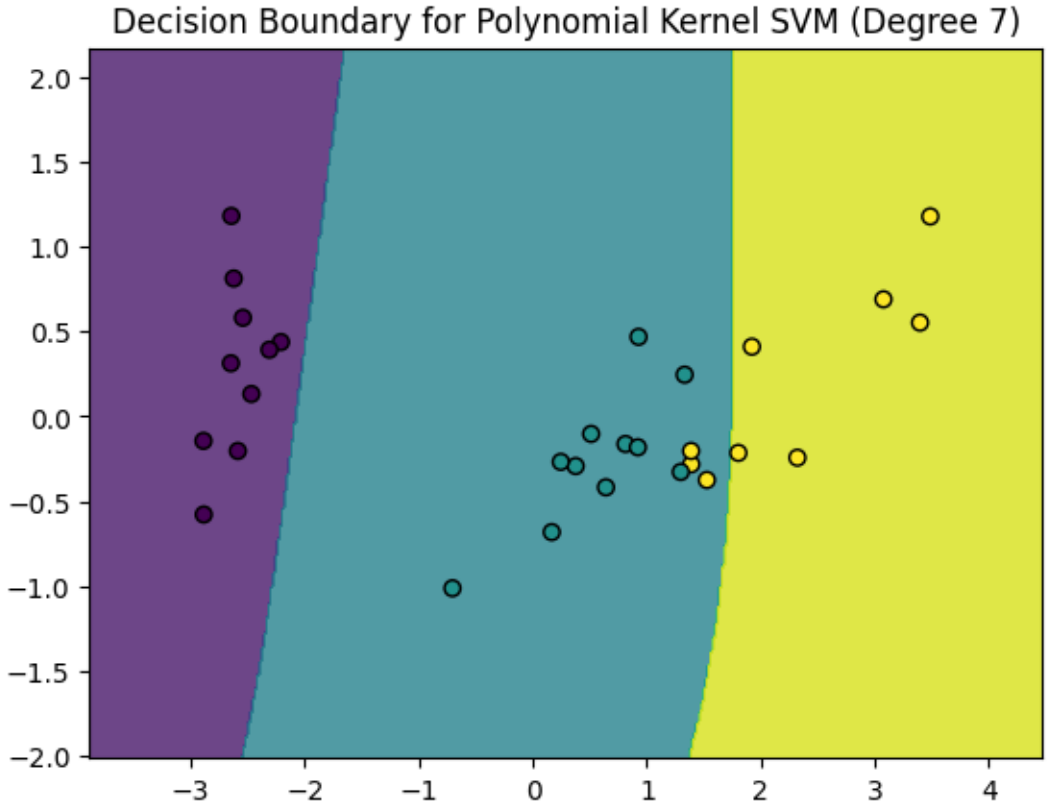

Decision Boundary for Polynomial Kernel SVM (Degree 2)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2
3
4

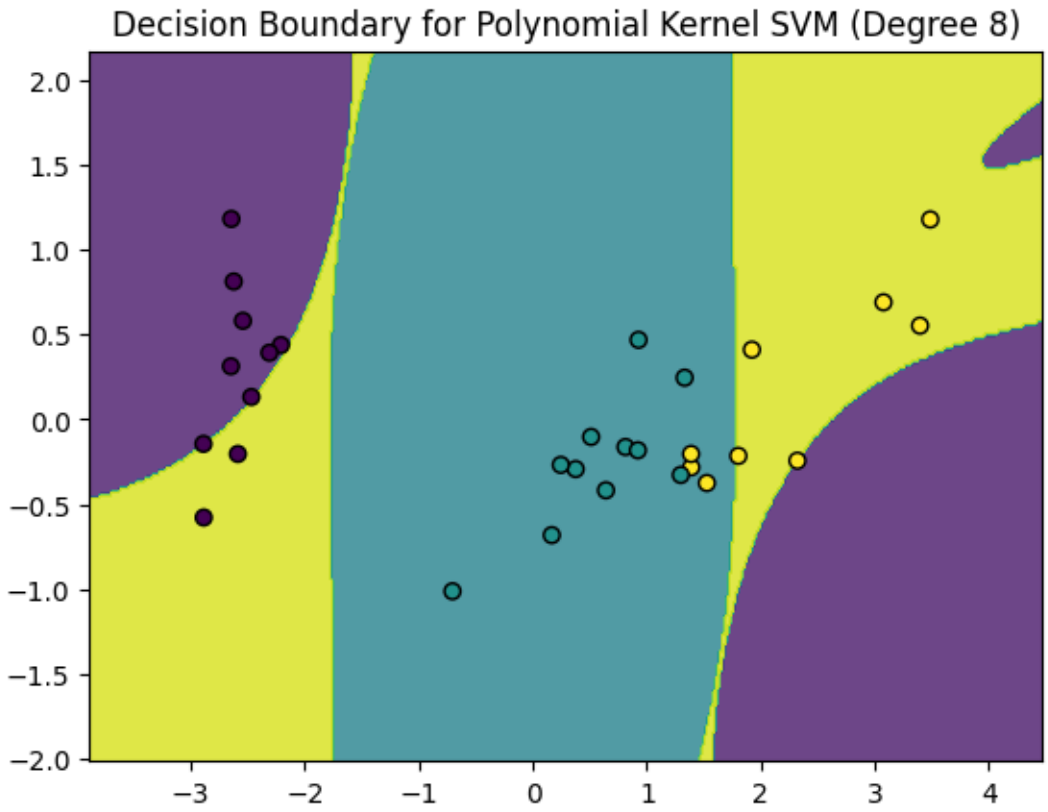

Decision Boundary for Polynomial Kernel SVM (Degree 3)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2
3
4

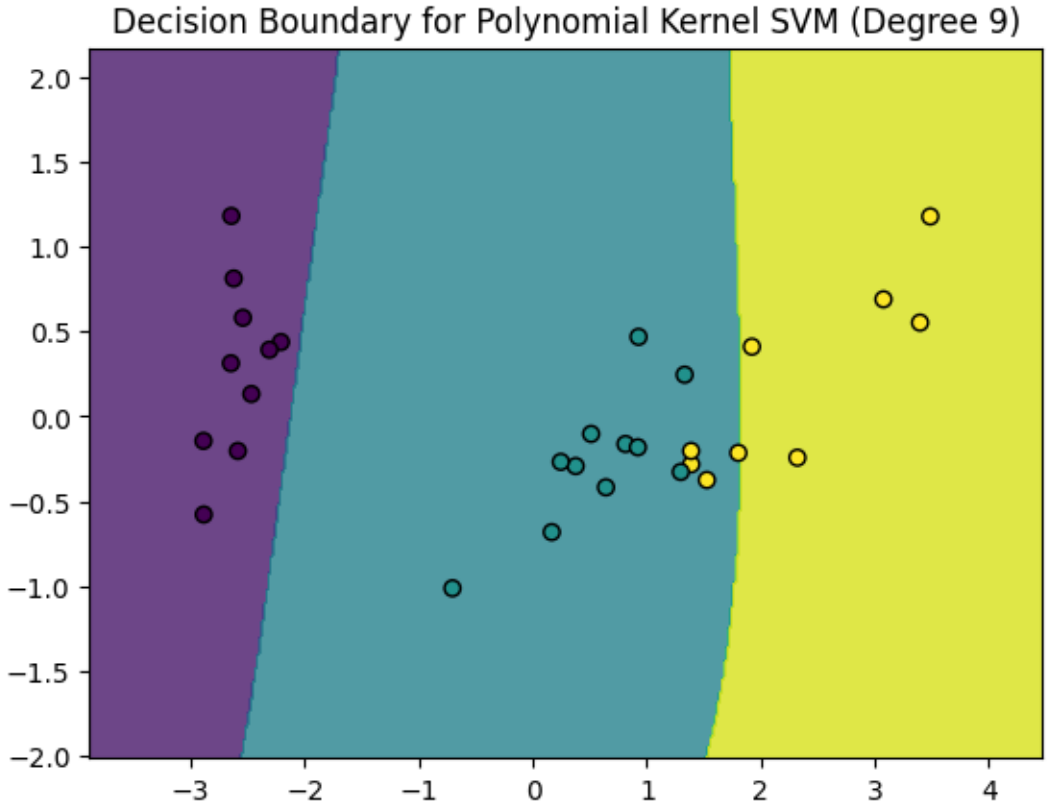

Decision Boundary for Polynomial Kernel SVM (Degree 4)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2
3
4

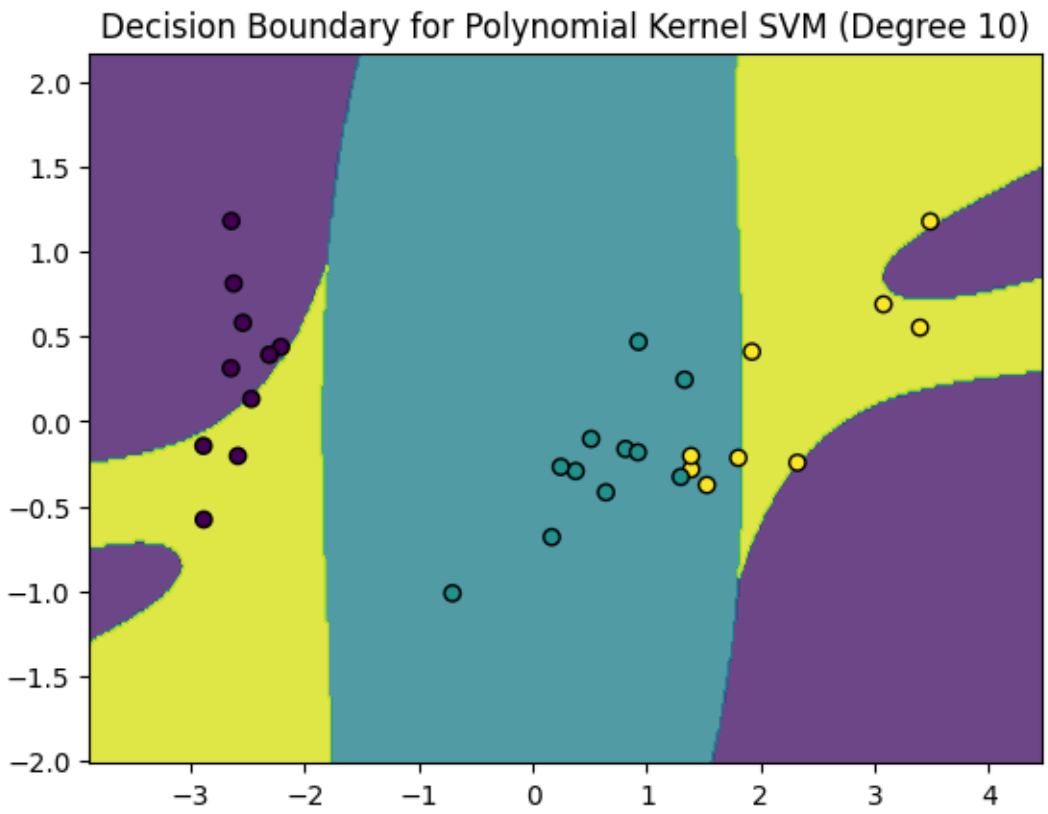

Decision Boundary for Polynomial Kernel SVM (Degree 5)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2
3
4


Decision Boundary for Polynomial Kernel SVM (Degree 6)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2
3
4


Decision Boundary for Polynomial Kernel SVM (Degree 7)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2
3
4


Decision Boundary for Polynomial Kernel SVM (Degree 8)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2
3
4


Decision Boundary for Polynomial Kernel SVM (Degree 9)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−3
−2
−1
0
1
2
3
4




```
<ipython-input-24-1dff1f966228>:37: DeprecationWarning: Starting with ImageIO
v3 the behavior of this function will switch to that of iio.v3.imread. To
keep the current behavior (and make this warning disappear) use `import
imageio.v2 as imageio` or call `imageio.v2.imread` directly.
  images.append(imageio.imread(f'svm_poly_degree_{degree}.png'))
Degree 1: Accuracy = 0.9666666666666667
Degree 2: Accuracy = 0.8733333333333333
Degree 3: Accuracy = 0.9466666666666667
Degree 4: Accuracy = 0.8333333333333334
Degree 5: Accuracy = 0.9066666666666666
Degree 6: Accuracy = 0.7733333333333333
Degree 7: Accuracy = 0.9
Degree 8: Accuracy = 0.7666666666666667
Degree 9: Accuracy = 0.88
Degree 10: Accuracy = 0.74
GIF saved as 'svm_poly_kernels.gif'
```

همان‌طور که مشاهده می‌شود،دقت پیش‌بینی مدل با افزایش درجه‌ی مدل،لزوما افزایش نداشته است و بیشترین دقت مربوط به درجه‌ی ۱ می‌باشد.پس در هر مسئله‌ای،لزوما افزایش درجه باعث افزایش دقت نخواهد شد.

همچنین فایل گیف ایجاد شده در این لینک در دسترس است.

د)

د. حال الگوریتم SVM را برای مورد قبلی، بدون استفاده از کتابخانهٔ scikit-learn و به‌صورت From Scratch پیاده‌سازی کنید. در این بخش لازم است که یک کلاس SVM تعریف کنید. این کلاس می‌بایست حداقل دارای سه تابع (متد) Fit، Polynomial_kernel و Predict باشد. متد Polynomial_kernel می‌بایست با دریافت درجه‌های ۱ تا ۱۰، هسته‌های چندجمله‌ای را محاسبه کند. دقت الگوریتم را با افزایش درجه گزارش کنید و نتایج حاصل را با بخش قبلی مقایسه کنید. در این قسمت نیز جداسازی ویژگی‌های اصلی را برای درجات ۱ تا ۱۰ در قالب یک GIF به تصویر بکشید پیوند دسترسی مستقیم آن را در گزارش خود قرار دهید.

کد مربوط به تعریف توابع به صورت زیر است:

```
import cvxopt
def linear_kernel( x1, x2):
    return np.dot(x1, x2)

def polynomial_kernel( x, y, C=1.0, d=3):
    return (np.dot(x, y) + C) ** d

def gaussian_kernel( x, y, gamma=0.5):
    return np.exp(-gamma*np.linalg.norm(x - y) ** 2)

def sigmoid_kernel( x, y, alpha=1, C=0.01):
    a= alpha * np.dot(x, y) + C
    return np.tanh(a)

def SVM_scratch(X, X_t, y, C, kernel_type, poly_params=(1, 4),
RBF_params=0.5, sigmoid_params=(1, 0.01)):
    kernel_and_params=(kernel_type,poly_params, RBF_params,
sigmoid_params,C)
    n_samples, n_features = X.shape
    # Compute the Gram matrix
    K = np.zeros((n_samples, n_samples))
    if kernel_type == 'linear':
        for i in range(n_samples):
            for j in range(n_samples):
                K[i, j] = linear_kernel(X[i], X[j])
    elif kernel_type == 'polynomial':
        for i in range(n_samples):
            for j in range(n_samples):
```

```
                K[i, j] = polynomial_kernel(X[i], X[j], poly_params[0],
poly_params[1])
    elif kernel_type == 'RBF':
        for i in range(n_samples):
            for j in range(n_samples):
                K[i, j] = gaussian_kernel(X[i], X[j], RBF_params)
    elif kernel_type == 'sigmoid':
        for i in range(n_samples):
            for j in range(n_samples):
                K[i, j] = sigmoid_kernel(X[i], X[j], sigmoid_params[0],
sigmoid_params[1])
    else:
        raise ValueError("Invalid kernel type")

    # construct P, q, A, b, G, h matrices for CVXOPT
    P = cvxopt.matrix(np.outer(y, y) * K)
    q = cvxopt.matrix(np.ones(n_samples) * -1)
    A = cvxopt.matrix(y, (1, n_samples))
    b = cvxopt.matrix(0.0)
    G = cvxopt.matrix(np.vstack((np.diag(np.ones(n_samples) * -1),
np.identity(n_samples))))
    h = cvxopt.matrix(np.hstack((np.zeros(n_samples), np.ones(n_samples) *
C)))
    # solve QP problem
    cvxopt.solvers.options['show_progress'] = False
    solution = cvxopt.solvers.qp(P, q, G, h, A, b)
    # Lagrange multipliers
    a = np.ravel(solution['x'])
    # Support vectors have non zero lagrange multipliers
    sv = a > 1e-5  # some small threshold

    ind = np.arange(len(a))[sv]
    a = a[sv]
    sv_x = X[sv]
    sv_y = y[sv]
    numbers_of_sv=len(sv_y)
    # Bias (For linear it is the intercept):
    bias = 0
    for n in range(len(a)):
        # For all support vectors:
        bias += sv_y[n]
        bias -= np.sum(a * sv_y * K[ind[n], sv])
    bias = bias / (len(a)+0.0001)

    if kernel_type == 'linear':
```

```
        w = np.zeros(n_features)
        for n in range(len(a)):
            w += a[n] * sv_y[n] * sv_x[n]
    else:
        w = None

    y_pred=0
    if w is not None:
        y_pred = np.sign(np.dot(X_t, w) + bias)
    else:
        y_predict = np.zeros(len(X_t))
        for i in range(len(X_t)):
            s = 0
            for a1, sv_y1, sv1 in zip(a ,sv_y, sv_x):
                # a : Lagrange multipliers, sv : support vectors.
                # Hypothesis: sign(sum^S a * y * kernel + b)

                if kernel_type == 'linear':
                    s += a1 * sv_y1 * linear_kernel(X_t[i], sv1)
                if kernel_type=='RBF':
                    s += a1 * sv_y1 * gaussian_kernel(X_t[i], sv1,
RBF_params)   # Kernel trick.
                if kernel_type == 'polynomial':
                    s += a1 * sv_y1 * polynomial_kernel(X_t[i], sv1,
poly_params[0], poly_params[1])
                if kernel_type == 'sigmoid':
                    s=+ a1 * sv_y1 *sigmoid_kernel( X_t[i],  sv1,
sigmoid_params[0], sigmoid_params[1])
            y_predict[i] = s
        y_pred = np.sign(y_predict + bias)

    return w, bias, solution,a, sv_x, sv_y, y_pred, kernel_and_params



def multiclass_svm(X, X_t, y, C, kernel_type, poly_params=(1, 4),
RBF_params=0.5, sigmoid_params=(1, 0.01)):
    class_labels = list(set(y))

    classifiers = {}
    w_catch = {}  # catching w, b only for plot part
    b_catch = {}
    a_catch = {}
    sv_x_catch = {}
    sv_y_catch = {}
```

```
    for i, class_label in enumerate(class_labels):
        binary_y = np.where(y == class_label, 1.0, -1.0)
        w, bias, solution, a, sv_x, sv_y, prediction, kernel_and_params =
SVM_scratch(X, X_t, binary_y, C, kernel_type, poly_params, RBF_params,
sigmoid_params)
        classifiers[class_label] = (w, bias, a, sv_x, sv_y,
kernel_and_params)
        w_catch[class_label] = w
        b_catch[class_label] = bias
        a_catch[class_label] = a
        sv_x_catch[class_label] = sv_x
        sv_y_catch[class_label] = sv_y

    def decision_function(X_t):
        decision_scores = np.zeros((X_t.shape[0], len(class_labels)))
        for i, label in enumerate(class_labels):
            w, bias, a, sv_x, sv_y, kernel_and_params = classifiers[label]
            if w is not None:
                decision_scores[:, i] = np.dot(X_t, w) + bias
            else:
                decision_values = np.zeros(X_t.shape[0])
                for j in range(X_t.shape[0]):
                    s = 0
                    for a1, sv_y1, sv1 in zip(a, sv_y, sv_x):
                        if kernel_type == 'linear':
                            s += a1 * sv_y1 * linear_kernel(X_t[j], sv1)
                        elif kernel_type == 'RBF':
                            s += a1 * sv_y1 * gaussian_kernel(X_t[j], sv1,
RBF_params)
                        elif kernel_type == 'polynomial':
                            s += a1 * sv_y1 * polynomial_kernel(X_t[j],
sv1, poly_params[0], poly_params[1])
                        elif kernel_type == 'sigmoid':
                            s += a1 * sv_y1 * sigmoid_kernel(X_t[j], sv1,
sigmoid_params[0], sigmoid_params[1])
                    decision_values[j] = s
                decision_scores[:, i] = decision_values + bias
        return np.argmax(decision_scores, axis=1), kernel_and_params,
w_catch, b_catch, classifiers

    return decision_function(X_t)
```

```
def visualize_multiclass_classification1(X_train, y_train1, kernel_type,
trainset, classifiers, class_labels, w_stack, b_stack,save_path_scr,
epsilon=1e-10):
    plt.figure(figsize=(6, 4))
    for i, target_name in enumerate(class_labels):
        plt.scatter(X_train[y_train1 == i, 0], X_train[y_train1 == i, 1],
label=target_name)

    if kernel_type == 'linear':
        for i in range(len(class_labels)):
            w = w_stack[i]
            bias = b_stack[i]
            x_points = np.linspace(np.min(X_train[:, 0]) - 1,
np.max(X_train[:, 0]) + 1, 200)
            y_points = -(w[0] / (w[1] + epsilon)) * x_points - bias /
(w[1] + epsilon)
            plt.plot(x_points, y_points, c='r', label='Decision Boundary')

    elif kernel_type == 'polynomial':
        x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
        y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
        xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
np.linspace(y_min, y_max, 200))
        Z = np.zeros(xx.shape)
        for i in range(len(class_labels)):
            Z = np.zeros(xx.shape)
            for j in range(xx.shape[0]):
                for k in range(xx.shape[1]):
                    sample_point = np.array([xx[j, k], yy[j, k]])
                    decision_value = 0
                    w, bias, a, sv_x, sv_y, kernel_and_params =
classifiers[i]
                    for a1, sv_y1, sv1 in zip(a, sv_y, sv_x):
                        decision_value += a1 * sv_y1 *
polynomial_kernel(sample_point, sv1, C=kernel_and_params[1][0],
d=kernel_and_params[1][1])
                    decision_value += bias
                    Z[j, k] = decision_value
            plt.contour(xx, yy, Z, levels=[0], colors='r')

    if trainset:
        plt.title('Data Points')
    else:
        plt.title('Data Points on Test Set')
```

```
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.legend()
    plt.xlim(np.min(X_train[:, 0]) - 1, np.max(X_train[:, 0]) + 1)
    plt.ylim(np.min(X_train[:, 1]) - 1, np.max(X_train[:, 1]) + 1)
    plt.savefig(save_path_scr)
    plt.show()
```

در این بخش از کد،ابتدا کدهایی برای تعریف هسته‌های مدل آمده است.انواع هسته‌های خطی،چندجمله‌ای و ... .

در بخش بعد تابع توابع مربوط به ساختن مدل SVM تعریف می‌شوند.این توابع به گونه‌ای تعریف می‌شوند که ابزارهایی نظر fit و predict و ... که در توابع آماده وجود داشتند،به ما بدهند.در قسمت آخر هم تابعی برای نمایش خروجی‌ها و نتایج و مرزهای تصمیم‌گیری تعریف شده که ما به این کد،خاصیت ذخیره‌کردن تصاویر را هم اضافه کردیم تا بتوانیم از این تصاویر در ساخت گیف استفاده کنیم.

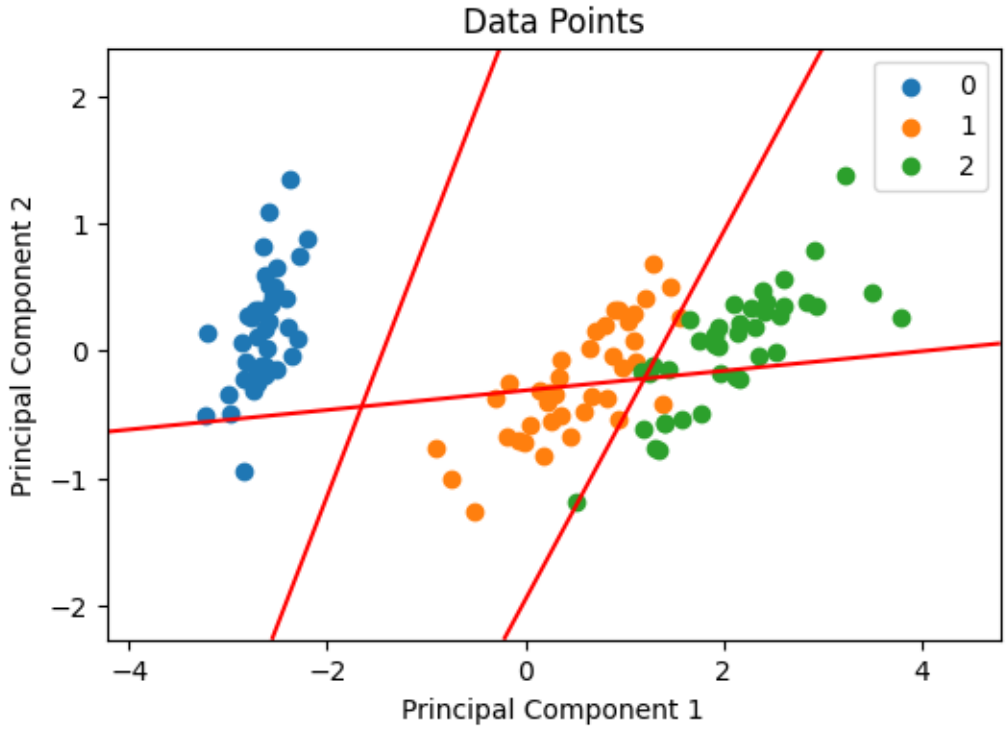
```
accuracies = []
from sklearn.metrics import accuracy_score
for degree in range(1, 11):
    print(f"Training with polynomial degree {degree}")
    predictions, kernel_and_params, w_catch, b_catch, classifiers =
multiclass_svm(
        X_reduced_train, X_reduced_test, y_train, C=1.0,
kernel_type='polynomial', poly_params=(1.0, degree)
    )
    accuracy = accuracy_score(y_test, predictions)
    accuracies.append(accuracy)
    print(f"Degree: {degree}, Accuracy: {accuracy}")

    visualize_multiclass_classification1(X_reduced_train, y_train,
'polynomial', True, classifiers, np.unique(y_train), w_catch, b_catch ,
save_path_scr = f'polynum_scratch_degree{degree}')
```

در این قسمت از کد،مدل به ازای درجه‌های مختلف آموزش دیده و پیش‌بینی انجام می‌دهد و مانند قسمت قبل در هر مرحله دقت محاسبه شده و به یک لیست اضافه می‌شود.همچنین در

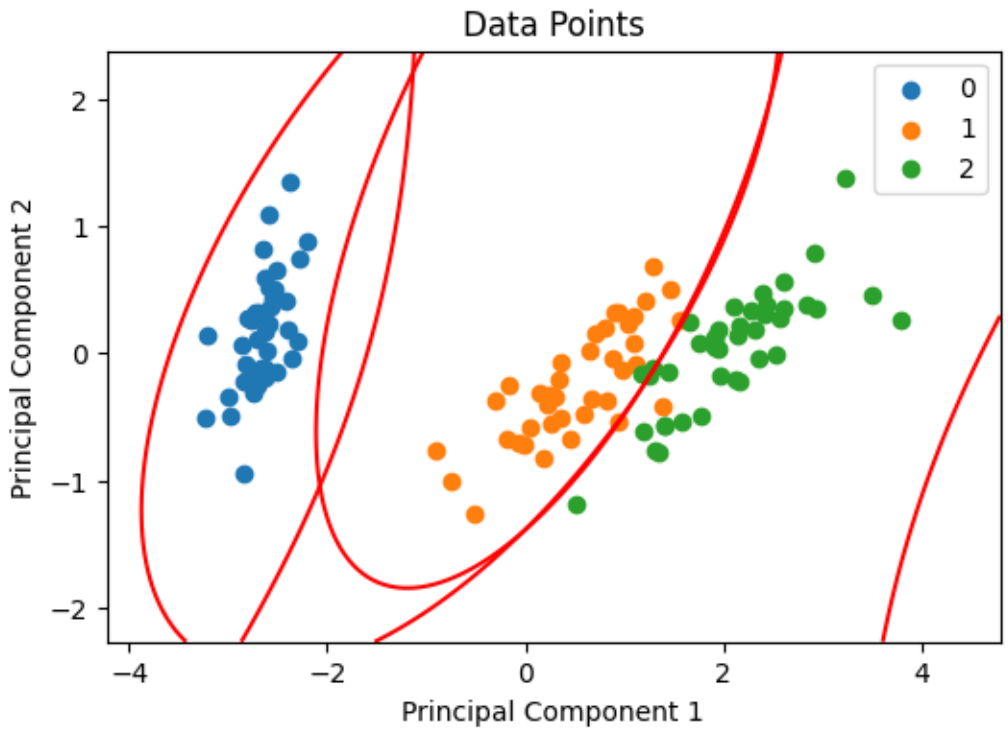
قسمت آخر،تصاویر مربوط به مرزهای تصمیم‌گیری رسم شده و همچنین ذخیره می‌شوند تا در ساخت گیف مورداستفاده قرار گیرند.

نتیجه به صورت زیر است:

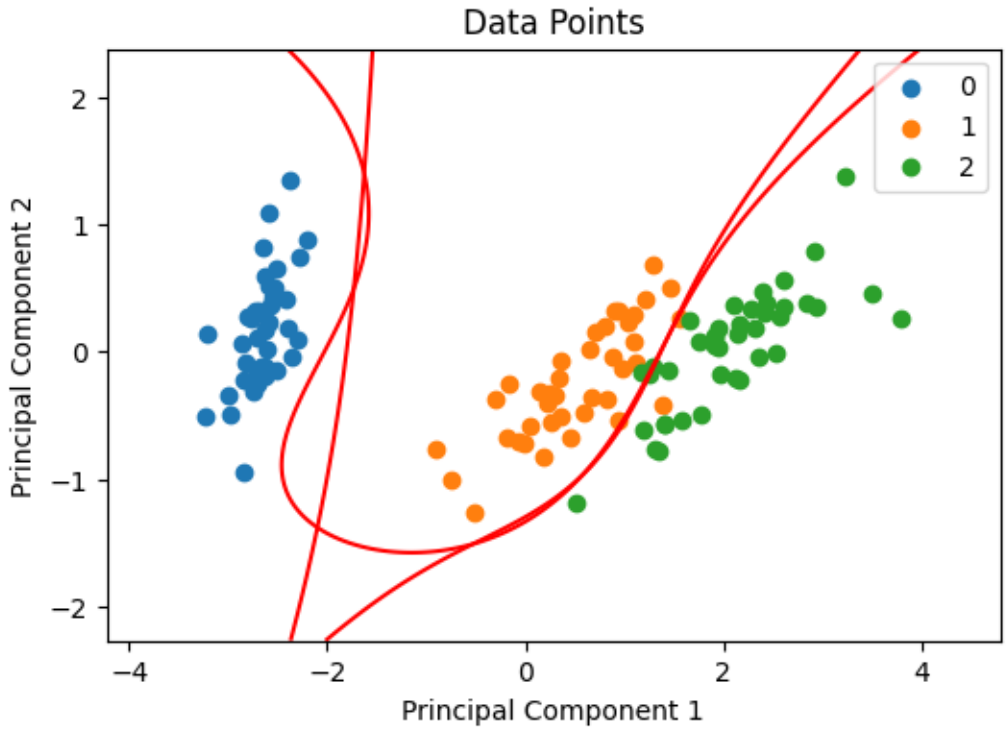
```
Training with polynomial degree 1
Degree: 1, Accuracy: 0.9666666666666667
```



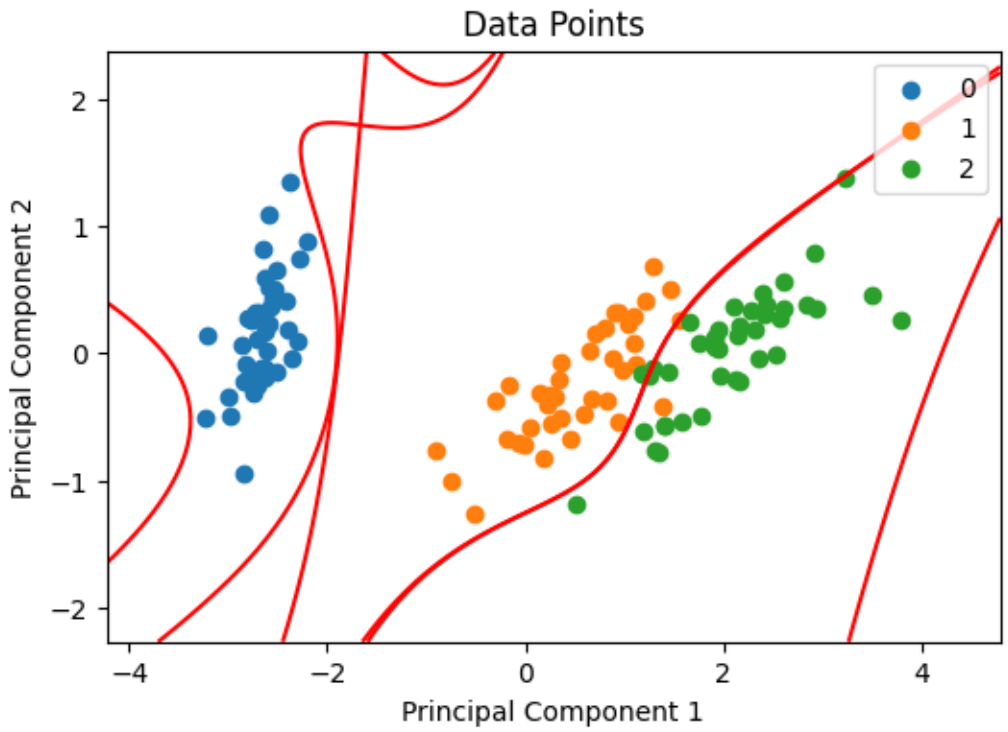

```
Training with polynomial degree 2
Degree: 2, Accuracy: 0.9666666666666667
```



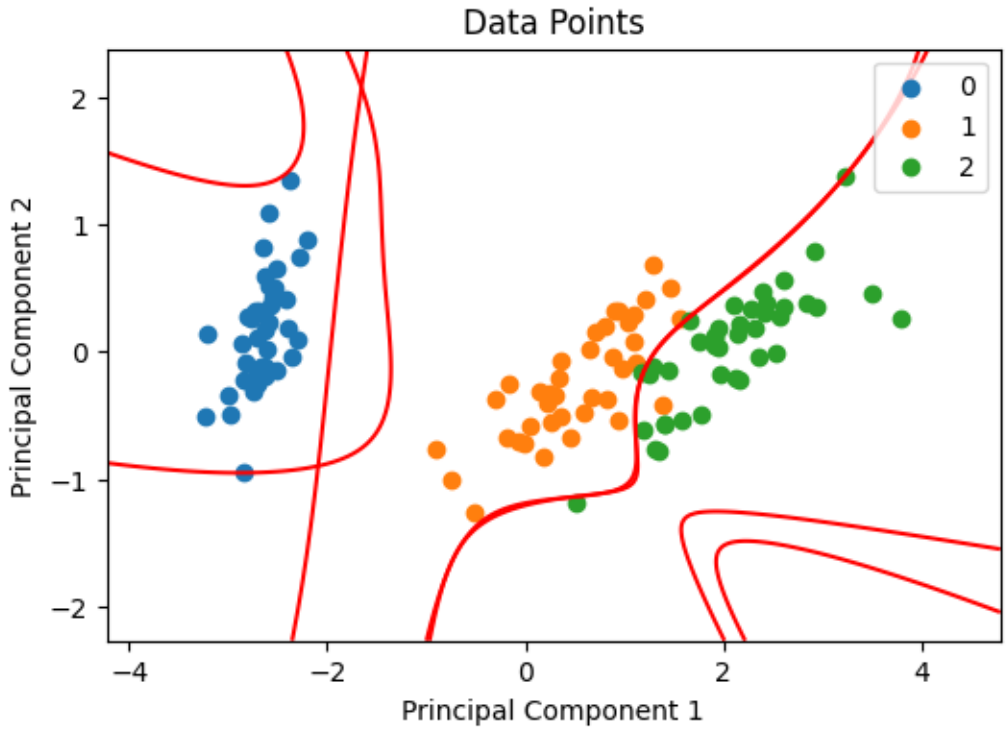

```
Training with polynomial degree 3
Degree: 3, Accuracy: 0.9666666666666667
```



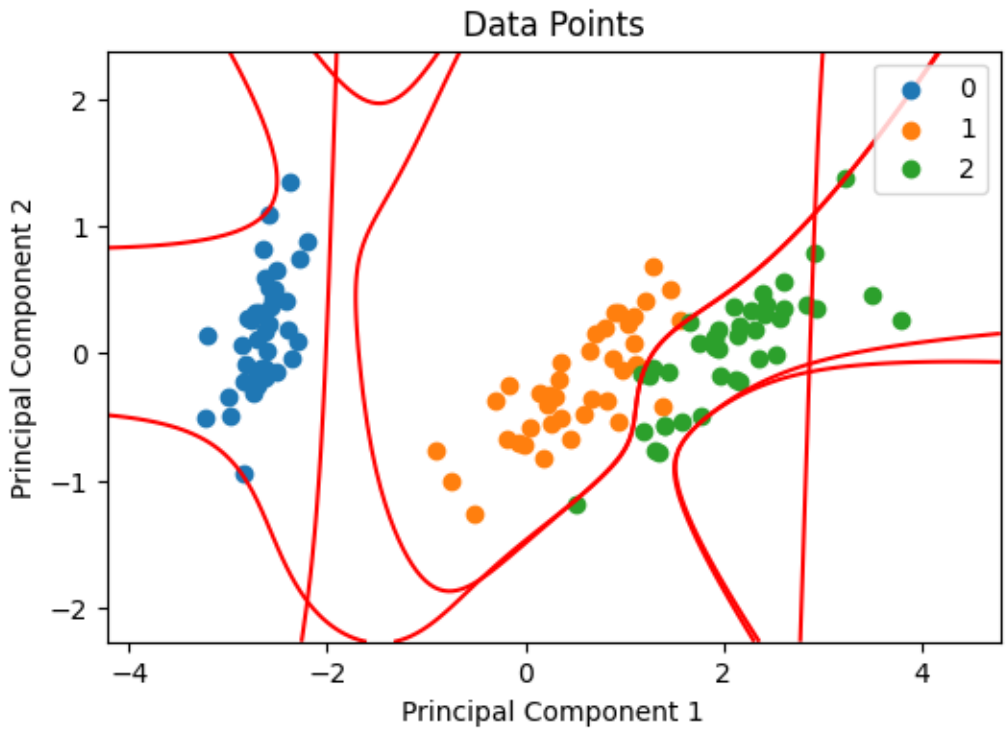

```
Training with polynomial degree 4
Degree: 4, Accuracy: 0.9666666666666667
```



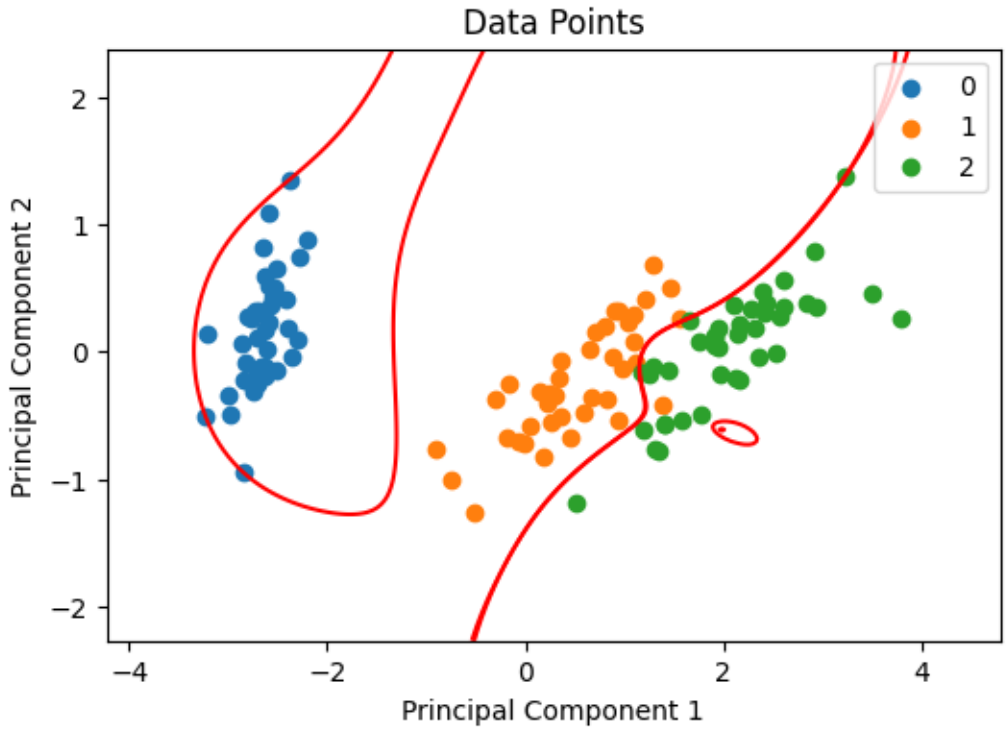

```
Training with polynomial degree 5
Degree: 5, Accuracy: 0.9666666666666667
```



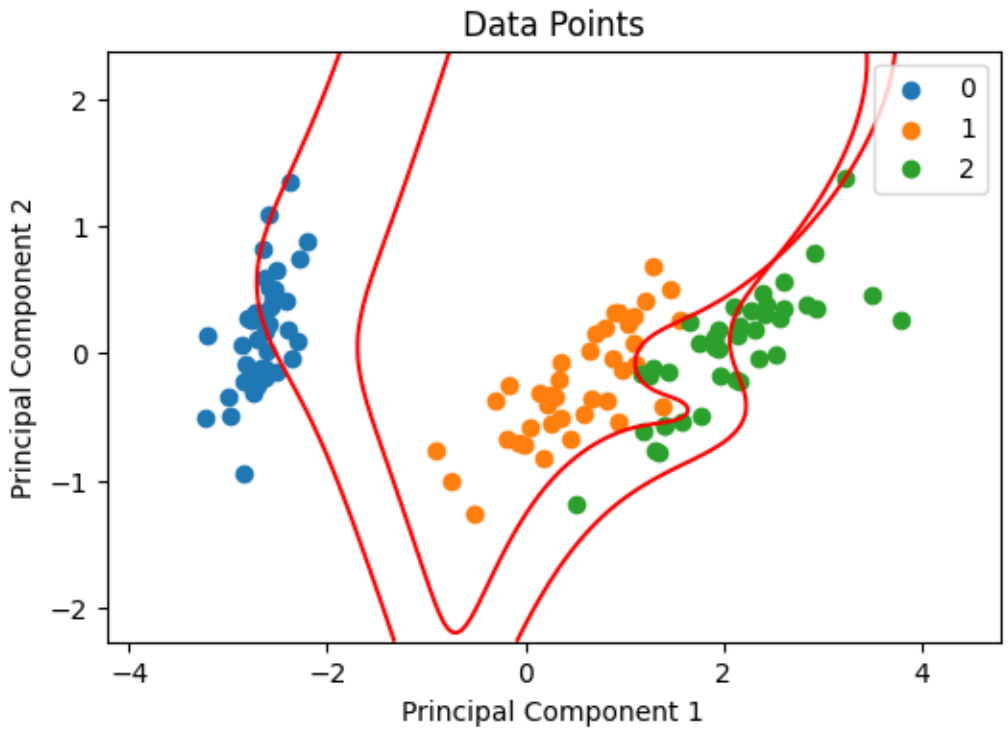

```
Training with polynomial degree 6
Degree: 6, Accuracy: 0.9333333333333333
```



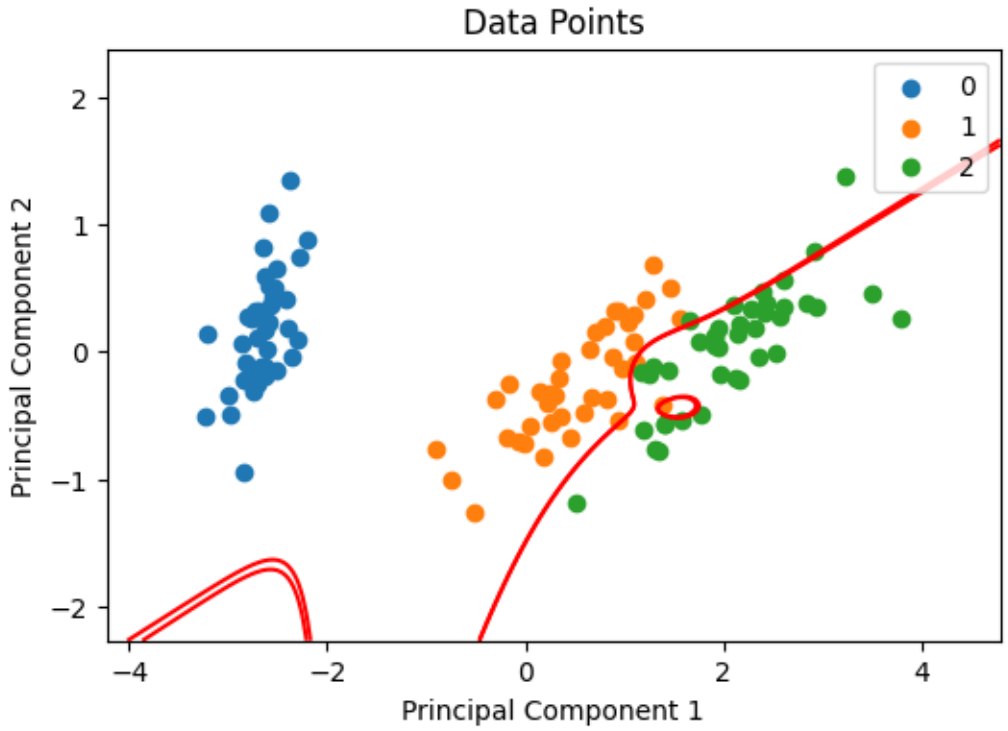

```
Training with polynomial degree 7
Degree: 7, Accuracy: 0.9
```



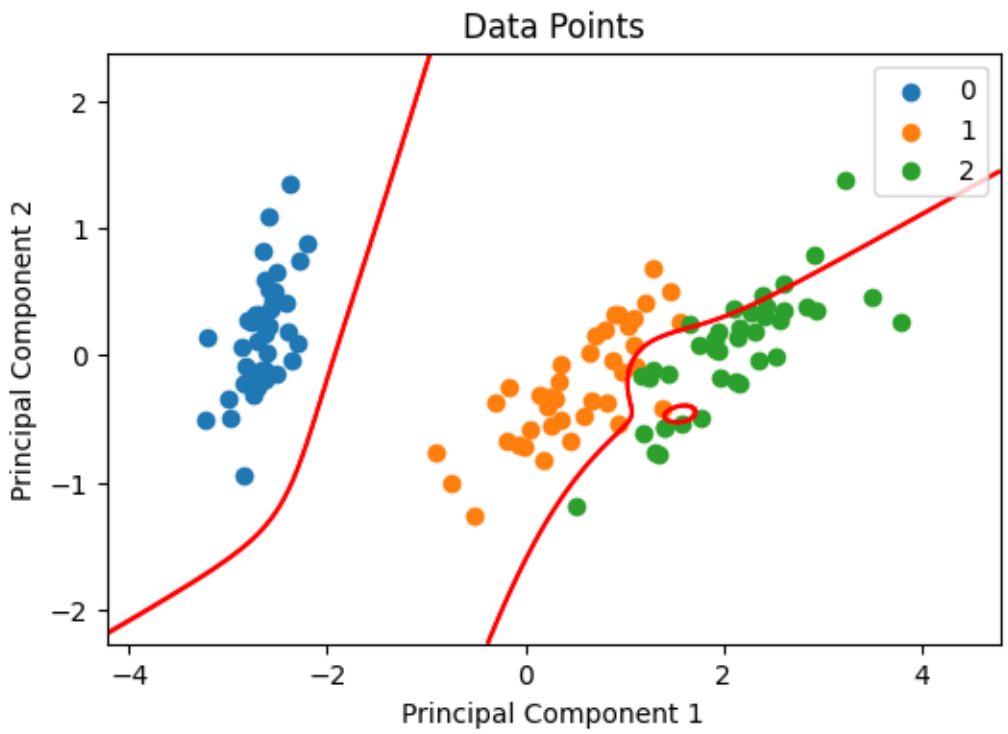

```
Training with polynomial degree 8
Degree: 8, Accuracy: 0.7
```




```
Training with polynomial degree 9
Degree: 9, Accuracy: 0.5333333333333333
```




```
Training with polynomial degree 10
Degree: 10, Accuracy: 0.5666666666666667
```



همان‌طور که می‌بینیم،دقت در درجه‌های پایین ۱ تا ۵ بالا بوده است اما هر چه درجه‌ی مدل بالاتر می‌رود،دقت کمتر می‌شود.

قسمت بعد کد به صورت زیر است:

```
import imageio

images_scr = []
for degree in range(1, 11):
    images_scr.append(imageio.imread(f'/content/polynum_scratch_degree{deg
ree}.png'))
imageio.mimsave('svm_poly_scratch_kernels.gif', images_scr, duration=1)
print("GIF saved as 'svm_poly_scratch_kernels.gif'")
```

در این کد،تصاویر ذخیره‌شده از مرزهای تصمیم‌گیری،به صورت یک گیف با زمان یک ثانیه ذخیره می‌شوند.

این گیف از این لینک قابل دسترس است.

قسمت آخر کد:

```
accuracy_scr = []
for degree , acc in results:
  accuracy_scr.append(acc)
for i in range(10):
  print(f'accuracy in degree{i+1}:\n sklearn_model:{accuracy_scr[i]} and
our model:{accuracies[i]}')
```

در این قسمت از کد،دقت‌های به دست آمده از هر یک از مدل‌ها به ازای درجه‌های ۱ تا ۱۰ مقایسه می‌شوند.

خروجی به صورت زیر است:

```
accuracy in degree1:
 sklearn_model:0.9666666666666667 and our model:0.9666666666666667
accuracy in degree2:
 sklearn_model:0.8733333333333333 and our model:0.9666666666666667
accuracy in degree3:
 sklearn_model:0.9466666666666667 and our model:0.9666666666666667
accuracy in degree4:
 sklearn_model:0.8333333333333334 and our model:0.9666666666666667
accuracy in degree5:
 sklearn_model:0.9066666666666666 and our model:0.9666666666666667
accuracy in degree6:
 sklearn_model:0.7733333333333333 and our model:0.9333333333333333
accuracy in degree7:
 sklearn_model:0.9 and our model:0.9
accuracy in degree8:
 sklearn_model:0.7666666666666667 and our model:0.7
accuracy in degree9:
 sklearn_model:0.88 and our model:0.5333333333333333
accuracy in degree10:

 sklearn_model:0.74 and our model:0.5666666666666667
```

همان‌طور که مشاهده می‌شود،دقت برای درجه‌های پایین در مدل دستی بیشتر از مدل آماده‌ی استفاده شده sklearn است.اما هر چه درجه بالاتر می‌رود،دقت مدل آماده بیشتر از مدل دستی است.

سوال ۳)

الف)

آ. بزرگ‌ترین چالش‌ها در توسعهٔ مدل‌های تشخیص تقلب چیست؟ این مقاله برای حل این چالش‌ها از چه روش‌هایی استفاده کرده است؟

در توسعه‌ی مدل‌های تشخیص تقلب چالش‌هایی وجود دارد که به صورت زیر است:

۱.متوازن نبودن داده‌ها: در اغلب دیتاهای تشخیص،تعداد داده‌هایی که کلاس تقلب هستند خیلی کمتر از داده‌های کلاس غیر تقلب هستند و این امر امری طبیعی است.

۲.سختی تشخیص داده‌های تقلب: معمولا داده‌های تقلب به گونه‌ای هستند که خیلی شبیه به داده‌های فاقد تقلب هستند.به بیانی دیگر،تقلب معمولا به گونه‌ای انجام می‌شود که تشخیص داده نشود و شبیه حالت بدون تقلب باشد.در نتیجه،تشخیص داده‌های تقلب سخت خواهد بود.

روش‌های حل این چالش‌ها:

۱.استفاده از روش oversampling: در این روش،داده‌هایی بیشتر از کلاسی که دارای داده‌های کمتری است تولید می‌شود تا به نوعی توازن در داده‌ها به وجود بیاید.

۲.استفاده از Denoising Autoencoder یا (DAE): می‌توان از یک شبکه‌ی Autoencoder برای حذف نویز و بازسازی داده‌های تمیز استفاده کرد تا به بهبود قابلیت تعمیم کمک کند.

ب)

ب. در مورد معماری شبکهٔ ارائه‌شده در مقاله به‌صورت مختصر توضیح دهید.

دو شبکه‌ی اصلی در این شبکه استفاده شده است که به صورت زیر هستند:

Denoising Autoencoder: همان‌طور که گفته شد،یکی از روش‌های حل چالش‌های این دیتاست،استفاده از DAE است.پس اولین شبکه‌ی طراحی شده در مدل کلی ما،همین شبکه است.

این شبکه شامل ۷ لایه می‌باشد که داده‌های نویزدار را در ورودی دریافت و آن‌ها تمیز می‌کند.لایه‌ی ورودی شامل ۲۹ نورون است.همچنین لایه‌های میانی یا مخفی مدل،به ترتیب دارای ۲۲ نورون،۱۵ نورون،۱۰ نورون،۱۵ نورون،۲۲ نورون و در نهایت ۲۹ نورون است که همان تعداد نورون‌های ورودی است.در واقع هدف تمیزکردن داده‌ها از نویز است.همچنین تابع هزینه که در این شبکه مورد استفاده قرار می‌گیرد،Square Loss Function می‌باشد.

Classifier: این شبکه برای طبقه‌بندی داده‌های موردنظر در نظر گرفته می‌شود.این شبکه دارای ۶ لایه است.۲۹ نورون برای لایه‌ی ورودی در نظر گرفته می‌شود.چون ورودی این شبکه خروجی همان شبکه‌ی DAE می‌باشد.همچنین لایه‌های مخفی به ترتیب دارای ۲۲ نورون،۱۵ نورون،۱۰ نورون،۵ نورون و در نهایت ۲ نورون هستند که این ۲ نورون به این دلیل است که مسئله‌ی طبقه‌بندی ما دو کلاسه می‌باشد.همچنین تابع هزینه در این شبکه، SoftMax Cross Entropy Loss Function می‌باشد.

ج)

ج. مدل ارائه‌شده 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.

کد به صورت زیر است:

قسمت اول:

```
!gdown 1n5RV7u62SjBK5-Degrx_DqyvHaB0agiJ
data = pd.read_csv('/content/creditcard.csv')
```

```
X = data.drop(columns=['Time', 'Class'])
y = data['Class']


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


X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=74, stratify=y)


sm = SMOTE(random_state=74)
X_res, y_res = sm.fit_resample(X_train, y_train)
```

در قسمت اول از کد،ابتدا دیتاست موردنظر از درایو فراخوانی می‌شود.سپس برای ساخت دیتا،ستون‌های Time و Class از این دیتاست حذف می‌شوند.همچین ستون Class به عنوان برچسب در نظر گرفته می‌شود.پس از آن دیتا با StandardScaler نرمال‌سازی می‌شود و به دو دسته‌ی آموزش و ارزیابی با نسبت ۸ به ۲ تقسیم می‌شود.همچنین در نهایت از تابع Smote استفاده می‌شود تا برای جبران تفاوت زیاد داده‌های دو کلاس،داده‌سازی انجام شود.

قسمت دوم کد:

```
dae = Sequential([
    Input(shape=(X_res.shape[1],)),
    GaussianNoise(0.1),
    Dense(22, activation='relu'),
    Dense(15, activation='relu'),
    Dense(10, activation='relu'),
    Dense(15, activation='relu'),
    Dense(22, activation='relu'),
    Dense(X_res.shape[1], activation='sigmoid')
])

dae.compile(optimizer='adam', loss='mean_squared_error')


es = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
```

```
mc = ModelCheckpoint('dae_best_model.h5', monitor='val_loss',
save_best_only=True)


dae.fit(X_res, X_res, epochs=100, batch_size=256, validation_split=0.2,
callbacks=[es, mc])
```

در این قسمت مدلی برای حذف نویز طبق توضیحات قسمت‌های قبل طراحی شده است.همچنین حالتی برای این مدل در نظر گرفته شده که در طول آموزش،اگر مقادیر هزینه تا ۵ مرحله تغییر محسوسی نداشتند،فرایند آموزش متوقف شود.

همچنین خروجی به صورت زیر است:

```
Epoch 1/100
1422/1422 [==============================] - 9s 5ms/step - loss: 8.6344 - val_loss: 21.2328
Epoch 2/100
 19/1422 [.....] - ETA: 8s - loss: 9.1066
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
1422/1422 [==============================] - 6s 4ms/step - loss: 8.5170 - val_loss: 21.2228
Epoch 3/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.5056 - val_loss: 21.2196
Epoch 4/100
1422/1422 [==============================] - 7s 5ms/step - loss: 8.4965 - val_loss: 21.2164
Epoch 5/100
1422/1422 [==============================] - 5s 4ms/step - loss: 8.4837 - val_loss: 21.2093
Epoch 6/100
1422/1422 [==============================] - 8s 5ms/step - loss: 8.4748 - val_loss: 21.2055
Epoch 7/100
1422/1422 [==============================] - 5s 4ms/step - loss: 8.4692 - val_loss: 21.1972
Epoch 8/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4639 - val_loss: 21.1954
Epoch 9/100
1422/1422 [==============================] - 7s 5ms/step - loss: 8.4614 - val_loss: 21.1934
Epoch 10/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4583 - val_loss: 21.1935
Epoch 11/100
1422/1422 [==============================] - 8s 6ms/step - loss: 8.4567 - val_loss: 21.1920
Epoch 12/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4560 - val_loss: 21.1903
Epoch 13/100
1422/1422 [==============================] - 7s 5ms/step - loss: 8.4548 - val_loss: 21.1885
Epoch 14/100
1422/1422 [==============================] - 6s 5ms/step - loss: 8.4534 - val_loss: 21.1884
Epoch 15/100
1422/1422 [==============================] - 5s 4ms/step - loss: 8.4528 - val_loss: 21.1878
Epoch 16/100
```

```
1422/1422 [==============================] - 8s 5ms/step - loss: 8.4521 - val_loss: 21.1873
Epoch 17/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4514 - val_loss: 21.1870
Epoch 18/100
1422/1422 [==============================] - 8s 5ms/step - loss: 8.4503 - val_loss: 21.1871
Epoch 19/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4492 - val_loss: 21.1863
Epoch 20/100
1422/1422 [==============================] - 7s 5ms/step - loss: 8.4476 - val_loss: 21.1862
Epoch 21/100
1422/1422 [==============================] - 7s 5ms/step - loss: 8.4455 - val_loss: 21.1854
Epoch 22/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4446 - val_loss: 21.1853
Epoch 23/100
1422/1422 [==============================] - 8s 5ms/step - loss: 8.4440 - val_loss: 21.1854
Epoch 24/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4436 - val_loss: 21.1845
Epoch 25/100
1422/1422 [==============================] - 8s 5ms/step - loss: 8.4431 - val_loss: 21.1845
Epoch 26/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4428 - val_loss: 21.1847
Epoch 27/100
1422/1422 [==============================] - 6s 5ms/step - loss: 8.4425 - val_loss: 21.1840
Epoch 28/100
1422/1422 [==============================] - 7s 5ms/step - loss: 8.4421 - val_loss: 21.1844
Epoch 29/100
1422/1422 [==============================] - 8s 5ms/step - loss: 8.4416 - val_loss: 21.1839
Epoch 30/100
1422/1422 [==============================] - 7s 5ms/step - loss: 8.4413 - val_loss: 21.1838
Epoch 31/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4409 - val_loss: 21.1834
Epoch 32/100
1422/1422 [==============================] - 8s 5ms/step - loss: 8.4404 - val_loss: 21.1835
Epoch 33/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4399 - val_loss: 21.1829
Epoch 34/100
1422/1422 [==============================] - 7s 5ms/step - loss: 8.4394 - val_loss: 21.1832
Epoch 35/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4393 - val_loss: 21.1832
Epoch 36/100
1422/1422 [==============================] - 6s 4ms/step - loss: 8.4385 - val_loss: 21.1840
Epoch 37/100
1422/1422 [==============================] - 7s 5ms/step - loss: 8.4381 - val_loss: 21.1833
Epoch 38/100
1422/1422 [==============================] - 5s 4ms/step - loss: 8.4376 - val_loss: 21.1833
```

همان‌طور که می‌بینیم،با وجود این که تعداد تکرار ۱۰۰ بوده است،اما در مرحله‌ی ۳۸ این فرایند متوقف شده است.(به خاطر تنظیماتی که انجام شده است)

قسمت سوم کد:

```
X_res_denoised = dae.predict(X_res)
X_test_denoised = dae.predict(X_test)


classifier = Sequential([
    Input(shape=(X_res_denoised.shape[1],)),
    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='sparse_categorical_crossentropy', metrics=['accuracy'])


es_clf = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
mc_clf = ModelCheckpoint('classifier_best_model.h5', monitor='val_loss',
save_best_only=True)


classifier.fit(X_res_denoised, y_res, epochs=100, batch_size=256,
validation_split=0.2, callbacks=[es_clf, mc_clf])
```

در این قسمت،ابتدا توسط مدلی که قبلا آموزش دیده شد،داده‌ها نویزگیری می‌شوند.سپس مدل طبقه‌بند مطابق آن چه در مورد ساختار آن گفته شد،طراحی می‌شود.سپس مدل روی این دیتا آموزش می‌بیند که نتیجه به صورت زیر است:(گفتنی است که تنظیمات مربوط به توقف آموزش که در مرحله‌ی قبل انجام شد،در این جا هم صورت گرفته است)

```
14216/14216 [==============================] - 29s 2ms/step
1781/1781 [==============================] - 4s 2ms/step
Epoch 1/100
1422/1422 [==============================] - 7s 4ms/step - loss: 0.1451 - accuracy: 0.9462 - val_loss:
0.0981 - val_accuracy: 0.9634
Epoch 2/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0599 - accuracy: 0.9782 - val_loss:
0.0606 - val_accuracy: 0.9771
Epoch 3/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0470 - accuracy: 0.9828 - val_loss:
0.0451 - val_accuracy: 0.9840
```

```
Epoch 4/100
1422/1422 [==============================] - 6s 5ms/step - loss: 0.0411 - accuracy: 0.9851 - val_loss:
0.0326 - val_accuracy: 0.9898
Epoch 5/100
1422/1422 [==============================] - 5s 4ms/step - loss: 0.0370 - accuracy: 0.9867 - val_loss:
0.0473 - val_accuracy: 0.9839
Epoch 6/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0347 - accuracy: 0.9877 - val_loss:
0.0290 - val_accuracy: 0.9921
Epoch 7/100
1422/1422 [==============================] - 7s 5ms/step - loss: 0.0323 - accuracy: 0.9885 - val_loss:
0.0430 - val_accuracy: 0.9852
Epoch 8/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0308 - accuracy: 0.9891 - val_loss:
0.0376 - val_accuracy: 0.9888
Epoch 9/100
1422/1422 [==============================] - 5s 4ms/step - loss: 0.0295 - accuracy: 0.9895 - val_loss:
0.0290 - val_accuracy: 0.9921
Epoch 10/100
1422/1422 [==============================] - 7s 5ms/step - loss: 0.0285 - accuracy: 0.9898 - val_loss:
0.0253 - val_accuracy: 0.9936
Epoch 11/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0275 - accuracy: 0.9903 - val_loss:
0.0248 - val_accuracy: 0.9940
Epoch 12/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0270 - accuracy: 0.9904 - val_loss:
0.0305 - val_accuracy: 0.9918
Epoch 13/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0263 - accuracy: 0.9906 - val_loss:
0.0337 - val_accuracy: 0.9911
Epoch 14/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0254 - accuracy: 0.9910 - val_loss:
0.0529 - val_accuracy: 0.9813
Epoch 15/100
1422/1422 [==============================] - 7s 5ms/step - loss: 0.0249 - accuracy: 0.9913 - val_loss:
0.0274 - val_accuracy: 0.9915
Epoch 16/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0249 - accuracy: 0.9913 - val_loss:
0.0366 - val_accuracy: 0.9883
<keras.src.callbacks.History at 0x7830cb77e920>
```

در نهایت در مرحله‌ی آموزش،مدل به دقت ۹۹ درصد رسیده است.همچنین آموزش در مرحله‌ی ۱۶ام متوقف شده است.

د)

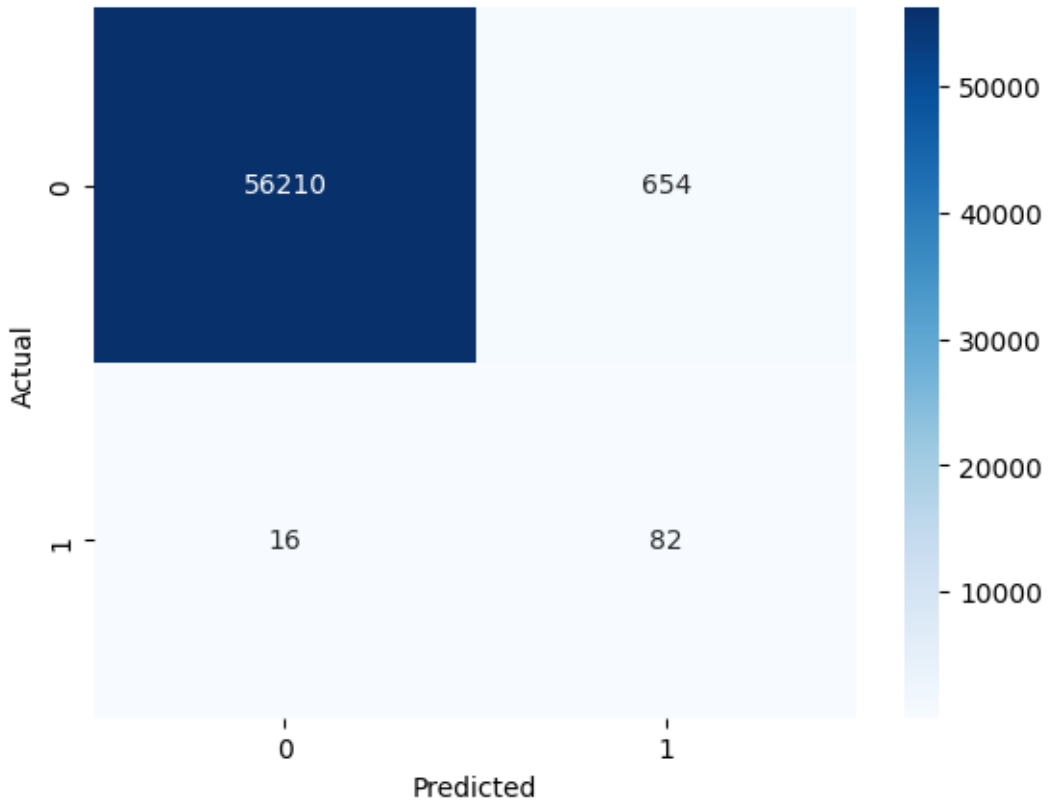
د. ماتریس درهم‌ریختگی را روی قسمت آزمون داده‌ها رسم کنید و مقادیر Accuracy، Precision، Recall و f1score را گزارش کنید. فکر می‌کنید در مسائلی که توزیع برچسب‌ها نامتوازن است، استفاده از معیاری مانند Accuracy به تنهایی عملکرد مدل را به‌درستی نمایش می‌دهد؟ چرا؟ اگر نه، کدام معیار می‌تواند به‌عنوان مکمل استفاده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

کد به صورت زیر است:

```
y_pred_prob = classifier.predict(X_test_denoised)
y_pred = np.argmax(y_pred_prob, axis=1)

cm = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)


print(f'Confusion Matrix:\n{cm}')
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')


import seaborn as sns
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

در این کد،ابتدا پیش‌بینی با مدل ما انجام می‌شود.سپس ماتریس درهم‌ریختگی برای این داده‌های ارزیابی رسم می‌شود.در نهایت معیارهایی مانند accuracy،Precision،Recall و F1 Score محاسبه می‌شوند.خروجی به صورت زیر است:

```
Confusion Matrix:
[[56210   654]
 [   16    82]]
Accuracy: 0.9882377725501211
Precision: 0.11141304347826086
Recall: 0.8367346938775511
F1 Score: 0.19664268585131894
```



همان‌طور که مشاهده می‌شود،در این دیتا مجموعا ۹۶ داده از کلاس ۱ وجود داشته است.اما در پیش‌بینی ۶۵۴ داده از کلاس ۱ پیش‌بینی شده اند.در حالی که در واقعیت از کلاس صفر بوده اند.هر چند مدل در پیش‌بینی داده‌های کلاس صفر عملکرد خوبی داشته است.در این حالت است که معیارهایی مانند Precision و F1-Score دید خوبی به ما می‌دهند.در این حالات،استفاده از معیارهایی مانند accuracy به تنهایی،معیار خوبی نخواهد بود و Precision و F1-Score برای ما مکمل خواهند بود.

در واقع به علت زیاد بودن تعداد داده‌های کلاس صفر و پیش‌بینی تعداد درستی از آن‌ها،مقدار accuracy خیلی بالاست.اما در این معیار،دقت در تشخیص داده‌های کلاس ۱ که خیلی کمتر هستند،در واقع در نظر گرفته نمی‌شود و دو معیار بالا در این زمینه ما را کمک می‌کنند.

ه)

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

کد به صورت زیر است:

```
thresholds = [0.2, 0.3, 0.4, 0.5, 0.6]
accuracies = []
recalls = []

for threshold in thresholds:
    classifier.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    classifier.fit(X_res_denoised, y_res, epochs=100, batch_size=256,
validation_split=0.2, callbacks=[es_clf, mc_clf])

    y_pred_prob = classifier.predict(X_test_denoised)[:, 1]
    y_pred_threshold = (y_pred_prob >= threshold).astype(int)

    accuracy = accuracy_score(y_test, y_pred_threshold)
    recall = recall_score(y_test, y_pred_threshold)

    accuracies.append(accuracy)
    recalls.append(recall)

plt.plot(thresholds, accuracies, label='Accuracy')
plt.plot(thresholds, recalls, label='Recall')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.legend()
plt.show()
```

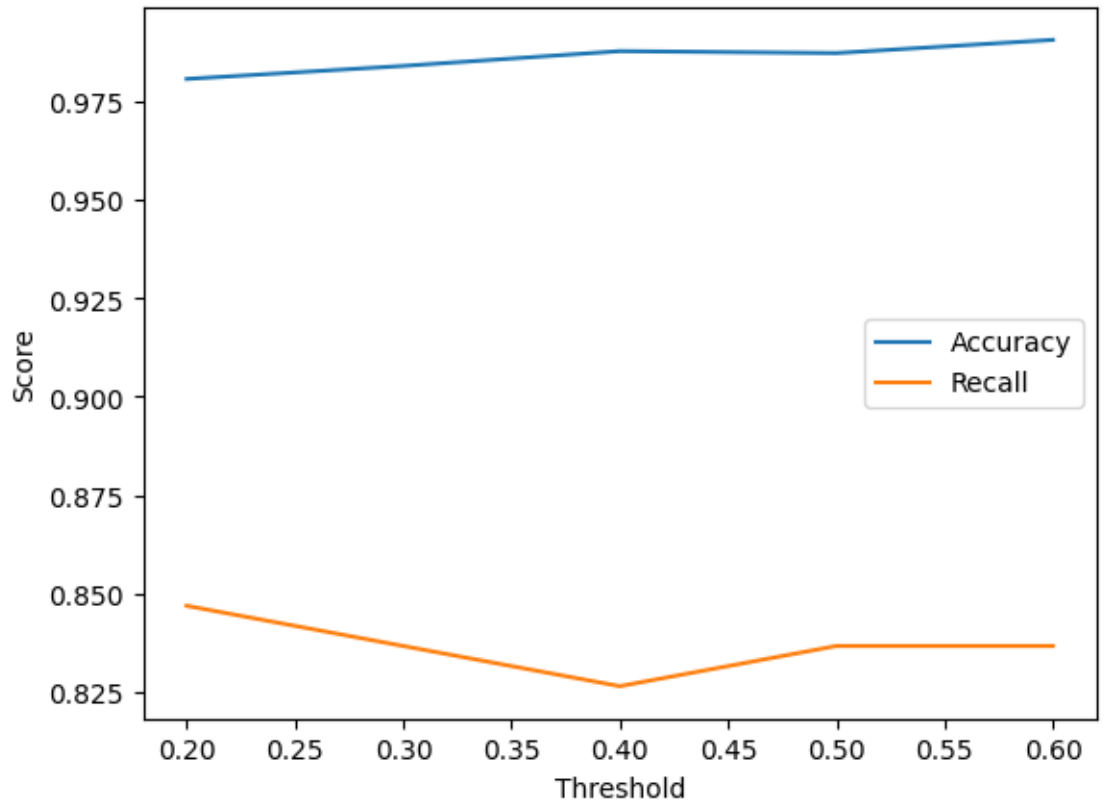
در این کد آستانه‌های مختلف برای oversampling مدل در نظر گرفته شده است و خروجی به صورت زیر است:

```
Epoch 1/100
1422/1422 [==============================] - 8s 5ms/step - loss: 0.0243 -
accuracy: 0.9914 - val_loss: 0.0201 - val_accuracy: 0.9953
Epoch 2/100
1422/1422 [==============================] - 5s 4ms/step - loss: 0.0236 -
accuracy: 0.9917 - val_loss: 0.0334 - val_accuracy: 0.9898
Epoch 3/100
```

```
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0230 -
accuracy: 0.9921 - val_loss: 0.0145 - val_accuracy: 0.9976
Epoch 4/100
1422/1422 [==============================] - 7s 5ms/step - loss: 0.0228 -
accuracy: 0.9920 - val_loss: 0.0377 - val_accuracy: 0.9885
Epoch 5/100
1422/1422 [==============================] - 5s 4ms/step - loss: 0.0227 -
accuracy: 0.9921 - val_loss: 0.0276 - val_accuracy: 0.9923
Epoch 6/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0223 -
accuracy: 0.9923 - val_loss: 0.0184 - val_accuracy: 0.9962
Epoch 7/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0221 -
accuracy: 0.9924 - val_loss: 0.0155 - val_accuracy: 0.9972
Epoch 8/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0216 -
accuracy: 0.9925 - val_loss: 0.0195 - val_accuracy: 0.9961
1781/1781 [==============================] - 3s 2ms/step
Epoch 1/100
1422/1422 [==============================] - 8s 5ms/step - loss: 0.0230 -
accuracy: 0.9920 - val_loss: 0.0422 - val_accuracy: 0.9860
Epoch 2/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0226 -
accuracy: 0.9922 - val_loss: 0.0171 - val_accuracy: 0.9967
Epoch 3/100
1422/1422 [==============================] - 5s 4ms/step - loss: 0.0219 -
accuracy: 0.9924 - val_loss: 0.0186 - val_accuracy: 0.9960
Epoch 4/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0219 -
accuracy: 0.9926 - val_loss: 0.0108 - val_accuracy: 0.9979
Epoch 5/100
  56/1422 [>.............................] - ETA: 3s - loss: 0.0252 -
accuracy: 0.9919
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the native
Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
1422/1422 [==============================] - 4s 3ms/step - loss: 0.0219 -
accuracy: 0.9925 - val_loss: 0.0150 - val_accuracy: 0.9973
Epoch 6/100
1422/1422 [==============================] - 5s 4ms/step - loss: 0.0215 -
accuracy: 0.9926 - val_loss: 0.0159 - val_accuracy: 0.9967
Epoch 7/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0209 -
accuracy: 0.9928 - val_loss: 0.0335 - val_accuracy: 0.9902
Epoch 8/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0210 -
accuracy: 0.9928 - val_loss: 0.0231 - val_accuracy: 0.9944
Epoch 9/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0207 -
accuracy: 0.9929 - val_loss: 0.0115 - val_accuracy: 0.9983
1781/1781 [==============================] - 4s 2ms/step
Epoch 1/100
1422/1422 [==============================] - 6s 3ms/step - loss: 0.0218 -
accuracy: 0.9925 - val_loss: 0.0167 - val_accuracy: 0.9968
Epoch 2/100
```

```
1422/1422 [==============================] - 7s 5ms/step - loss: 0.0213 -
accuracy: 0.9928 - val_loss: 0.0170 - val_accuracy: 0.9961
Epoch 3/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0213 -
accuracy: 0.9927 - val_loss: 0.0237 - val_accuracy: 0.9945
Epoch 4/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0204 -
accuracy: 0.9931 - val_loss: 0.0333 - val_accuracy: 0.9899
Epoch 5/100
1422/1422 [==============================] - 7s 5ms/step - loss: 0.0207 -
accuracy: 0.9930 - val_loss: 0.0285 - val_accuracy: 0.9918
Epoch 6/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0206 -
accuracy: 0.9929 - val_loss: 0.0259 - val_accuracy: 0.9932
1781/1781 [==============================] - 3s 2ms/step
Epoch 1/100
1422/1422 [==============================] - 7s 4ms/step - loss: 0.0215 -
accuracy: 0.9926 - val_loss: 0.0128 - val_accuracy: 0.9975
Epoch 2/100
1422/1422 [==============================] - 5s 4ms/step - loss: 0.0213 -
accuracy: 0.9927 - val_loss: 0.0217 - val_accuracy: 0.9945
Epoch 3/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0209 -
accuracy: 0.9929 - val_loss: 0.0187 - val_accuracy: 0.9961
Epoch 4/100
1422/1422 [==============================] - 7s 5ms/step - loss: 0.0206 -
accuracy: 0.9930 - val_loss: 0.0123 - val_accuracy: 0.9975
Epoch 5/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0205 -
accuracy: 0.9931 - val_loss: 0.0377 - val_accuracy: 0.9868
Epoch 6/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0202 -
accuracy: 0.9932 - val_loss: 0.0104 - val_accuracy: 0.9980
Epoch 7/100
  64/1422 [>.............................] - ETA: 3s - loss: 0.0193 -
accuracy: 0.9932
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the native
Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
1422/1422 [==============================] - 7s 5ms/step - loss: 0.0200 -
accuracy: 0.9932 - val_loss: 0.0151 - val_accuracy: 0.9968
Epoch 8/100
1422/1422 [==============================] - 4s 3ms/step - loss: 0.0202 -
accuracy: 0.9931 - val_loss: 0.0378 - val_accuracy: 0.9874
Epoch 9/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0198 -
accuracy: 0.9934 - val_loss: 0.0186 - val_accuracy: 0.9962
Epoch 10/100
1422/1422 [==============================] - 7s 5ms/step - loss: 0.0198 -
accuracy: 0.9933 - val_loss: 0.0251 - val_accuracy: 0.9925
Epoch 11/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0194 -
accuracy: 0.9935 - val_loss: 0.0258 - val_accuracy: 0.9929
1781/1781 [==============================] - 3s 2ms/step
Epoch 1/100
```

```
1422/1422 [==============================] - 8s 5ms/step - loss: 0.0204 -
accuracy: 0.9931 - val_loss: 0.0170 - val_accuracy: 0.9966
Epoch 2/100
1422/1422 [==============================] - 5s 3ms/step - loss: 0.0199 -
accuracy: 0.9933 - val_loss: 0.0114 - val_accuracy: 0.9978
Epoch 3/100
1422/1422 [==============================] - 5s 4ms/step - loss: 0.0196 -
accuracy: 0.9935 - val_loss: 0.0244 - val_accuracy: 0.9939
Epoch 4/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0196 -
accuracy: 0.9934 - val_loss: 0.0147 - val_accuracy: 0.9973
Epoch 5/100
1422/1422 [==============================] - 4s 3ms/step - loss: 0.0197 -
accuracy: 0.9934 - val_loss: 0.0132 - val_accuracy: 0.9975
Epoch 6/100
1422/1422 [==============================] - 5s 4ms/step - loss: 0.0193 -
accuracy: 0.9936 - val_loss: 0.0282 - val_accuracy: 0.9924
Epoch 7/100
1422/1422 [==============================] - 6s 4ms/step - loss: 0.0190 -
accuracy: 0.9936 - val_loss: 0.0168 - val_accuracy: 0.9963
1781/1781 [==============================] - 3s 2ms/step
```



همان‌طور که می‌بینیم،به ازای آستانه‌های مختلف مقادیر Accuracy و Recall تغییرات زیادی نداشته‌اند.

و)

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.

کد به صورت زیر است:

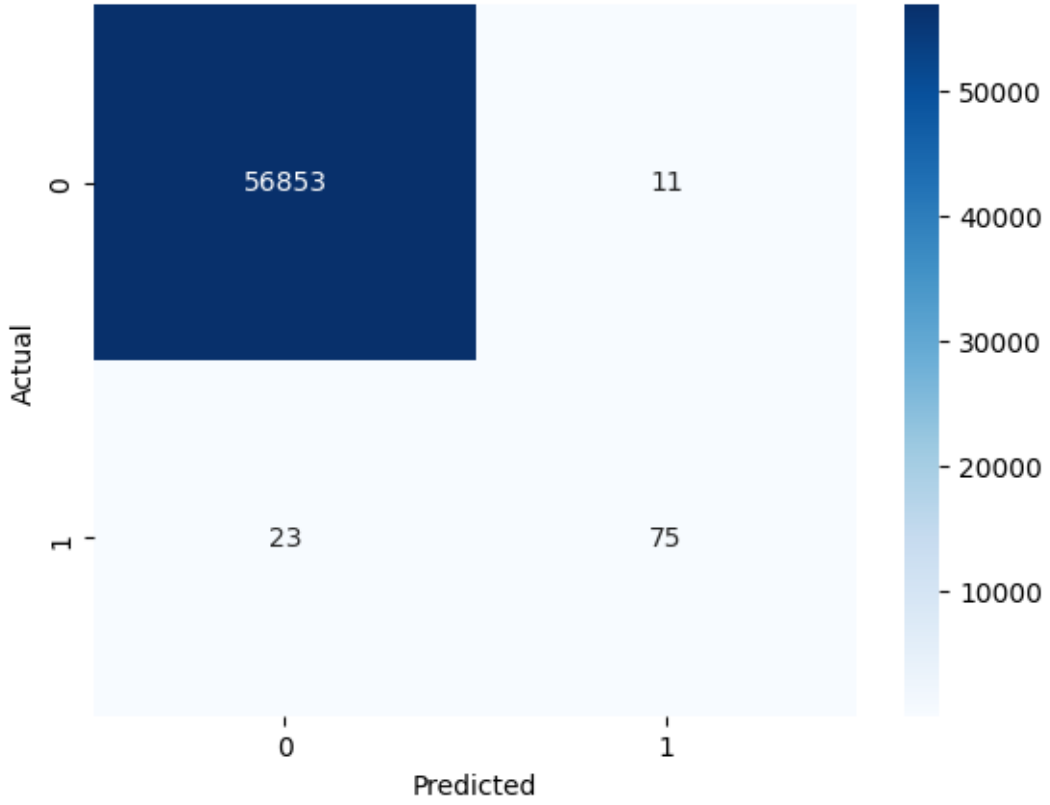
```
classifier_unbalanced = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(22, activation='relu'),
    Dense(15, activation='relu'),
    Dense(10, activation='relu'),
    Dense(5, activation='relu'),
    Dense(2, activation='softmax')
])

classifier_unbalanced.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])


classifier_unbalanced.fit(X_train, y_train, epochs=100, batch_size=256,
validation_split=0.2, callbacks=[es_clf, mc_clf])


y_pred_prob_unbalanced = classifier_unbalanced.predict(X_test)
y_pred_unbalanced = np.argmax(y_pred_prob_unbalanced, axis=1)

cm_unbalanced = confusion_matrix(y_test, y_pred_unbalanced)
accuracy_unbalanced = accuracy_score(y_test, y_pred_unbalanced)
precision_unbalanced = precision_score(y_test, y_pred_unbalanced)
recall_unbalanced = recall_score(y_test, y_pred_unbalanced)
f1_unbalanced = f1_score(y_test, y_pred_unbalanced)


print(f'Confusion Matrix (Unbalanced):\n{cm_unbalanced}')
print(f'Accuracy (Unbalanced): {accuracy_unbalanced}')
print(f'Precision (Unbalanced): {precision_unbalanced}')
```

```
print(f'Recall (Unbalanced): {recall_unbalanced}')
print(f'F1 Score (Unbalanced): {f1_unbalanced}')


sns.heatmap(cm_unbalanced, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

در این مدل داده‌ها بدون نویزگیری مورد استفاده قرار گرفته‌اند.خروجی به صورت زیر است:

```
Epoch 1/100
713/713 [==============================] - 6s 5ms/step - loss: 0.0393 -
accuracy: 0.9965 - val_loss: 0.0033 - val_accuracy: 0.9994
Epoch 2/100
713/713 [==============================] - 2s 3ms/step - loss: 0.0039 -
accuracy: 0.9994 - val_loss: 0.0032 - val_accuracy: 0.9994
Epoch 3/100
713/713 [==============================] - 2s 3ms/step - loss: 0.0034 -
accuracy: 0.9994 - val_loss: 0.0032 - val_accuracy: 0.9994
Epoch 4/100
713/713 [==============================] - 3s 4ms/step - loss: 0.0032 -
accuracy: 0.9994 - val_loss: 0.0030 - val_accuracy: 0.9994
Epoch 5/100
713/713 [==============================] - 4s 6ms/step - loss: 0.0029 -
accuracy: 0.9994 - val_loss: 0.0030 - val_accuracy: 0.9994
Epoch 6/100
713/713 [==============================] - 2s 3ms/step - loss: 0.0028 -
accuracy: 0.9994 - val_loss: 0.0031 - val_accuracy: 0.9994
Epoch 7/100
713/713 [==============================] - 2s 3ms/step - loss: 0.0026 -
accuracy: 0.9994 - val_loss: 0.0030 - val_accuracy: 0.9995
Epoch 8/100
713/713 [==============================] - 2s 3ms/step - loss: 0.0024 -
accuracy: 0.9994 - val_loss: 0.0030 - val_accuracy: 0.9994
Epoch 9/100
 57/713 [=>............................] - ETA: 1s - loss: 7.8970e-04 -
accuracy: 0.9999
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the native
Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
713/713 [==============================] - 3s 4ms/step - loss: 0.0023 -
accuracy: 0.9994 - val_loss: 0.0031 - val_accuracy: 0.9994
Epoch 10/100
713/713 [==============================] - 4s 5ms/step - loss: 0.0021 -
accuracy: 0.9995 - val_loss: 0.0031 - val_accuracy: 0.9994
Epoch 11/100
713/713 [==============================] - 3s 4ms/step - loss: 0.0020 -
accuracy: 0.9995 - val_loss: 0.0032 - val_accuracy: 0.9994
Epoch 12/100
```

```
713/713 [==============================] - 2s 3ms/step - loss: 0.0019 -
accuracy: 0.9995 - val_loss: 0.0032 - val_accuracy: 0.9994
Epoch 13/100
713/713 [==============================] - 3s 4ms/step - loss: 0.0018 -
accuracy: 0.9995 - val_loss: 0.0030 - val_accuracy: 0.9995
1781/1781 [==============================] - 5s 3ms/step
Confusion Matrix (Unbalanced):
[[56853    11]
 [   23    75]]
Accuracy (Unbalanced): 0.999403110845827
Precision (Unbalanced): 0.872093023255814
Recall (Unbalanced): 0.7653061224489796
F1 Score (Unbalanced): 0.8152173913043479
```



همان‌گونه که مشاهده می‌شود،در این حالت معیارهای Precision و F1-Score خیلی بهتر شده‌اند.دلیل این تغییر این است که داده‌هایی که دارای کلاس صفر بوده‌اند و در کلاس ۱ تشخیص داده شده‌اند،بسیار کمتر از حالت قبل هستند و دقت مدل بیشتر شده است.پس می‌بینیم معیارهایی مانند Precision و F1-Score می‌توانند در تحلیل مدل،کمک کنند و نقش مکمل در کنار accuracy داشته باشند.

در واقع به علت زیاد بودن تعداد داده‌های کلاس صفر و پیش‌بینی تعداد درستی از آن‌ها،مقدار accuracy خیلی بالاست.اما در این معیار،دقت در تشخیص داده‌های کلاس ۱ که خیلی کمتر هستند،در واقع در نظر گرفته نمی‌شود و دو معیار بالا در این زمینه ما را کمک می‌کنند.