**به نام خدا**

****

**دانشکده مهندسی برق**

**گزارش درس یادگیری ماشین**

**مقطع: کارشناسی ارشد گرایش: مهندسی کنترل**

**گزارش مینی پروژه سوم**

**توسط:**

**مرجان محمدی**

**40111534**

**استاد درس:**

**دکتر علیاری**

[لینک کولب](https://drive.google.com/drive/folders/1Rk7YX4KUpmQP9FU9-LF_Fyu7VDx1HRc-?usp=sharing)

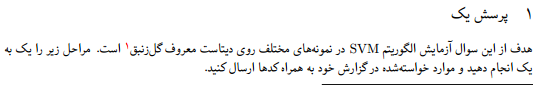
[**لینک گیت هاب**](https://github.com/marjanMohammadi1375/MachineLearning2023)

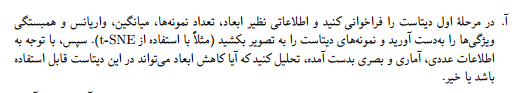
**بهار 1403**

**فهرست مطالب**

پرسش اول.....................................................................................................................................................3

پرسش سوم.................................................................................................................................................34





دیتاست Iris یکی از مشهورترین و قدیمی‌ترین دیتاست‌ها در حوزه یادگیری ماشین است که توسط رونالد فیشر در سال 1936 معرفی شد. این دیتاست شامل 150 نمونه از گل‌های زنبق (Iris) با سه گونه مختلف Iris setosa، Iris versicolor، و Iris virginica است. هر نمونه دارای چهار ویژگی (طول و عرض کاسبرگ، طول و عرض گلبرگ) می‌باشد. هدف این دیتاست، طبقه‌بندی نمونه‌ها به یکی از سه گونه زنبق با استفاده از ویژگی‌های ذکر شده است. دیتاست Iris به دلیل سادگی و استاندارد بودن، اغلب برای آموزش و ارزیابی الگوریتم‌های طبقه‌بندی استفاده می‌شود. در این سوال از دیتاست iris استفاده می کنیم.

ابتدا کتابخانه های مورد نیاز سوال را فراخوانی می کنیم:

# Importing necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import imageio

# Importing scikit-learn libraries

from sklearn.manifold import TSNE

from sklearn.decomposition import PCA

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import f1\_score, confusion\_matrix, accuracy\_score

from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load\_iris

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

# Importing additional libraries

from mpl\_toolkits.mplot3d import Axes3D

from mlxtend.plotting import plot\_decision\_regions

در ادامه دیتاست iris را با استفاده از تابع load iris فراخوانی کرده و iris.feature\_names شامل نام‌های ویژگی‌ها (ستون‌ها) در دیتاست است (مانند طول سپال (سانتیمتر) , عرض سپال (سانتیمتر) و غیره) و iris.target\_names شامل نام‌های کلاس‌های هدف (مانند setosa , versicolor, virginica است. این نام‌ها به ترتیب به feature\_names و target\_names اختصاص داده می‌شوند. قسمت ورودی(ویژگی ها) و تارگت(برچسب) را جدا می کنیم و آن ها را برای تحلیل بهتر به دیتا فریم تبدیل می کنیم. در نتیجه یک تابع به نام def print\_iris\_attributes(dataset) تعریف می کنیم که تمام ویژگی‌ها (ویژگی‌ها و متدها) را از شیء دیتاست با نادیده گرفتن ویژگی‌های شروع شده با زیرخط (\_) بازیابی و چاپ می‌کند.

در نهایت، این بخش print\_iris\_attributes(iris) را فراخوانی می‌کند که تمام ویژگی‌ها و مقادیر مربوط به آنها را از شیء دیتاست iris چاپ می‌کند.

# Load Iris dataset

iris = load\_iris()

X, y = iris.data, iris.target

feature\_names, target\_names = iris.feature\_names, iris.target\_names

# Create a DataFrame from the iris data

df = pd.DataFrame(data=X, columns=feature\_names)

# Display the shape of X and y

print(f"Feature matrix shape: {X.shape}, Target vector shape: {y.shape}")

# Function to print attributes and their values of the iris dataset

def print\_iris\_attributes(dataset):

    attributes = [attr for attr in dir(dataset) if not attr.startswith('\_')]

    for attr in attributes:

        print(f"{attr}: {getattr(dataset, attr)}")

# Print attributes and values of the iris dataset

print("\nAttributes and values of the iris dataset:")

print\_iris\_attributes(iris)

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

# Compute the correlation matrix

corr\_matrix = df.corr()

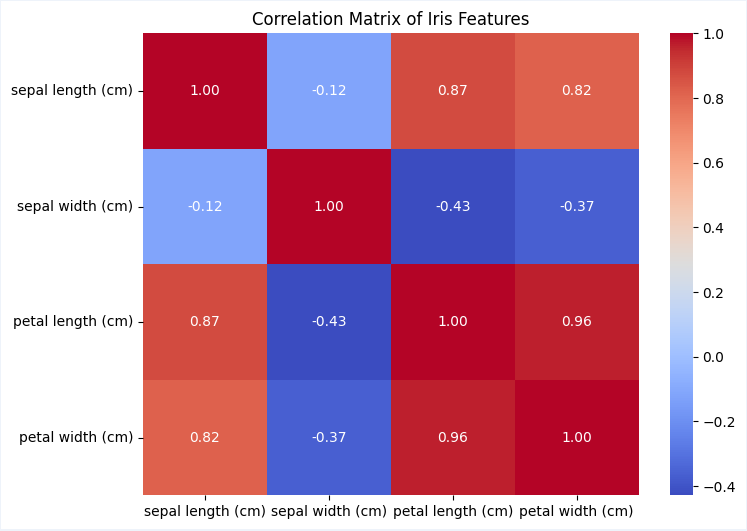
# Plot the heatmap

plt.figure(figsize=(8, 6))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', fmt='.2f')

plt.title('Correlation Matrix of Iris Features')

plt.show()



در ادامه یک ستون جدید به DataFrame df اضافه می‌شود که نام آن species است. برای ایجاد این ستون، از تابع pd.Categorical.from\_codes() استفاده می‌شود. این تابع با استفاده از آرایه y و نام‌های کلاس‌های هدف (target\_names)، یک ستون دسته‌ای از کلاس‌های هدف (نام گونه‌های گل‌ها) ایجاد می‌کند و به DataFrame اضافه می‌کند.

سپس یک پالت رنگی سفارشی تعریف می‌شود با استفاده از sns.color\_palette() و با نام "husl" که یک پالت رنگی متعلق به seaborn است. تعداد رنگ‌ها در این پالت برابر با تعداد کلاس‌های هدف (len(target\_names)) است.

در نهایت یک نمودار Pairplot از داده‌ها رسم می‌کنیم و از یک پالت رنگ سفارشی برای تفکیک رنگ‌های نمودار استفاده می‌کنیم..

# Add species column to the DataFrame

df['species'] = pd.Categorical.from\_codes(y, target\_names)

# Define a custom color palette

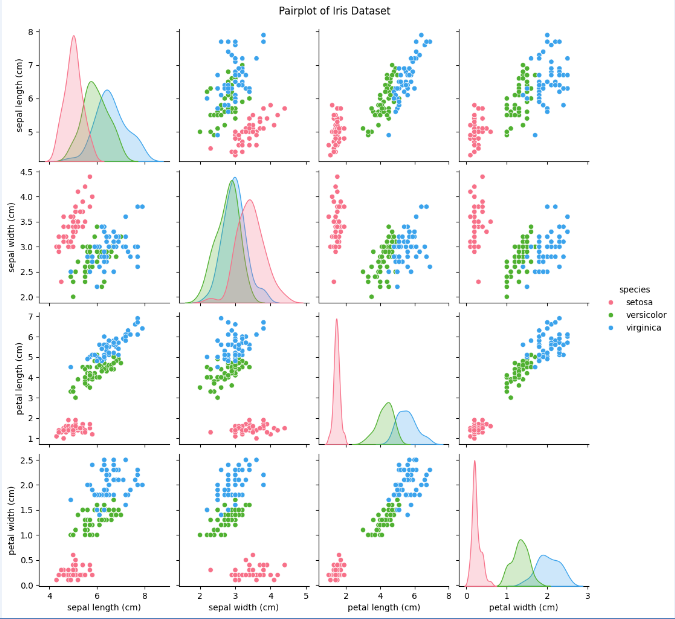
custom\_palette = sns.color\_palette("husl", len(target\_names))

# Plot the pairplot with the custom color palette

sns.pairplot(df, hue='species', palette=custom\_palette)

plt.suptitle('Pairplot of Iris Dataset', y=1.02)  # Adjust the title position

plt.show()



در ادامه ، الگوریتم t-SNE بر روی داده‌های ویژگی X اجرا می‌شود. TSNE n\_components=2, یک نمونه از کلاس TSNE ایجاد می‌کند که تعداد مؤلفه‌ها را n\_components=2 برای نمایش دو بعدی و random\_state=34 برای تولید نتایج قابل تکرار تنظیم می‌کند. سپس tsne.fit\_transform(X) داده‌های ویژگی X را به فضای دوبعدی تبدیل می‌کند و نتایج را در X\_tsne ذخیره می‌کند.(عدد رندم استیت دو رقم آخر شماره دانشجویی است)

سپس در این بخش، یک DataFrame جدید به نام df\_tsne برای داده‌های تبدیل شده با t-SNE ایجاد می‌شود. این DataFrame دارای دو ستون به نام‌های 'Component 1' و 'Component 2' است که مختصات نمونه‌ها در فضای دوبعدی t-SNE را نشان می‌دهد. همچنین ستون 'Target' به عنوان برچسب‌ها (y) و ستون 'Target Name' برای نام گونه‌های زنبق (با استفاده از target\_names) اضافه می‌شود.

به طور کلی، این کد داده‌های دیتاست Iris را با استفاده از الگوریتم t-SNE به فضای دو بعدی تبدیل کرده و نموداری از آنها رسم می‌کند که نمونه‌ها را بر اساس کلاس‌های هدف (نام گونه‌های زنبق) با رنگ‌های مختلف تفکیک می‌کند.

# Perform t-SNE

tsne = TSNE(n\_components=2, random\_state=34)

X\_tsne = tsne.fit\_transform(X)

# Create a DataFrame for the t-SNE data

df\_tsne = pd.DataFrame(X\_tsne, columns=['Component 1', 'Component 2'])

df\_tsne['Target'] = y

df\_tsne['Target Name'] = df\_tsne['Target'].apply(lambda i: target\_names[i])

# Define a custom color palette

custom\_palette = sns.color\_palette("Set2", len(target\_names))

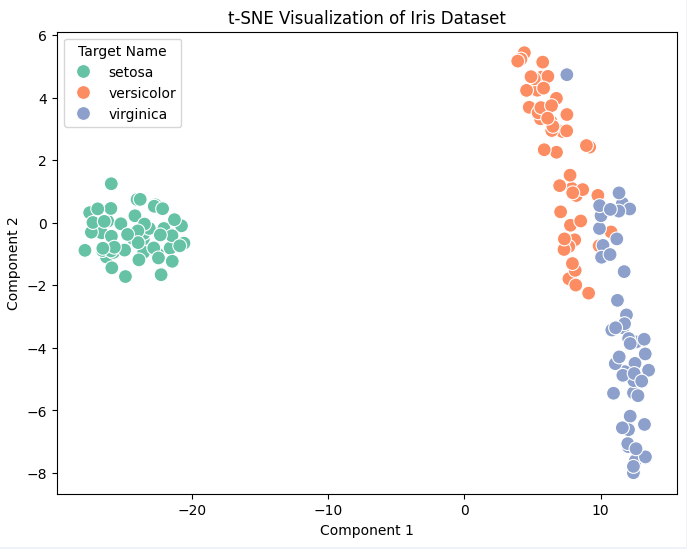
# Plot the t-SNE results with the custom color palette

plt.figure(figsize=(8, 6))

sns.scatterplot(data=df\_tsne, x='Component 1', y='Component 2', hue='Target Name', palette=custom\_palette, s=100)

plt.title('t-SNE Visualization of Iris Dataset')

plt.show()



در مرحله بعد با استفاده از PCA و LDA به کاهش بعد می پردازیم که مراحل آنها مانند t-sne می باشد.

# Perform PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

explained\_variance = pca.explained\_variance\_ratio\_

# Create a DataFrame for the PCA data

df\_pca = pd.DataFrame(X\_pca, columns=['PCA Component 1', 'PCA Component 2'])

df\_pca['Target'] = y

df\_pca['Target Name'] = df\_pca['Target'].apply(lambda i: target\_names[i])

# Define a custom color palette

custom\_palette = sns.color\_palette("viridis", len(target\_names))

# Plot the PCA results with the custom color palette

plt.figure(figsize=(8, 6))

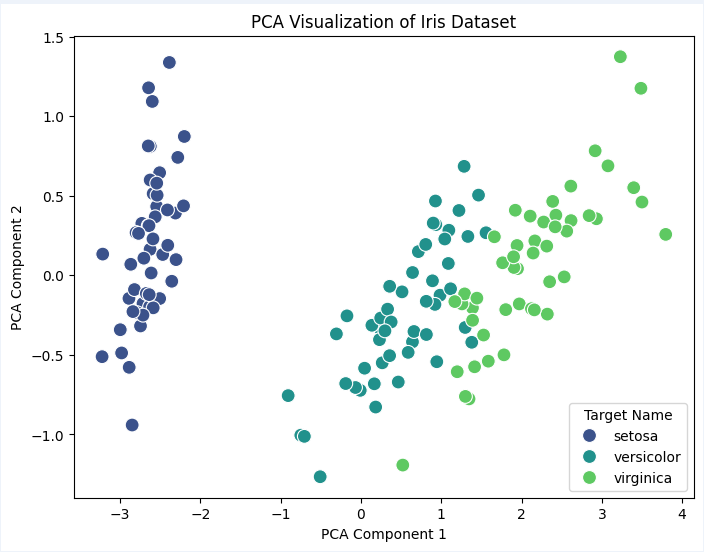
sns.scatterplot(data=df\_pca, x='PCA Component 1', y='PCA Component 2', hue='Target Name', palette=custom\_palette, s=100)

plt.title('PCA Visualization of Iris Dataset')

plt.show()

# Print the explained variance ratio

print("Explained variance ratio:", explained\_variance)



# Perform LDA

lda = LinearDiscriminantAnalysis(n\_components=2)

X\_lda = lda.fit\_transform(X, y)

# Create a DataFrame for the LDA data

df\_lda = pd.DataFrame(X\_lda, columns=['Component 1', 'Component 2'])

df\_lda['species'] = pd.Categorical.from\_codes(y, target\_names)

# Define a custom color palette

custom\_palette = sns.color\_palette("viridis", len(target\_names))

# Plot the LDA results with the custom color palette

plt.figure(figsize=(8, 6))

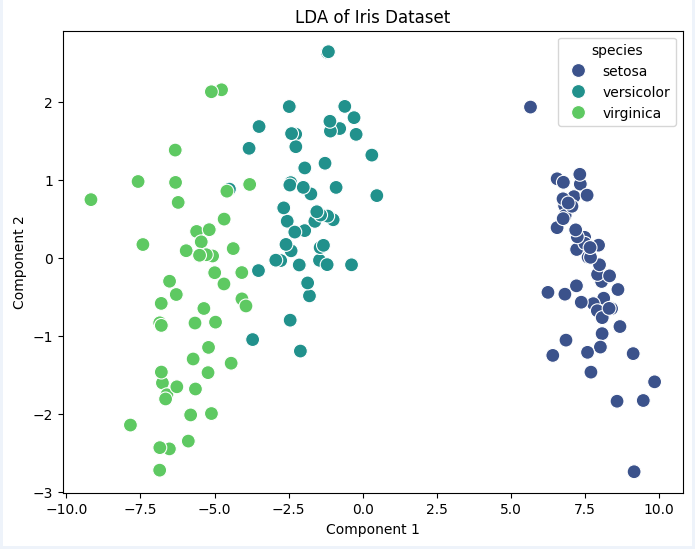
sns.scatterplot(data=df\_lda, x='Component 1', y='Component 2', hue='species', palette=custom\_palette, s=100)

plt.title('LDA of Iris Dataset')

plt.xlabel('Component 1')

plt.ylabel('Component 2')

plt.show()



کد زیر ، رویکردی است برای تغییر ساختار و تصویرسازی داده‌ها در دیتاست Iris، که با حذف برخی از ویژگی‌های اصلی و تمرکز بر دیگران انجام می‌شود. ابتدا، دو ستون مرتبط با طول گلبرگ‌ها و طول کاسبرگ‌ها حذف شده و سپس اطلاعات برچسب (گونه‌های مختلف زنبق) به DataFrame اضافه می‌شوند. این تغییرات ساختاری به داده‌ها اجازه می‌دهد که در یک نمودار پراکندگی با استفاده از کتابخانه seaborn نمایش داده شوند، که نقاط بر اساس عرض گلبرگ‌ها و عرض کاسبرگ‌ها و با رنگ‌های متفاوت برای هر گونه رسم می‌شود. نمودار نهایی با عنوان مناسب به ما نشان می‌دهد چگونه تغییرات در ویژگی‌ها و نحوه نمایش داده‌ها می‌تواند به ما درک بهتری از تفاوت‌های بین گونه‌های مختلف زنبق بدهد. این روش نه تنها برای تحلیل بلکه برای بصری سازی داده‌های پیچیده در قالبی قابل فهم مفید است.

# Drop specific columns and add target information

df\_2 = df.drop(['petal length (cm)', 'sepal length (cm)'], axis=1)

df\_2['Target'] = y

df\_2['Target Name'] = df\_2['Target'].apply(lambda i: target\_names[i])

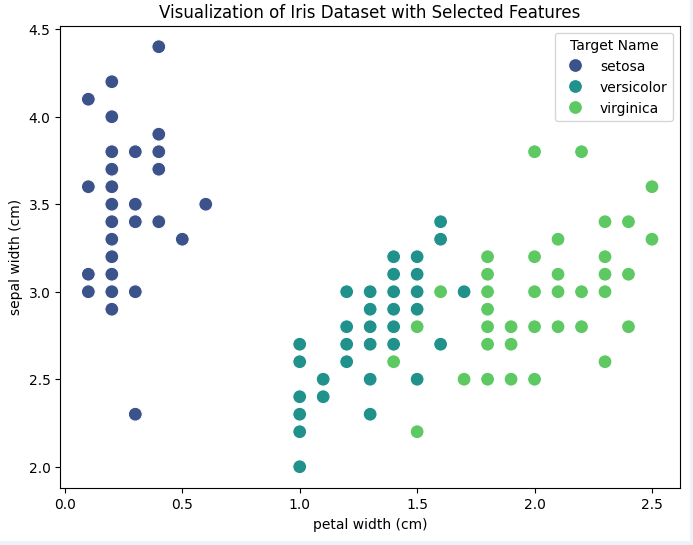
# Plot the results with the dropped columns

plt.figure(figsize=(8, 6))

sns.scatterplot(data=df\_2, x='petal width (cm)', y='sepal width (cm)', hue='Target Name', palette='viridis', s=100)

plt.title('Visualization of Iris Dataset with Selected Features')

plt.show()



در دنیای علم داده، کاهش ابعاد اغلب به عنوان یک ابزار قدرتمند برای فهم بهتر داده‌ها و کاهش پیچیدگی آن‌ها استفاده می‌شود. از آنجایی که دیتاست Iris شامل چهار ویژگی کمی است و سه کلاس متفاوت را شامل می‌شود، استفاده از تکنیک‌های کاهش ابعاد می‌تواند به ما کمک کند تا دیدگاهی واضح‌تر و مفیدتر درباره این داده‌ها به دست آوریم.

**تکنیک‌های کاهش ابعاد**

1. تحلیل مولفه‌های اصلی : PCA - این روش به ما اجازه می‌دهد تا مولفه‌هایی که بیشترین تغییرات را در داده‌ها توضیح می‌دهند شناسایی کنیم. در دیتاست Iris، PCA می‌تواند به ما نشان دهد که کدام ویژگی‌ها بیشترین اطلاعات را در مورد تفاوت‌های بین گونه‌ها دارند.

2. : t-SNE این تکنیک در کشف ساختارهای پیچیده در داده‌ها بسیار موثر است و می‌تواند به ما کمک کند تا روابط غیرخطی میان داده‌های Iris را بهتر درک کنیم.

3. تحلیل تفکیک خطی (LDA): LDA بر خلاف PCA، تمرکز خود را بر روی بیشینه کردن جدایی بین کلاس‌های مختلف قرار می‌دهد. این روش می‌تواند به ما نشان دهد که چگونه گونه‌های مختلف زنبق از نظر آماری از هم متمایز هستند.

**تصویرسازی و بینش‌های بصری**

استفاده از نمودارهای بعد کاهش یافته می‌تواند به ما کمک کند تا یک درک بصری از داده‌ها به دست آوریم. این نمودارها نه تنها زیبا هستند بلکه اطلاعات مفیدی را نیز درباره توزیع داده‌ها و روابط بین نمونه‌ها فراهم می‌کنند.

**نتیجه‌گیری**

به طور خلاصه، کاهش ابعاد می‌تواند به شدت در دیتاست Iris مفید باشد، نه تنها به خاطر توانایی آن در ساده‌سازی داده‌ها بلکه به دلیل کمک به ما در فهم بهتر و دقیق‌تر ساختار و روابط موجود در داده‌ها. این تکنیک‌ها به ما امکان می‌دهند تا با استفاده از داده‌های کمتر، اطلاعات بیشتری کسب کنیم.



در این قسمت ابتدا اسکیل دیتا ها را با استفاده از standard scaler تغییر می دهیم(نرمالایز میکنیم) سپس دیتا را به دو بخش آموزش(train) 80 درصد و آزمایش(test) 20 درصد تقسیم می کنیم.

# Scale the data

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=24)

# Display the shapes of the training and testing sets

print(f"Training feature matrix shape: {X\_train.shape}")

print(f"Testing feature matrix shape: {X\_test.shape}")

print(f"Training target vector shape: {y\_train.shape}")

print(f"Testing target vector shape: {y\_test.shape}")

سپس الگوریتم ماشین بردار پشتیبان(svm) را طبق زیر پیاده سازی می کنیم.

**ایجاد و آموزش مدل SVM:**

در این بخش، یک شیء از کلاس SVCبا استفاده از هسته خطی ایجاد می‌شود. انتخاب یک هسته خطی برای مدل نشان می‌دهد که ما انتظار داریم داده‌ها با یک مرز خطی قابل تفکیک باشند.

مدل با استفاده از داده‌های آموزشی X\_train و y\_train آموزش داده می‌شود.

**پیش‌بینی روی داده‌های تست:**

با استفاده از مدل آموزش‌دیده، پیش‌بینی‌ها روی مجموعه داده‌های تست X\_test انجام می‌شود و در y\_pred\_test ذخیره می‌شوند.

**چاپ نرخ دقت:**

دقت مدل روی داده‌های تست با استفاده از متد score محاسبه و چاپ می‌شود. این ارزیابی نشان می‌دهد که مدل چه درصدی از داده‌های تست را به درستی طبقه‌بندی کرده است.

**ماتریس درهم‌ریختگی برای داده‌های تست:**

ماتریس درهم‌ریختگی برای داده‌های تست محاسبه می‌شود که نحوه طبقه‌بندی دسته‌های مختلف توسط مدل را نشان می‌دهد. این ماتریس به تحلیل خطاهای مدل کمک می‌کند.

**چاپ پارامترهای مدل SVM:**

وزن‌ها (ضرایب) و بایاس (عرض از مبدأ) مدل چاپ می‌شوند. این پارامترها تعیین‌کننده مرز تصمیم‌گیری در فضای ویژگی هستند.

**پیش‌بینی روی کل داده‌ها:**

مدل برای پیش‌بینی روی کل داده‌ها (X\_scaled ) استفاده می‌شود. این کار به ارزیابی عملکرد مدل در کل داده‌ها کمک می‌کند.

**ماتریس درهم‌ریختگی برای کل داده‌ها:**

ماتریس درهم‌ریختگی برای کل داده‌ها محاسبه و چاپ می‌شود تا نحوه عملکرد مدل در طبقه‌بندی کل مجموعه داده‌ها نشان داده شود.

در مجموع، این کد به ارزیابی و تحلیل عملکرد یک مدل SVM با هسته خطی در طبقه‌بندی داده‌ها می‌پردازد و از ماتریس‌های درهم‌ریختگی برای نمایش توانایی مدل در تشخیص صحیح کلاس‌های مختلف استفاده می‌کند.

# Create and train the SVM model with a linear kernel

svm = SVC(kernel='linear')

svm.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred\_test = svm.predict(X\_test)

# Print the accuracy score on the test set

print(f"Accuracy on test data: {svm.score(X\_test, y\_test):.2f}")

# Calculate and print the confusion matrix for the test data

cm\_test = confusion\_matrix(y\_test, y\_pred\_test)

print("\nConfusion Matrix of test data:")

print(cm\_test)

# Print the SVM model parameters (weights and bias)

print(f"Weights: {svm.coef\_[0]}")

print(f"Bias: {svm.intercept\_[0]}")

# Make predictions on all data

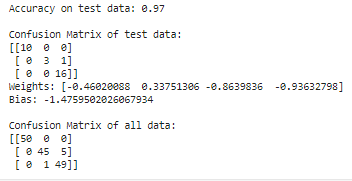
y\_pred\_all = svm.predict(X\_scaled)

# Calculate and print the confusion matrix for all data

cm\_all = confusion\_matrix(y, y\_pred\_all)

print("\nConfusion Matrix of all data:")

print(cm\_all)



حالا از PCA برای کاهش ابعاد داده‌ها استفاده کرده و سپس طبقه‌بندی داده‌ها با استفاده از ماشین بردار پشتیبانی (SVM) با هسته خطی انجام میشود. هر بخش از کد را به صورت مفصل بررسی می‌کنیم:

**کاهش ابعاد با :PCA**

PCA(n\_components=2) یک شیء PCA با دو مولفه اصلی ایجاد می‌کند. هدف این است که ابعاد دیتاست را از چهار ویژگی به دو ویژگی کاهش دهد تا تحلیل و تصویرسازی داده‌ها ساده‌تر شود.

X\_pca = pca.fit\_transform(X\_scaled) این دستور داده‌های مقیاس شده `X\_scaled` را به فضای دو بعدی تبدیل می‌کند. `X\_scaled` باید نسخه‌ای از داده‌های اصلی باشد که به منظور نرمال‌سازی اسکیل شده‌اند.

**تعریف و آموزش مدل :SVM**

SVC(kernel='linear') یک مدل SVM با هسته خطی تعریف می‌کند. هسته خطی برای مواردی که رابطه خطی بین ویژگی‌ها و برچسب‌ها وجود دارد مناسب است.

svm.fit(X\_scaled, y) این مدل SVM با داده‌های مقیاس شده و برچسب‌های مربوطه آموزش داده می‌شود.

**تعریف شبکه مش برای تصویرسازی مرزهای تصمیم:**

محاسبه محدوده‌های x و y برای شبکه مش با اضافه کردن یک واحد به حداقل و حداکثر مقادیرX\_pca

:np.meshgrid ایجاد یک شبکه دو بعدی از نقاط برای تصویرسازی مرز تصمیم SVM.

**پیش‌بینی تابع تصمیم SVM روی شبکه:**

svm.predict(pca.inverse\_transform(np.c\_[xx.ravel(), yy.ravel()])) :این دستور ابتدا نقاط شبکه را به فضای اصلی بازمی‌گرداند با (pca.inverse\_transform و سپس با استفاده از مدل SVM پیش‌بینی‌ها را انجام می‌دهد.

Z = Z.reshape(xx.shape) تغییر شکل نتایج پیش‌بینی به شکل شبکه مش برای تصویرسازی.

**تصویرسازی مرزهای تصمیم:**

plt.figure: تعیین اندازه تصویر.

:plt.contourf رسم مرزهای تصمیم با استفاده از رنگ‌ها برای نمایش مناطق مختلف.

:plt.scatter نمایش داده‌های اصلی در فضای دو بعدی PCA با رنگ‌بندی متفاوت برای هر کلاس.

این کد نه تنها برای نمایش قدرت PCA در کاهش ابعاد و SVM در طبقه‌بندی استفاده می‌شود، بلکه با ارائه تصویرسازی بصری مرزهای تصمیم، درک عمیق‌تری از چگونگی کارکرد این مدل‌ها فراهم می‌آورد.

# Reducing the four features to two using PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Define the SVM model and fit it to the scaled data

svm = SVC(kernel='linear')

svm.fit(X\_scaled, y)

# Define the mesh grid for plotting decision boundaries

x\_min, x\_max = X\_pca[:, 0].min() - 1, X\_pca[:, 0].max() + 1

y\_min, y\_max = X\_pca[:, 1].min() - 1, X\_pca[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.02), np.arange(y\_min, y\_max, 0.02))

# Predict the SVM decision function on the grid

Z = svm.predict(pca.inverse\_transform(np.c\_[xx.ravel(), yy.ravel()]))

Z = Z.reshape(xx.shape)

# Plot the decision boundaries with a custom colormap

plt.figure(figsize=(8, 6))

plt.contourf(xx, yy, Z, alpha=0.8, cmap='viridis')  # Change colormap here

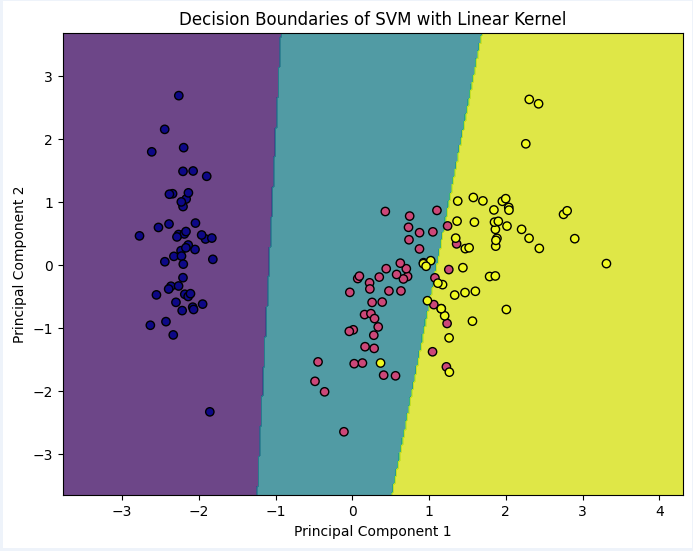
plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, edgecolors='k', cmap='plasma')  # Change colormap here

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('Decision Boundaries of SVM with Linear Kernel')

plt.show()



در ادامه دو ویژگی مهم طول گلبرگ و عرض گلبرگ را انتخاب کرده و با ایجاد یک شبکه مش برای فضای ویژگی ها، با استفاده از این دو ویژگی svm را پیاده سازی می کنیم:

# Select the two most important features for visualization

feature1\_index = 2  # Petal length in iris dataset

feature2\_index = 3  # Petal width in iris dataset

# Create a mesh grid for plotting decision boundaries using the two selected features

x\_min, x\_max = X\_scaled[:, feature1\_index].min() - 1, X\_scaled[:, feature1\_index].max() + 1

y\_min, y\_max = X\_scaled[:, feature2\_index].min() - 1, X\_scaled[:, feature2\_index].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.02), np.arange(y\_min, y\_max, 0.02))

# Predict the class for each point in the mesh grid

# Create a full feature array for prediction

Z = np.array([

    svm.predict([[

        xx.ravel()[i] if j == feature1\_index else yy.ravel()[i] if j == feature2\_index else 0

        for j in range(X.shape[1])

    ]])[0]

    for i in range(xx.ravel().shape[0])

])

Z = Z.reshape(xx.shape)

# Plot the decision boundaries and the data points

plt.figure(figsize=(8, 6))

plt.contourf(xx, yy, Z, alpha=0.8, cmap='viridis')  # Change colormap here

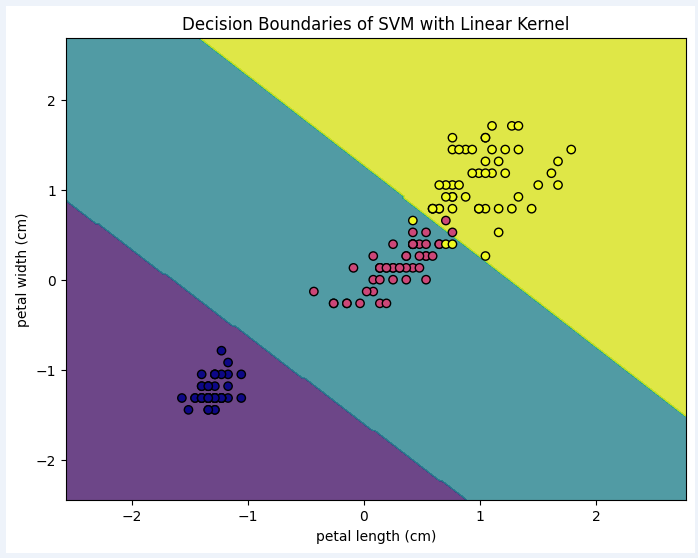
plt.scatter(X\_scaled[:, feature1\_index], X\_scaled[:, feature2\_index], c=y, cmap='plasma', edgecolors='k')  # Change colormap here

plt.xlabel(iris.feature\_names[feature1\_index])

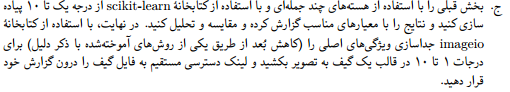
plt.ylabel(iris.feature\_names[feature2\_index])

plt.title('Decision Boundaries of SVM with Linear Kernel')

plt.show()



همانطور که مشخص است، ماشین بردار پشتیبان در همه ی بخش ها به خوبی توانسته است که به خوبی کلاس های مختلف را از هم جدا کند.



در این قسمت ابتدا کتابخانه های مورد نیاز را فراخوانی می کنیم سپس دیتاست را فراخوانی کرده و نرمالایز می کنیم سپس به قسمت آموزش با درصد 80 و قسمت آزمایش با درصد 20 تقسیم می کنیم.

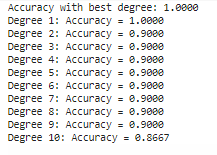
در نهایت با استفاده از PCA به کاهش بعد برای تصویر سازی می پردازیم.

سپس مدل svm با هسته ی چند جمله ای را تعریف می کنیم و برای یافتن بهترین درجه چندجمله ای از 1 تا 10 با استفاده از GridSearchCV پرداخته و بهترین مدل را انتخاب می کنیم.

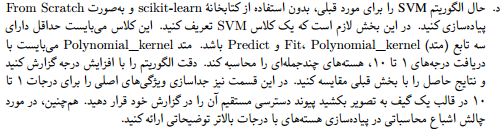
سپس مدل svm را با بهترین هسته چندجمله ای آموزش می دهیم و با استفاده از داده های تست و دقت به دست آمده به ارزیابی مدل و پیش بینی آن می پردازیم.

سپس برای درجات مختلف از 1 تا 10 مدل را آموزش میدهیم و دقت مدل را ارزیابی می کنیم.

در نتیجه یک دایرکتوری برای گیف ایجاد می کنیم و از تصاویر همه درجات GIF ایجاد می کنیم و آن را ذخیره می کنیم.



گیف با اسم q1-c در فایل موجود است.



طبق قسمت قبل ابتدا کتابخانه های مورد نیاز سوال را ایمپورت کرده و دیتاست را فراخوانی می کنیم سپس نرمالایز کرده و دیتا را شافل می کنیم و به بخش آموزش و آزمایش اسپلیت میکنیم.

در ادامه چندین تابع هسته برای SVM تعریف می‌شوند، از جمله خطی، چندجمله‌ای، گاوسی (RBF)، و سیگموئید.

# Define kernel functions

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):

    return np.tanh(alpha \* np.dot(x, y) + C)

در ادامه تابع svm1 را تعریف می کنیم:

* این تابع برای پیاده‌سازی SVM با هسته‌های مختلف (خطی، چندجمله‌ای، گاوسی و سیگموئیدی) استفاده می‌شود.
* X: ماتریس ویژگی‌های داده آموزشی.
* X\_t: ماتریس ویژگی‌های داده تست.
* y: بردار برچسب‌های کلاس داده‌های آموزشی.
* C: پارامتر جریمه برای کنترل انعطاف‌پذیری مدل.
* kernel\_type: نوع هسته (خطی، چندجمله‌ای، گاوسی یا سیگموئیدی).
* poly\_params, RBF\_params, sigmoid\_params: پارامترهای مخصوص هر نوع هسته

def SVM1(X, X\_t, y, C, kernel\_type, poly\_params=(1, 4), RBF\_params=0.5, sigmoid\_params=(1, 0.01)):

n\_samples, n\_features = X.shape

در ادامه ماتریس گرمی را محاسبه می کنیم

ماتریس گرمی (K) یک ماتریس n×n است که در آن n تعداد نمونه‌هاست. این ماتریس برای محاسبه حاصل ضرب داخلی بین نمونه‌ها در فضای ویژگی‌های هسته استفاده می‌شود.

بسته به نوع هسته، برای هر جفت نمونه (i, j)، مقدار مناسبی در ماتریس K محاسبه می‌شود.

# 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")

در ادامه ماتریس های موردنیاز برای حل مسئله برنامه ریزی مربعی را می سازیم:

این بخش ماتریس‌های مورد نیاز برای حل مسئله برنامه‌ریزی مربعی (QP) را با استفاده از کتابخانه CVXOPT می‌سازد:

* P: ماتریس متقارن مثبت که شامل ماتریس گرمی و بردارهای برچسب‌های کلاس است.
* q: بردار منفی یک‌ها.
* A و b: ماتریس و بردار معادله مساوی که محدودیت‌های برچسب‌های کلاس را تعیین می‌کنند.
* G و h: ماتریس و بردار معادله نامساوی که محدودیت‌های Lagrange multiplier ها را تعیین می‌کنند.
* # 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)))

در نتیجه طبق زیر مسئله برنامه‌ریزی مربعی با استفاده از کتابخانه CVXOPT حل می‌شود. نتیجه شامل Lagrange multiplier ها است.

 # Solve QP problem

    cvxopt.solvers.options['show\_progress'] = False

    solution = cvxopt.solvers.qp(P, q, G, h, A, b)

سپس Lagrange multiplier ها (a) از حل مسئله برنامه‌ریزی مربعی استخراج می‌شوند

بردارهای پشتیبان نمونه‌هایی هستند که Lagrange multiplier های غیرصفر دارند

 # 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]

در ادامه بایاس با استفاده از بردارهای پشتیبان محاسبه می‌شود.

 # Bias

    bias = 0

    for n in range(len(a)):

        bias += sv\_y[n]

        bias -= np.sum(a \* sv\_y \* K[ind[n], sv])

    bias = bias / len(a)

در ادامه در صورتی که هسته خطی باشد، بردار وزن‌ها محاسبه می‌شود. برای دیگر هسته‌ها، وزن‌ها محاسبه نمی‌شوند و مقدار w برابر None خواهد بود.

 # Weight vector for linear kernel

    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

طبق زیر در صورتی که وزن‌ها محاسبه شده باشند (برای هسته خطی)، پیش‌بینی‌ها با استفاده از ضرب داخلی ویژگی‌ها و وزن‌ها به همراه بایاس انجام می‌شود.

در غیر این صورت، پیش‌بینی‌ها با استفاده از بردارهای پشتیبان و توابع هسته مربوطه محاسبه می‌شوند و تابع SVM1 مقادیر وزن‌ها (در صورت استفاده از هسته خطی)، بایاس، نتیجه حل مسئله QP، Lagrange multiplier ها، بردارهای پشتیبان و پیش‌بینی‌ها را برمی‌گرداند.

بطور کلی تابع SVM1 یک پیاده‌سازی جامع از SVM با استفاده از توابع هسته مختلف است که شامل محاسبه ماتریس گرمی، حل مسئله برنامه‌ریزی مربعی، تعیین بردارهای پشتیبان و انجام پیش‌بینی‌ها می‌باشد. این تابع برای تحلیل داده‌های چندکلاسه با استفاده از روش یک-در-برابر-بقیه به کار گرفته می‌شود و می‌تواند با توابع هسته مختلف (خطی، چندجمله‌ای، گاوسی، سیگموئیدی) کار کند.

 # Prediction

    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):

                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)

                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

در ادامه ی کد پیاده‌سازی یک SVM چندکلاسه با استفاده از روش یک-در-برابر-بقیه. هر کلاس به صورت دودویی طبقه‌بندی می‌شود و تصمیم‌گیری نهایی با استفاده از مقایسه امتیازهای تصمیم‌گیری برای هر کلاس انجام می‌شود.

# Multiclass SVM using one-vs-rest approach

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 = {}

    for class\_label in class\_labels:

        binary\_y = np.where(y == class\_label, 1.0, -1.0)

        w, bias, \_, a, sv\_x, sv\_y, prediction = SVM1(

            X, X\_t, binary\_y, C, kernel\_type, poly\_params, RBF\_params, sigmoid\_params)

        classifiers[class\_label] = prediction

    def decision\_function(X\_t):

        decision\_scores = np.zeros((X\_t.shape[0], len(class\_labels)))

        for i, label in enumerate(class\_labels):

            decision\_scores[:, i] = classifiers[label]

        return np.argmax(decision\_scores, axis=1)

    return decision\_function(X\_t)

سپس مدل svm با استفاده از تمام ویژگی ها و هسته RBF آموزش می بیند و دقت ترین و تست محاسبه می شود.

# Train SVM with all features

y\_pred\_train = multiclass\_svm(x\_train, x\_train, y\_train, C=1.0, kernel\_type='RBF', RBF\_params=0.5)

y\_pred\_test = multiclass\_svm(x\_train, x\_test, y\_train, C=1.0, kernel\_type='RBF', RBF\_params=0.5)

# Print training and test accuracy

train\_accuracy = np.mean(y\_pred\_train == y\_train)

test\_accuracy = np.mean(y\_pred\_test == y\_test)

print(f"Training Accuracy: {train\_accuracy \* 100:.2f}%")

print(f"Test Accuracy: {test\_accuracy \* 100:.2f}%")

در ادامه ابتدا از PCA برای کاهش ابعاد استفاده کرده سپس مدل svm را آموزش می دهیم. همانگونه که مشخص است دقت ترین به 98 و دقت تست به حدود 97 درصد رسیده است

# Use PCA for visualization

pca = PCA(n\_components=2)

x\_train\_pca = pca.fit\_transform(x\_train)

x\_test\_pca = pca.transform(x\_test)

# Create a mesh grid for plotting decision boundaries

x\_min, x\_max = x\_train\_pca[:, 0].min() - 1, x\_train\_pca[:, 0].max() + 1

y\_min, y\_max = x\_train\_pca[:, 1].min() - 1, x\_train\_pca[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.02), np.arange(y\_min, y\_max, 0.02))

grid\_points = np.c\_[xx.ravel(), yy.ravel()]

grid\_points\_original\_space = pca.inverse\_transform(grid\_points)

# Predict the decision boundaries

Z = multiclass\_svm(x\_train, grid\_points\_original\_space, y\_train, C=1.0, kernel\_type='RBF', RBF\_params=0.5)

Z = Z.reshape(xx.shape)

# Visualize the results with decision boundaries

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.contourf(xx, yy, Z, alpha=0.8, cmap='viridis')

plt.scatter(x\_train\_pca[:, 0], x\_train\_pca[:, 1], c=y\_train, cmap='viridis', edgecolor='k', s=50)

plt.title('Training Data with Decision Boundaries (PCA)')

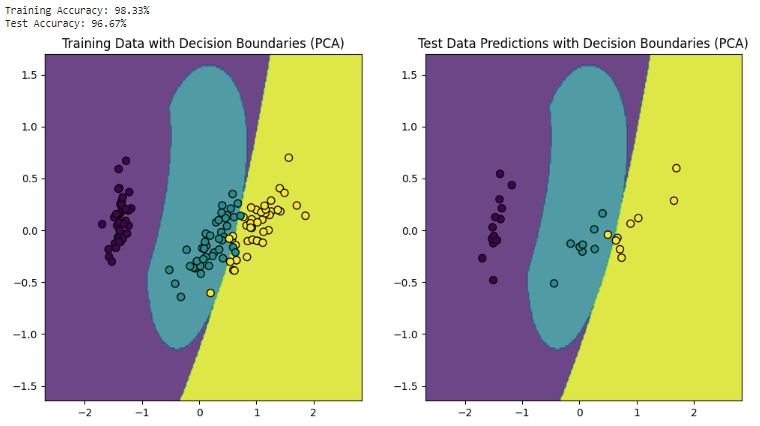
plt.subplot(1, 2, 2)

plt.contourf(xx, yy, Z, alpha=0.8, cmap='viridis')

plt.scatter(x\_test\_pca[:, 0], x\_test\_pca[:, 1], c=y\_pred\_test, cmap='viridis', edgecolor='k', s=50)

plt.title('Test Data Predictions with Decision Boundaries (PCA)')

plt.show()



در ادامه کد کلی است که بهتر از قبل است و توضیحات آن به صورت کلی به صورت زیر است:

**بارگذاری کتابخانه‌ها و تنظیمات اولیه:**

بارگذاری کتابخانه‌های مورد نیاز برای تحلیل داده‌ها، مدل‌سازی، پیش‌پردازش و تصویرسازی.

تعریف دایرکتوری برای ذخیره GIF

# Import necessary libraries

from sklearn.datasets import load\_iris

import numpy as np

import cvxopt

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.decomposition import PCA

import imageio.v2 as imageio

import os

from IPython.display import Image, display

**بارگذاری و پیش‌پردازش داده‌ها:**

بارگذاری داده‌های مجموعه Iris.

استانداردسازی داده‌ها.

اختلاط داده‌ها به صورت تصادفی.

تقسیم داده‌ها به مجموعه‌های آموزشی و آزمایشی با نسبت 80/20.

# Define the directory in Colab to store the GIF

output\_dir = '/content/gif'

if not os.path.exists(output\_dir):

    os.makedirs(output\_dir)

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Standardize the features

X = (X - X.mean(axis=0)) / X.std()

# Shuffle the data

np.random.seed(1234)

indices = np.random.permutation(len(y))

X = X[indices]

y = y[indices]

# Split the data into training and test sets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**تعریف توابع هسته:**

تعریف چندین تابع هسته برای SVM، شامل خطی، چندجمله‌ای، گاوسی (RBF)، و سیگموئیدی.

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):

    return np.tanh(alpha \* np.dot(x, y) + C)

**پیاده‌سازی SVM با توابع هسته مختلف:**

این تابع SVM1 برای پیاده‌سازی SVM با توابع هسته مختلف استفاده می‌شود. شامل محاسبه ماتریس گرمی، حل مسئله برنامه‌ریزی مربعی QP با استفاده از CVXOPT، تعیین بردارهای پشتیبان، بایاس و وزن‌ها (برای هسته خطی) و انجام پیش‌بینی‌ها است.

def SVM1(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)

    # Weight vector

    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

    # Create the decision boundary for the plots. Calculates the hypothesis

    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)

                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

**SVM چندکلاسه با استفاده از روش یک-در-برابر-بقیه:**

این تابع multiclass\_svm پیاده‌سازی SVM چندکلاسه با استفاده از روش یک-در-برابر-بقیه است. هر کلاس به صورت دودویی طبقه‌بندی می‌شود و تصمیم‌گیری نهایی با استفاده از مقایسه امتیازهای تصمیم‌گیری برای هر کلاس انجام می‌شود.

def multiclass\_svm(X, X\_t, y, C, kernel\_type, poly\_params=(1, 4), RBF\_params=0.5, sigmoid\_params=(1, 0.01)):

    # Step 1: Identify unique class labels

    class\_labels = list(set(y))

    # Step 2: Initialize classifiers dictionary

    classifiers = {}

    w\_catch = {}  # Catching w, b only for plot part

    b\_catch = {}

    a\_catch = {}

    sv\_x\_catch = {}

    sv\_y\_catch = {}

    # Step 3: Train binary SVM models for each required class combination

    for class\_label in class\_labels:

        # Create binary labels for current class vs. all others

        binary\_y = np.where(y == class\_label, 1.0, -1.0)

        # Train SVM classifier for binary classification

        w, bias, \_, a, sv\_x, sv\_y, prediction, kernel\_and\_params = SVM1(

            X, X\_t, binary\_y, C, kernel\_type, poly\_params, RBF\_params, sigmoid\_params)

        classifiers[class\_label] = prediction

        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):

            decision\_scores[:, i] = classifiers[label]

        return np.argmax(decision\_scores, axis=1), kernel\_and\_params, w\_catch, b\_catch, classifiers

    return decision\_function(X\_t)

**استفاده از PCA برای تصویرسازی:**

استفاده از PCA برای کاهش ابعاد داده‌ها به دو بعد برای تصویرسازی.

ایجاد شبکه مش برای پیش‌بینی مرزهای تصمیم.

آموزش و پیش‌بینی مدل SVM چندکلاسه با درجه‌های مختلف هسته چندجمله‌ای.

# Use PCA for visualization

pca = PCA(n\_components=2)

x\_train\_pca = pca.fit\_transform(x\_train)

x\_test\_pca = pca.transform(x\_test)

**ایجاد و نمایش GIF:**

تصویرسازی مرزهای تصمیم برای هر درجه و ذخیره تصاویر.

ارزیابی دقت مدل روی داده‌های آزمایشی و ذخیره دقت‌ها.

ایجاد GIF از تصاویر مرزهای تصمیم برای درجه‌های مختلف هسته چندجمله‌ای.

نمایش دقت مدل برای درجه‌های مختلف و نمایش نمودار دقت‌ها.

نمایش GIF ایجاد شده در محیط نوت‌بوک.

# Create a mesh grid for plotting decision boundaries

x\_min, x\_max = x\_train\_pca[:, 0].min() - 1, x\_train\_pca[:, 0].max() + 1

y\_min, y\_max = x\_train\_pca[:, 1].min() - 1, x\_train\_pca[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.02),

                     np.arange(y\_min, y\_max, 0.02))

# Store the images for the GIF

images = []

accuracies = []

for degree in range(1, 11):

    grid\_points = np.c\_[xx.ravel(), yy.ravel()]

    grid\_points\_original\_space = pca.inverse\_transform(grid\_points)

    # Predict the decision boundaries

    \_, \_, \_, \_, classifiers = multiclass\_svm(

        x\_train, grid\_points\_original\_space, y\_train, C=1.0, kernel\_type='polynomial', poly\_params=(1, degree))

    # Reshape the predictions to match the grid shape

    Z = np.argmax(np.vstack([classifiers[class\_label] for class\_label in sorted(set(y\_train))]), axis=0)

    Z = Z.reshape(xx.shape)

    # Visualize the results with decision boundaries

    fig, ax = plt.subplots(figsize=(12, 6))

    ax.contourf(xx, yy, Z, alpha=0.8, cmap='viridis')

    ax.scatter(x\_train\_pca[:, 0], x\_train\_pca[:, 1], c=y\_train, cmap='viridis', edgecolor='k', s=50)

    ax.set\_title(f'Training Data with Decision Boundaries (PCA) - Degree {degree}')

    # Save the figure

    plt.savefig(f'Q1D\_degree\_{degree}.png')

    images.append(imageio.imread(f'Q1D\_degree\_{degree}.png'))

    plt.close(fig)

    # Evaluate accuracy

    y\_pred\_test, \_, \_, \_, \_ = multiclass\_svm(

        x\_train, x\_test, y\_train, C=1.0, kernel\_type='polynomial', poly\_params=(1, degree))

    test\_accuracy = np.mean(y\_pred\_test == y\_test)

    accuracies.append(test\_accuracy)

# Create GIF

gif\_path = os.path.join(output\_dir, 'Q1D\_polynomial\_kernel\_degrees.gif')

imageio.mimsave(gif\_path, images, loop=5, fps=2)

# Plot accuracy for different degrees

plt.figure()

plt.plot(range(1, 11), accuracies, marker='o')

plt.xlabel('Polynomial Degree')

plt.ylabel('Test Accuracy')

plt.title('Test Accuracy vs Polynomial Degree')

plt.grid(False)

plt.show()

print("GIF saved as 'Q1D\_polynomial\_kernel\_degrees.gif'")

print("Accuracy for degrees 1 to 10:")

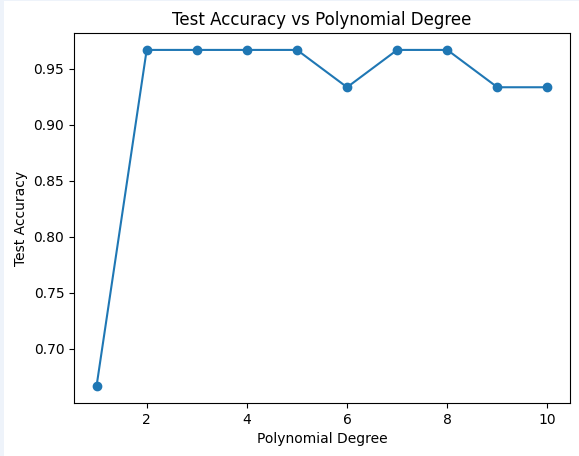
for degree, accuracy in enumerate(accuracies, start=1):

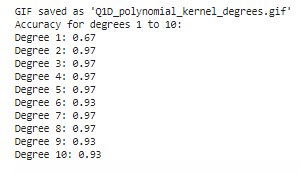
    print(f"Degree {degree}: {accuracy:.2f}")

# Display GIF

display(Image(filename=gif\_path))

نتایج به صورت زیر است:





گیف با اسم q1-d در فایل موجود است.



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

در ادامه دقیق تر به بررسی چالش ها و راه حل ها می پردازیم.

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

1. پروفایل رفتارهای تقلبی پویا:

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

2. عدم توازن داده‌ها:

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

3. انتخاب ویژگی‌های بهینه:

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

4. معیارهای ارزیابی مناسب:

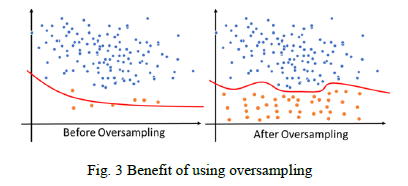
انتخاب معیارهای مناسب برای ارزیابی عملکرد مدل‌ها در داده‌های نامتوازن یکی دیگر از چالش‌هاست. معیارهای سنتی دقت (Accuracy) نمی‌توانند به خوبی عملکرد مدل را در چنین شرایطی نشان دهند .

پس در ارزیابی مدل‌های طبقه‌بندی، استفاده از معیار دقت (Accuracy) به تنهایی کافی نیست، به ویژه زمانی که با مجموعه داده‌های نامتوازن مواجه هستیم. به عنوان مثال، فرض کنید یک مجموعه داده داریم که 99.9٪ آن شامل داده‌های عادی و تنها 0.1٪ آن شامل داده‌های غیرعادی است. اگر یک مدل طبقه‌بندی همه نمونه‌ها را به عنوان داده‌های عادی برچسب‌گذاری کند، دقت آن مدل 99.9٪ خواهد بود. این عدد ممکن است در نگاه اول بسیار خوب به نظر برسد، اما در واقعیت مدل نتوانسته هیچ‌یک از داده‌های غیرعادی را شناسایی کند و در شناسایی ناهنجاری‌ها کاملاً ناکارآمد است.

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

1. استفاده از الگوریتم Oversampling :

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



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

فرایند SMOTE به شرح زیر است:

الف. شناسایی k-نزدیک‌ترین همسایه‌ها:

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

ب. انتخاب تصادفی یک همسایه:

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

ج. ایجاد نقطه داده جدید:

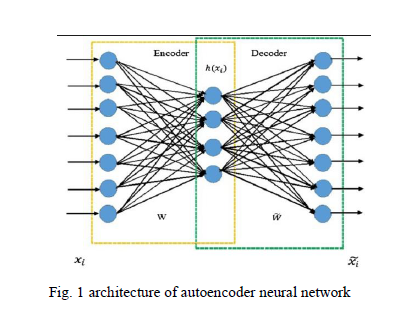
با استفاده از نمونه اصلی و نقطه همسایه انتخاب شده، یک نمونه جدید ایجاد می‌شود. این کار با استفاده از میانگین وزنی انجام می‌شود، به این صورت که مقداری از فاصله بین دو نقطه را بر اساس یک ضریب تصادفی محاسبه می‌کنند و این مقدار را به نقطه اصلی اضافه می‌کنند. به این ترتیب، نقطه داده جدید در میان دو نقطه اصلی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

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

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

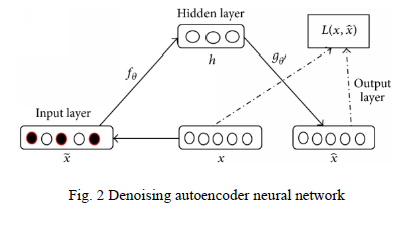
2. استفاده از شبکه‌های عصبی Autoencoder :

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



3. استفاده از Denoising Autoencoder :

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



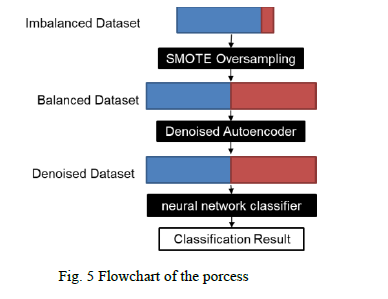
4. استفاده از معیارهای ارزیابی مختلف:

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

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

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

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





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

1. شبکه عصبی خود رمزگذار نویزگیری شده Denoising Autoencoder)) :

این شبکه شامل 7 لایه است که برای فرآیند نویزگیری طراحی شده است. بعد از اعمال نویز گوسی به داده‌های آموزشی، داده‌های نویزدار به این شبکه وارد می‌شوند و مدل خود رمزگذار آموزش می‌بیند تا قابلیت نویزگیری داده‌ها را در فرآیند پیش‌بینی به دست آورد.

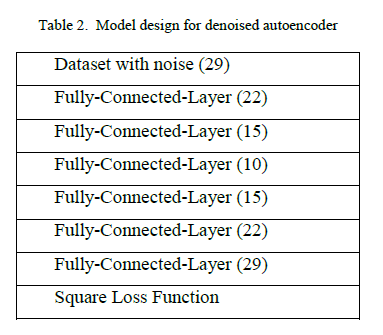
لایه‌ها شامل:

لایه ورودی با داده‌های نویزدار

چندین لایه کاملاً متصل با تعداد نورون‌های متغیر

استفاده از تابع زیان مربع برای بهینه‌سازی





2. طبقه‌بند:

این بخش شامل یک شبکه عصبی کاملاً متصل عمیق (Deep Fully Connected Neural Network) با 6 لایه است. پس از نویزگیری داده‌های آموزشی، داده‌ها به این طبقه‌بند وارد می‌شوند.

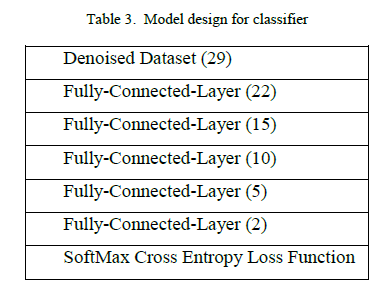
لایه‌ها شامل:

لایه ورودی با داده‌های نویزگیری شده

چندین لایه کاملاً متصل با تعداد نورون‌های متغیر

استفاده از تابع زیان انتروپی متقاطع SoftMax برای طبقه‌بندی نهایی

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





در ابتدا، از مجموعه داده‌ای که شامل 28315 تراکنش کارت اعتباری است و 0.5٪ از آن‌ها به عنوان کلاهبرداری دسته‌بندی شده‌اند، استفاده کرده است. سپس با استفاده از بیش‌نمونه‌گیری، مجموعه داده را به یک مجموعه داده متعادل تبدیل کرده است. بعد از آن، از اتوانکودر دنویز شده برای دریافت مجموعه داده دنویز شده استفاده شده است. در نهایت، از یک مدل شبکه عصبی کاملاً متصل و عمیق برای طبقه‌بندی نهایی استفاده می کند. در فرآیند پیش‌پردازش داده، داده "زمان" حذف و بخش "مقدار" نرمال‌سازی می‌شود. برای بیش‌نمونه‌گیری، تنها بر روی مجموعه داده آموزش این عمل انجام می‌شود و سپس مجموعه داده آموزش به یک مجموعه داده شامل تعداد مساوی نمونه‌های عادی و ناهنجار تبدیل می‌شود. سپس با استفاده از یک اتوانکودر دنویز کننده، مجموعه داده را دنویز می‌کنند تا بتوانند از آن در فرآیند طبقه‌بندی استفاده کنند. در پایان، از یک مدل طبقه‌بندی با استفاده از شبکه عصبی کاملاً متصل و عمیق برای طبقه‌بندی نهایی استفاده می‌کنند.

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

در ابتدا از kaggle دیتاست را دانلود می کنیم.

این مجموعه داده شامل تراکنش‌هایی است که توسط کارت‌های اعتباری در ماه سپتامبر 2013 توسط دارندگان کارت اروپایی انجام شده است. این مجموعه داده شامل تراکنش‌هایی است که در دو روز اتفاق افتاده است، ما 492 تقلب را از بین 284807 تراکنش داریم. این مجموعه داده بسیار نامتوازن است، کلاس مثبت (تقلب‌ها) حدود 0.172٪ از کل تراکنش‌ها را تشکیل می‌دهد. این شامل فقط متغیرهای ورودی عددی است که نتیجه تبدیل PCA است. ویژگی‌های V1، V2، .... V28کامپوننت‌های اصلی است که با PCA به دست آمده‌اند، تنها ویژگی‌هایی که با PCA تبدیل نشده‌اند، "زمان" و "مقدار" هستند. ویژگی 'زمان' حاوی ثانیه‌های گذشته شده بین هر تراکنش و اولین تراکنش در مجموعه داده است. ویژگی 'مقدار' مبلغ تراکنش است، این ویژگی می‌تواند برای یادگیری وابسته به مثال با حساسیت به هزینه استفاده شود. ویژگی 'کلاس' متغیر پاسخ است و مقدار 1 را در صورت وقوع تقلب و در غیر این صورت صفر می‌گیرد. ( 284807 ردیف و 31 ستون)

کتابخانه های مورد نظر را ایمپورت کرده سپس دیتا را با دستورgdown دانلود کرده و سپس فایل csv آن را میخوانیم. همچنین اطلاعات دیتاست را با دستور data.info() مشاهدا می کنیم.

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from imblearn.over\_sampling import SMOTE

import tensorflow as tf

from keras.models import Model, Sequential

from keras.layers import Input, Dense

from keras.optimizers import Adam

from sklearn.metrics import recall\_score,confusion\_matrix,classification\_report,precision\_score, f1\_score,precision\_recall\_curve,accuracy\_score, precision\_score

from keras.metrics import Recall, Accuracy, F1Score , confusion\_metrics , accuracy\_metrics

from imblearn.over\_sampling import SMOTE

import matplotlib.pyplot as plt

from keras.callbacks import ModelCheckpoint, EarlyStopping

from keras.utils import to\_categorical

from keras.layers import InputLayer, Dense, Dropout

import  keras.regularizers

from keras.optimizers import Adam

!pip install --upgrade --no-cache-dir gdown

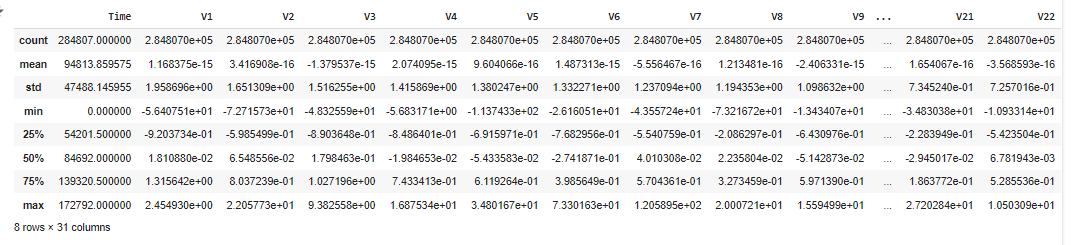
!gdown 1Ut0M6XtbZDDr--0TPDtdAf65fN9LmiKJ

data = pd.read\_csv('creditcard.csv')

data.info()

سپس یک خلاصه آماری از داده ها را توسط data.describe() میبینیم:

ata.describe()



در ادامه کد بالا چندین عملیات بررسی و تحلیل اولیه روی یک DataFrame انجام می‌دهد:

:dir()لیست همه متدها و صفات موجود در فضای نام جاری را نمایش می‌دهد.

cor=data.corr(): ماتریس همبستگی بین تمام ستون‌های عددی در DataFrame `data` را محاسبه می‌کند.

: print(cor) ماتریس همبستگی محاسبه‌شده را چاپ می‌کند.

: data.hist()هیستوگرام‌های ستون‌های عددی را ترسیم می‌کند تا توزیع داده‌ها را نشان دهد. data.describe()خلاصه آماری از DataFrame ارائه می‌دهد که شامل تعداد، میانگین، انحراف معیار، حداقل، چارک‌ها و حداکثر مقدار برای هر ستون عددی است.

data.head(): پنج ردیف اول DataFrame را نمایش می‌دهد.

data.info(): اطلاعات کلی درباره DataFrame را نمایش می‌دهد که شامل تعداد ردیف‌ها، ستون‌ها، نوع داده هر ستون و میزان حافظه مصرفی است.

: print(data.columns) نام تمام ستون‌های DataFrame را چاپ می‌کند.

: print(data.shape) شکل DataFrame را به صورت تعداد ردیف‌ها و تعداد ستون‌ها چاپ می‌کند.

dir()

# Calculate and print the correlation matrix

cor = data.corr()

print("Correlation Matrix:")

print(cor)

# Plot histograms for numeric columns

data.hist(figsize=(10, 8))

plt.tight\_layout()

plt.show()

# Display descriptive statistics

print("Descriptive Statistics:")

print(data.describe())

# Display the first five rows

print("First Five Rows:")

print(data.head())

# Display general information about the DataFrame

print("DataFrame Info:")

print(data.info())

# Print the column names

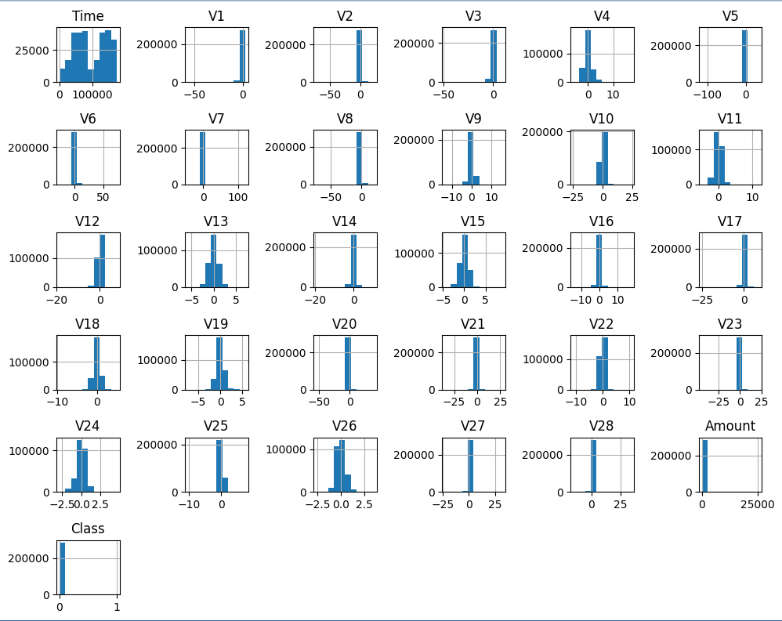
print("Column Names:")

print(data.columns)

# Print the shape of the DataFrame

print("DataFrame Shape:")

print(data.shape)



در ادامه ستون time را دراپ کرده و دیتا را نرمالایز می کنیم:

# Load dataset

data = pd.read\_csv('creditcard.csv')

print(f'raw dataset shape: {data.shape}')

# Drop the 'Time' column

data = data.drop(['Time'], axis=1)

# Normalize the 'Amount' column

data['Amount'] = StandardScaler().fit\_transform(data['Amount'].values.reshape(-1, 1))

# Split into features and target

X = data.drop('Class', axis=1)

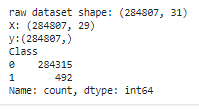
y = data['Class']

print(f'X: {X.shape}\ny:{y.shape}')

# Get unique value counts for a specific column

unique\_counts = data['Class'].value\_counts()

print(unique\_counts)



سپس با SMOTE دیتا را بالانس کرده و ورودی و تارگت را جدا کرده سپس اسپلیت می کنیم. در این بخش وان هات هم انجام می دهیم:

# Oversample the training data using SMOTE

sm = SMOTE(random\_state=24,sampling\_strategy='minority')

X\_res, y\_res = sm.fit\_resample(X, y)

# Split into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_res, y\_res, test\_size=0.2, random\_state=24)

X\_train.shape , X\_test.shape

# One-hot encode the labels

y\_train\_onehot = to\_categorical(y\_train, num\_classes=2)

y\_test\_onehot = to\_categorical(y\_test, num\_classes=2)

X\_res.shape, y\_res.shape,X\_train.shape , X\_test.shape



در ادامه به دیتا نویز گوسی با میانگین صفر وانحراف معیار 0.5 اضافه می کنیم و داده ی نویزی را بین صفر و یک اسکیل می کنیم.

سپس یک دینویزینگ اتوانکدر تعریف می کنیم و آن را روی دیتای ترین آموزش می دهیم. سپس مدلی با ورودی لایه اول خودرمزگذار و خروجی لایه رمزگذار ساخته می‌شود.

داده‌های آموزشی و آزمایشی با استفاده از مدل رمزگذار رمزگذاری می‌شوند تا ویژگی‌های فشرده‌شده استخراج شوند.

ایپاک و بقیه تنظیمات برای آموزش در کد زیر مشخص هستند.

input\_dim = X\_train.shape[1]

# Define the denoising autoencoder architecture with regularization and dropout

autoencoder = Sequential([

    Input(shape=(input\_dim,),name='input'),

    Dense(22, activation='tanh'),

    Dense(15, activation='tanh'),

    Dense(10, activation='leaky\_relu', name="encode"),

    Dense(15, activation='tanh', ),

    Dense(22, activation='leaky\_relu',),

    Dense(input\_dim, activation='linear')

])

autoencoder.compile(optimizer='Nadam', loss='mean\_squared\_error')

#optimizer=Adam(learning\_rate=0.001)

# Add Gaussian noise to the training data

noise\_factor = 0.1

X\_train\_noisy = X\_train + noise\_factor \* np.random.normal(loc=0.0, scale=0.5, size=X\_train.shape)

X\_train\_noisy = np.clip(X\_train\_noisy, 0., 1.)

# Train the autoencoder

dae\_history = autoencoder.fit(X\_train\_noisy, X\_train, epochs=30, batch\_size=200, shuffle=True, validation\_split=0.2,

                             callbacks=[EarlyStopping(monitor='val\_loss', patience=15, restore\_best\_weights=True)])

# Extract the encoder part for feature extraction

encoder\_model = Model(inputs=autoencoder.layers[0].input, outputs=autoencoder.get\_layer("encode").output)

X\_train\_encoded = encoder\_model.predict(X\_train)

X\_test\_encoded = encoder\_model.predict(X\_test)

سپس خطای ترین و ولیدیشن را پلات می کنیم:

# Extract loss values from the history object

loss = dae\_history.history['loss']

val\_loss = dae\_history.history['val\_loss']

# Create a new figure for the plot

plt.figure(figsize=(10, 6))

# Plot the training and validation loss for autoencoder

plt.plot(loss, label='Training Loss', color='blue', linestyle='-', linewidth=2)

plt.plot(val\_loss, label='Validation Loss', color='orange', linestyle='--', linewidth=2)

# Add title and labels

plt.title('Training and Validation Loss for Autoencoder')

plt.xlabel('Epochs')

plt.ylabel('Loss')

# Add grid

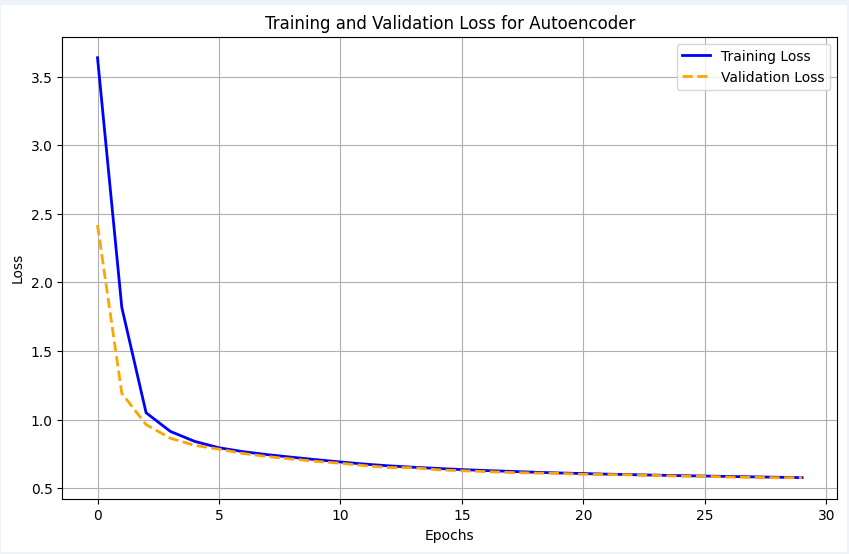
plt.grid(True)

# Add legend

plt.legend()

# Show the plot

plt.show()



در ادامه این کد یک مدل طبقه‌بندی با استفاده از Keras تعریف و آموزش می‌دهد:

**تعریف مدل طبقه‌بندی:**

یک مدل ترتیبی (Sequential) با لایه‌های Dense مختلف و توابع فعال‌سازی tanh، relu و softmax ساخته می‌شود.

لایه ورودی مدل دارای 10 ویژگی است و لایه خروجی دو نورون با تابع فعال‌سازی softmax برای طبقه‌بندی دوکلاسه دارد.

**کامپایل مدل:**

مدل با استفاده از بهینه‌ساز Adam و نرخ یادگیری 0.0001 کامپایل می‌شود.

از تابع هزینه categorical\_crossentropy و متریک‌های دقت و یادآوری برای ارزیابی مدل استفاده می‌شود.

**تعریف کال‌بک‌ها:**

ModelCheckpoint مدل را زمانی که بهبود در مقدار val\_loss مشاهده می‌شود، ذخیره می‌کند.

EarlyStopping آموزش را زمانی متوقف می‌کند که بهبود در val\_loss به مدت 10 دوره مشاهده نشود و بهترین وزن‌ها را بازیابی می‌کند.

**آموزش مدل:**

مدل با استفاده از داده‌های آموزشی رمزگذاری شده X\_train\_encoded و برچسب‌های یک‌داغ y\_train\_onehot به مدت 50 دوره آموزش داده می‌شود.

اندازه بسته‌ها 200 و 20٪ از داده‌ها برای اعتبارسنجی استفاده می‌شود.

داده‌ها در هر دوره به صورت تصادفی بازآرایی می‌شوند و از کال‌بک‌های تعریف شده برای ذخیره و متوقف کردن زودهنگام استفاده می‌شود.

# Define the classifier architecture using Sequential

classifier = Sequential([

    Input(shape=(10,)),

    Dense(22, activation='tanh'),

    Dense(15, activation='tanh'),

    Dense(10, activation='relu'),

    Dense(5, activation='relu'),

    Dense(2, activation='softmax')

])

classifier.compile(optimizer=Adam(learning\_rate=0.0001), loss='categorical\_crossentropy', metrics=['accuracy',Recall()])

# Define the ModelCheckpoint and EarlyStopping callbacks

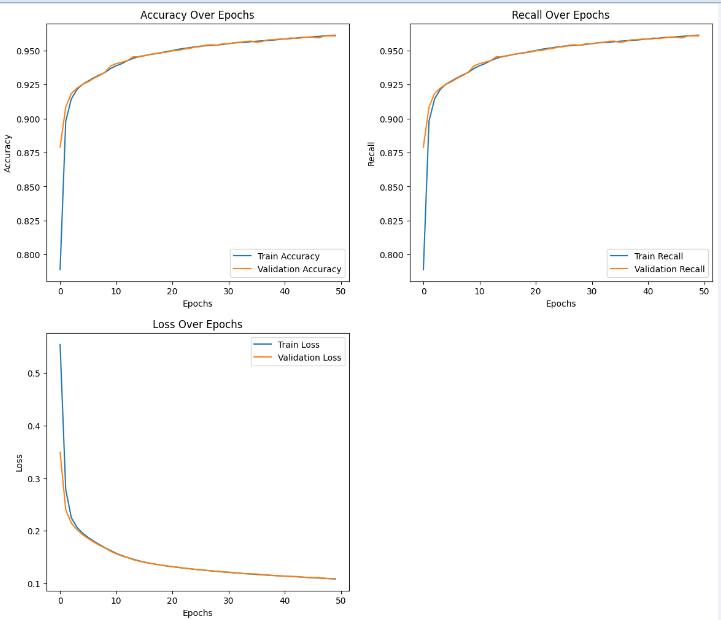
checkpoint = ModelCheckpoint('best\_model.keras', monitor='val\_loss', save\_best\_only=True, mode='min')

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, mode='min', verbose=1, restore\_best\_weights=True,min\_delta=0.005)

# Train the classifier

history = classifier.fit(X\_train\_encoded, y\_train\_onehot, epochs=50, batch\_size=200, shuffle=True, validation\_split=0.2, callbacks=[checkpoint, early\_stopping])

سپس ریکال و دقت و لاس را برای تزین و ولیدیشن پلات می کنیم:



سپس این کد مدل طبقه‌بندی آموزش‌دیده را ارزیابی می‌کند و دقت و یادآوریrecall را در آستانه‌های مختلف محاسبه و ترسیم می‌کند. ابتدا پیش‌بینی‌های مدل طبقه‌بندی برای داده‌های آزمایشی محاسبه می‌شود که خروجی شامل احتمالات کلاس‌ها برای هر نمونه است. سپس مجموعه‌ای از آستانه‌ها از 0.0 تا 1.0 با گام‌های 0.1 تعریف می‌شود و دو لیست خالی برای ذخیره دقت‌ها و یادآوری‌ها ایجاد می‌شود. برای هر آستانه، کلاس‌های پیش‌بینی‌شده با مقایسه احتمالات با آستانه محاسبه می‌شوند و دقت و یادآوری برای هر آستانه محاسبه و به لیست‌های مربوطه اضافه می‌شود. در نهایت، یک نمودار با اندازه 10x6 ایجاد می‌شود و دقت و یادآوری در مقابل آستانه‌ها ترسیم می‌شود. محورهای x و y برچسب‌گذاری می‌شوند و عنوان نمودار و لگند اضافه می‌شود. شبکه‌ای برای خوانایی بهتر اضافه می‌شود و نمودار نمایش داده می‌شود. این تحلیل به شما کمک می‌کند تا آستانه بهینه‌ای برای مدل خود انتخاب کنید که تعادل مناسبی بین دقت و یادآوری ایجاد کند.

# Evaluate the classifier

y\_pred\_probs = classifier.predict(X\_test\_encoded)

# Calculate metrics at various thresholds

thresholds = np.arange(0.0, 1.1, 0.1)

accuracies = []

recalls = []

for threshold in thresholds:

    y\_pred\_classes = (y\_pred\_probs[:, 1] >= threshold).astype(int)

    accuracies.append(accuracy\_score(y\_test, y\_pred\_classes))

    recalls.append(recall\_score(y\_test, y\_pred\_classes))

# Plot accuracy and recall vs threshold

plt.figure(figsize=(10, 6))

plt.plot(thresholds, accuracies, label='Accuracy')

plt.plot(thresholds, recalls, label='Recall')

plt.xlabel('Threshold')

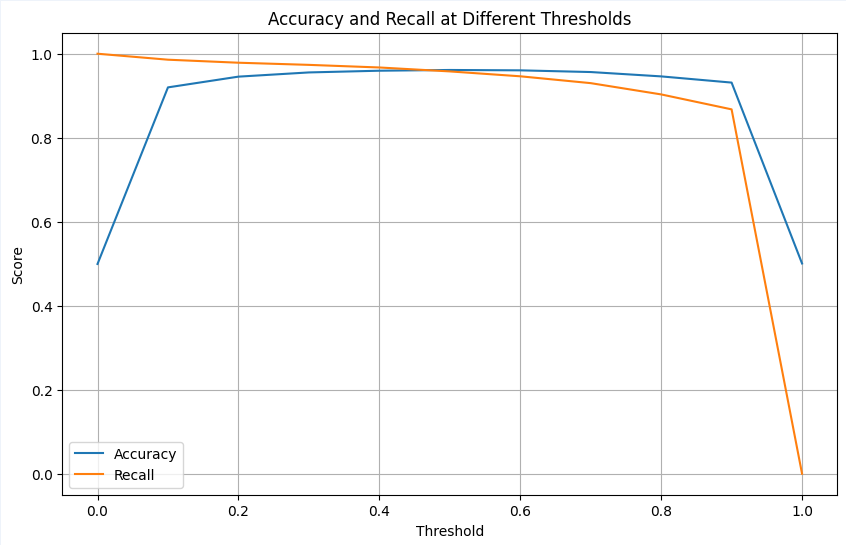
plt.ylabel('Score')

plt.title('Accuracy and Recall at Different Thresholds')

plt.legend()

plt.grid(True)

plt.show()



در ادامه توسط کد زیر دقت و ... برای قسمت آزمایش چاپ می شوند.

# Evaluate the classifier

y\_pred = classifier.predict(X\_test\_encoded)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

# Calculate evaluation metrics

accuracy = accuracy\_score(y\_test, y\_pred\_classes)

precision = precision\_score(y\_test, y\_pred\_classes)

recall = recall\_score(y\_test, y\_pred\_classes)

f1 = f1\_score(y\_test, y\_pred\_classes)

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_classes)

class\_report = classification\_report(y\_test, y\_pred\_classes)

print("Accuracy:", accuracy)

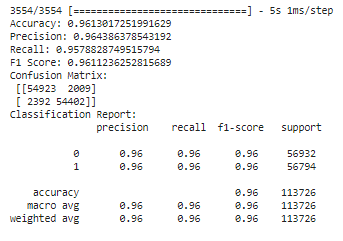
print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

print("Confusion Matrix:\n", conf\_matrix)

print("Classification Report:\n", class\_report)



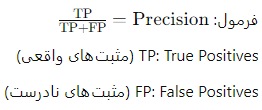
در مسائلی که توزیع برچسب‌ها نامتوازن است، استفاده از معیار Accuracy به تنهایی معمولاً عملکرد مدل را به درستی نمایش نمی‌دهد. دلیل این امر این است که در مسائل نامتوازن، مدل می‌تواند با نادیده گرفتن کلاس‌های کم‌تر شایع، دقت بالایی کسب کند، بدون اینکه واقعاً عملکرد خوبی داشته باشد. به عنوان مثال، اگر 95 درصد داده‌ها به کلاس 0 و 5 درصد به کلاس 1 تعلق داشته باشند، مدلی که همیشه کلاس 0 را پیش‌بینی می‌کند، 95 درصد دقت خواهد داشت، ولی عملاً هیچ نمونه‌ای از کلاس 1 را به درستی تشخیص نداده است.

**معیارهای مکمل**

برای ارزیابی بهتر مدل در مسائل نامتوازن، باید از معیارهای مکمل استفاده کرد که اطلاعات بیشتری در مورد عملکرد مدل در هر دو کلاس ارائه دهند. این معیارها شامل:

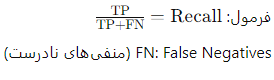
**دقت:**

دقت نشان می‌دهد که چه نسبتی از نمونه‌های پیش‌بینی‌شده به عنوان مثبت واقعاً مثبت هستند.



**یادآوری:**

یادآوری نشان می‌دهد که چه نسبتی از نمونه‌های واقعی مثبت به درستی پیش‌بینی شده‌اند.



**F1 Score:**

امتیاز F1 میانگینی از دقت و یادآوری است و به تعادل بین این دو معیار کمک می‌کند.



**ماتریس درهم‌ریختگی:**

ماتریس درهم‌ریختگی تعداد نمونه‌های واقعی و پیش‌بینی‌شده را در هر کلاس نشان می‌دهد و تصویر کاملی از عملکرد مدل ارائه می‌دهد.

شامل: TP, TN, FP, FN

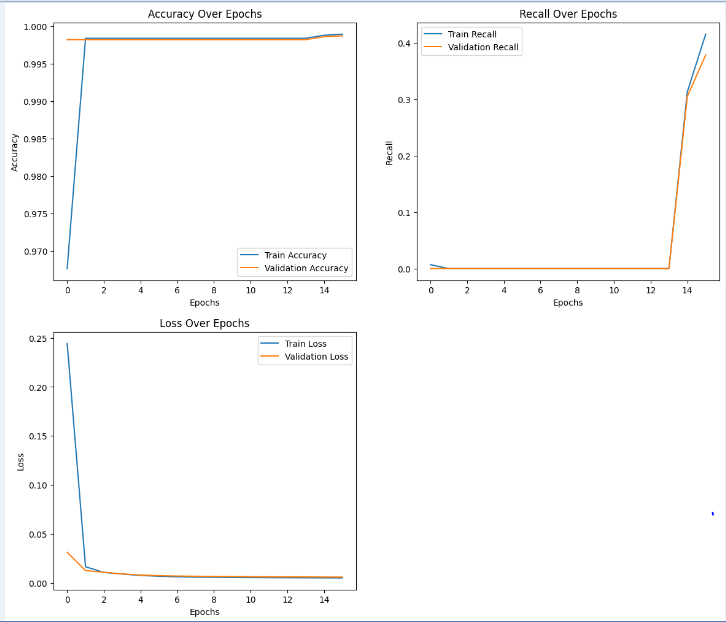
**ROC Curve و AUC (Area Under the Curve):**

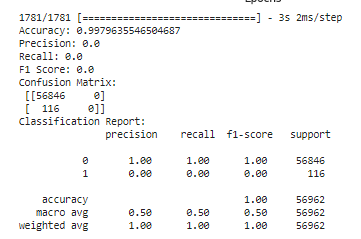
نمودار ROC (Receiver Operating Characteristic) نشان می‌دهد که مدل چگونه می‌تواند بین کلاس‌های مختلف تمایز قائل شود.

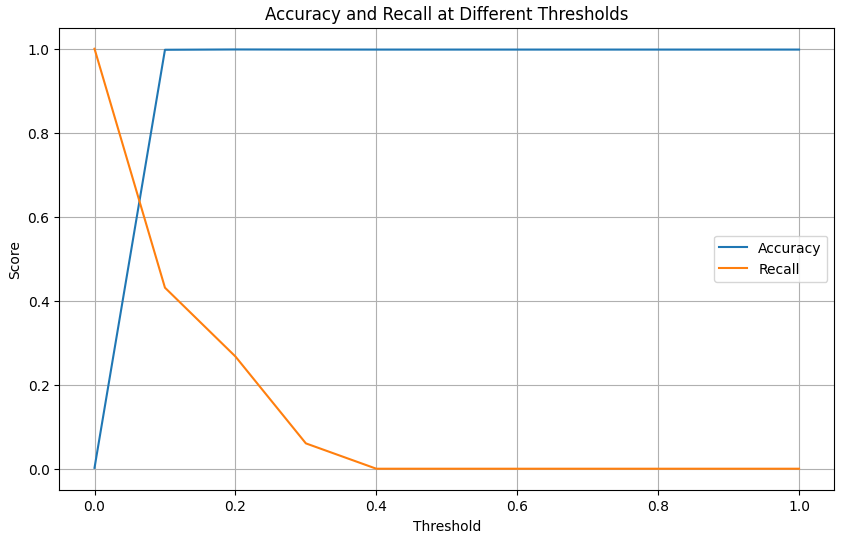
AUC یک مقدار عددی بین 0 و 1 است که عملکرد کلی مدل را در تمایز بین کلاس‌ها نشان می‌دهد. AUC بالاتر به معنای عملکرد بهتر است.

به طور کلی در مسائل نامتوازن، معیار Accuracy به تنهایی کافی نیست و بهتر است از معیارهای Precision، Recall، F1 Score، Confusion Matrix و ROC Curve/AUC استفاده شود تا تصویر کامل‌تری از عملکرد مدل به دست آید. این معیارها به تشخیص بهتر عملکرد مدل در تمایز بین کلاس‌های مختلف کمک می‌کنند و از توجه به تنها یک کلاس جلوگیری می‌کنند.

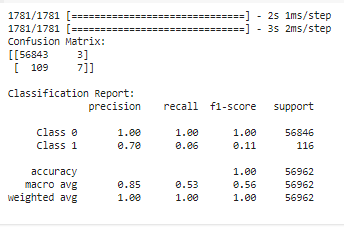
در ادامه بدون over sampling کد نوشته شده وجود دارد که به صورت زیر است:







در نهایت دقت و ... و کانفیوژن ماتریکس برای قسمت تست چاپ شده است:



دیتای آنبالانیس دقت بیشتری به ما داده است ولی همه را در کلاس با تعداد بیشتر تشخیص داده است..