به نام خدا

دانشکده مهندسی برق

گزارش درس یادگیری ماشین

مقطع: کارشناسی ارشد گرایش: مهندسی کنترل

گزارش مینی پروژه سوم

توسط:

مرجان محمدی

۴۰۱۱۱۵۳۴

استاد درس:

دکتر علیاری

لینک کولب

لینک گیت هاب

بهار ۱۴۰۳

فهرست مطالب

۱ پرسش یک

هدف از این سوال آزمایش الگوریتم SVM در نمونه‌های مختلف روی دیتاست معروف گل‌زنبق[1] است. مراحل زیر را یک به یک انجام دهید و موارد خواسته‌شده در گزارش خود به همراه کدها ارسال کنید.

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.

دیتاست Iris یکی از مشهورترین و قدیمی‌ترین دیتاست‌ها در حوزه یادگیری ماشین است که توسط رونالد فیشر در سال ۱۹۳۶ معرفی شد. این دیتاست شامل ۱۵۰ نمونه از گل‌های زنبق (Irisبا 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.

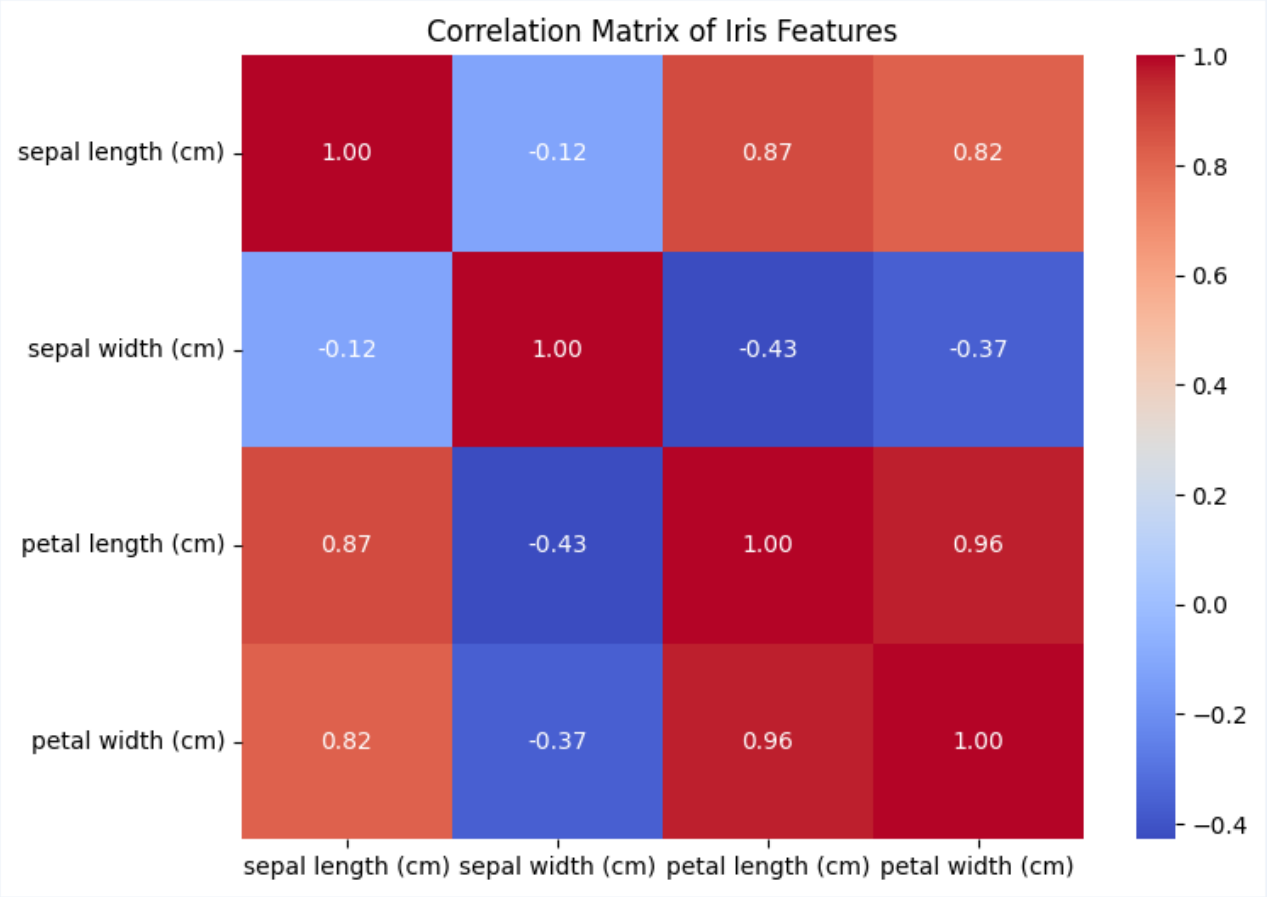
در نهایت، این بخش `print_iris_attributes(iris)` را فراخوانی می‌کند که تمام ویژگی‌ها و مقادیر مربوط به آنها را از شیء دیتاست `iris` چاپ می‌کند.

```
# 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()
```



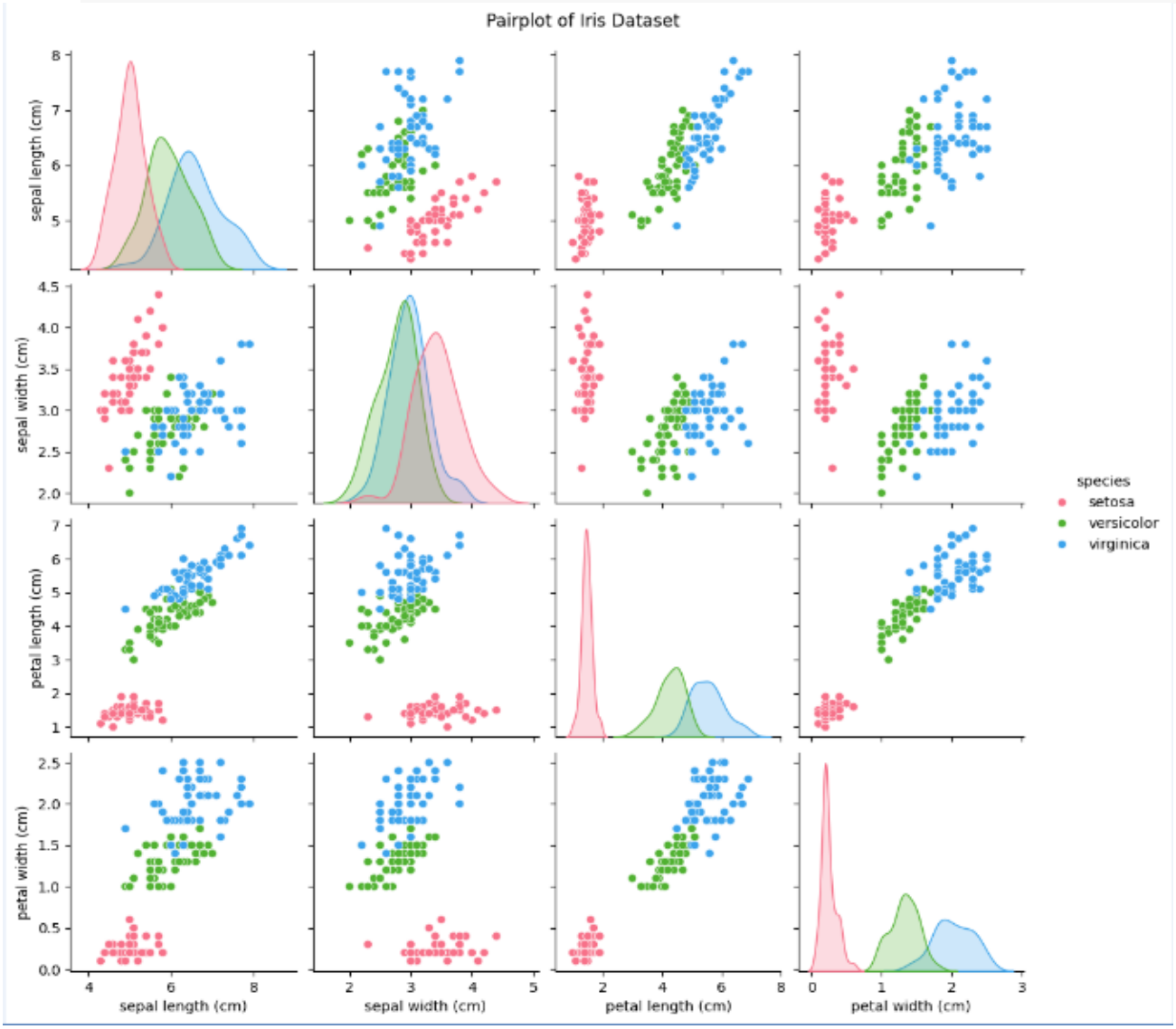
در ادامه یک ستون جدید به DataFrame `df` اضافه می‌شود که نام آن `species`است. برای ایجاد این ستون، از تابع `pd.Categorical.from_codes()`استفاده می‌شود. این تابع با استفاده از آرایه `y` و نام‌های کلاس‌های هدف(`target_names`)، یک ستون دسته‌ای از کلاس‌های هدف (نام گونه‌های گل‌ها) ایجاد می‌کند و به DataFrame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

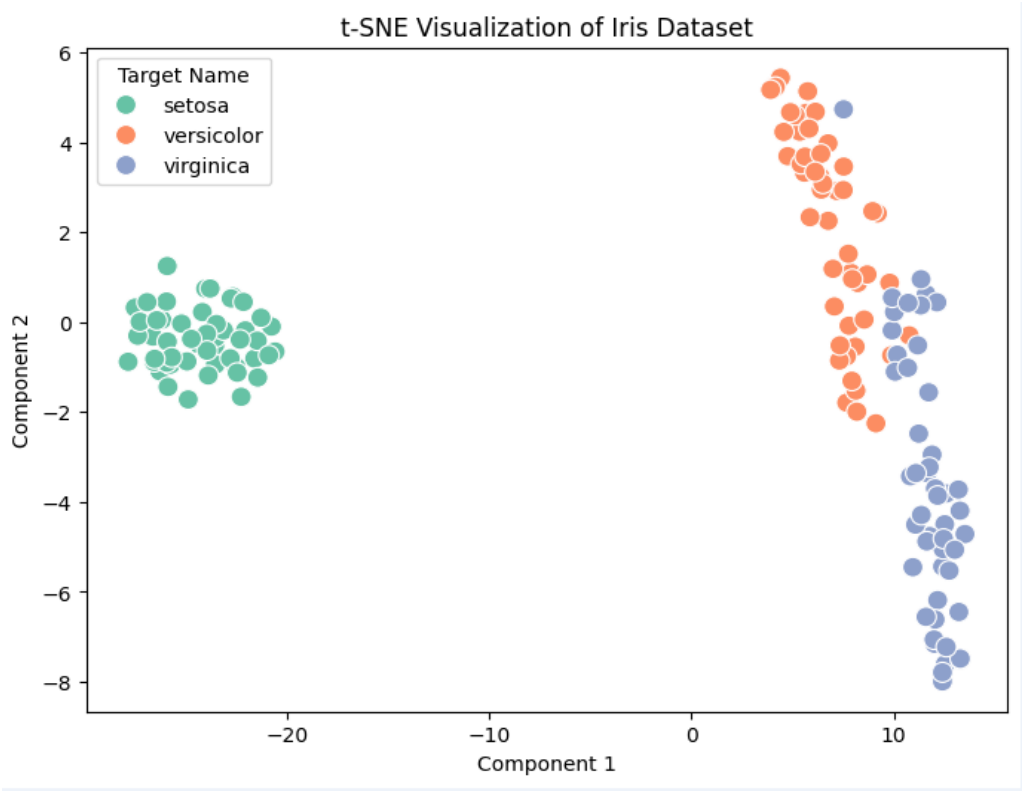


در ادامه ، الگوریتم t-SNE بر روی داده‌های ویژگی Xاجرا می‌شود `TSNE n_components=2,` . یک نمونه از کلاس TSNEایجاد می‌کند که تعداد مؤلفه‌ها را `n_components=2` برای نمایش دو بعدی و `random_state=34` برای تولید نتایج قابل تکرار تنظیم می‌کند. سپس `tsne.fit_transform(X)` داده‌های ویژگی Xرا به فضای دوبعدی 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.

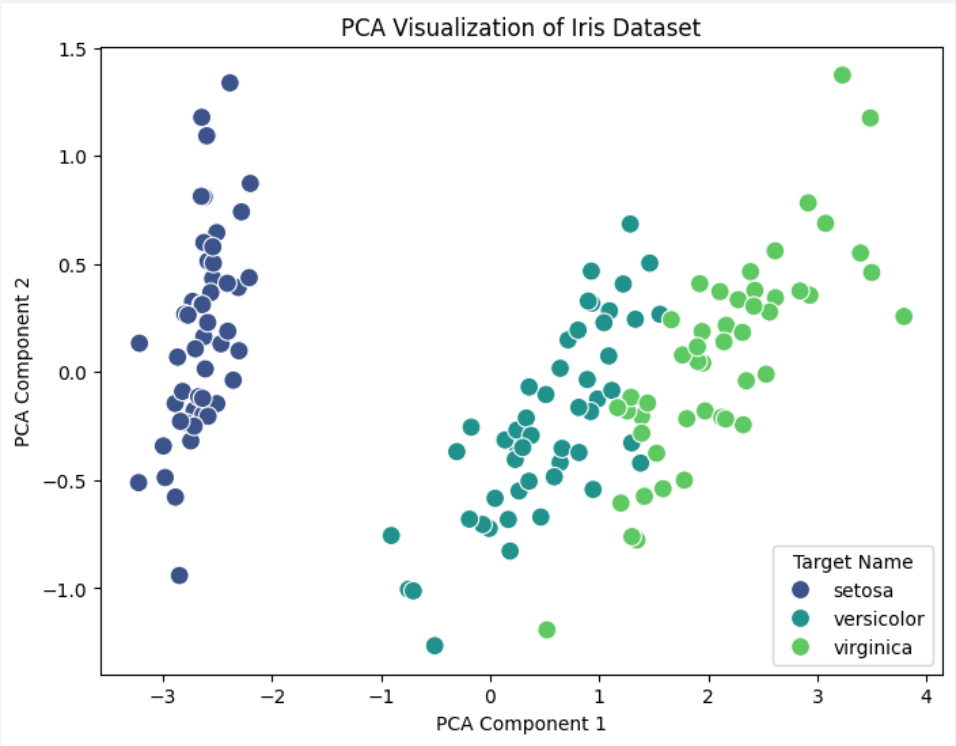
```
# Perform t-SNE
tsne = TSNE(n_components=2, random_state=34)
X_tsne = tsne.fit_transform(X)

# 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()
```



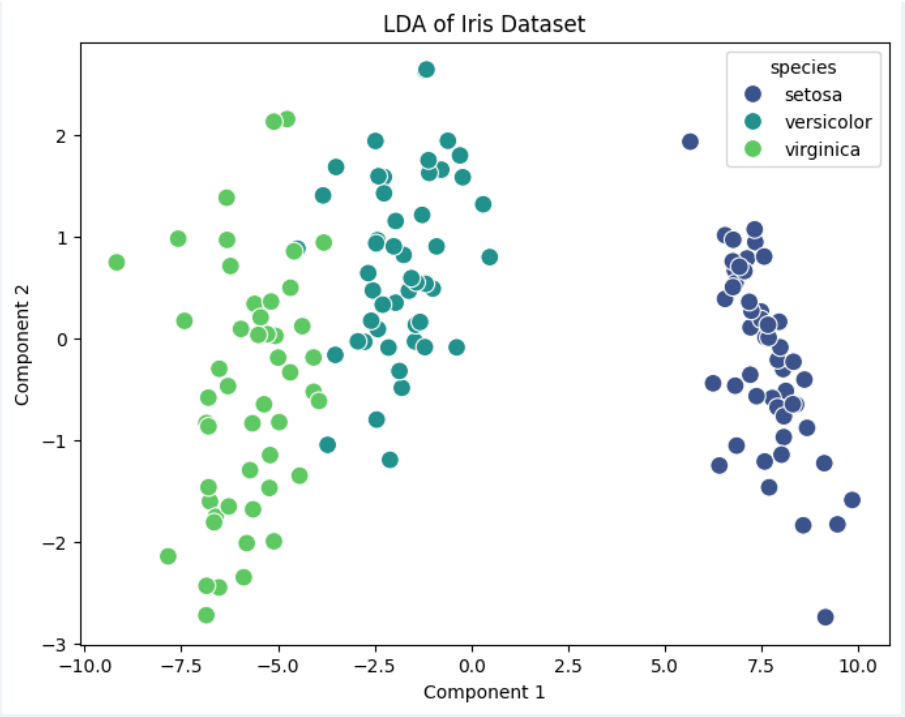
در مرحله بعد با استفاده از PCA و LDA به کاهش بعد می پردازیم که مراحل آنها مانند t-sne می باشد.

```
# 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)
```

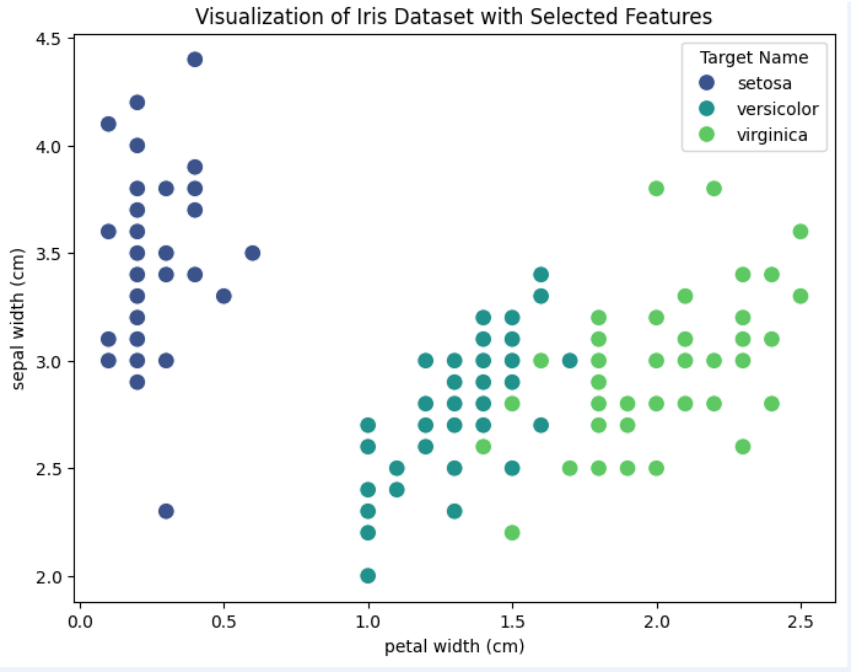



```
# Perform LDA
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)

# 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

۱. 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.

۲. 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.

۳. 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.

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.

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) ۸۰ درصد و آزمایش(test) ۲۰ 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 SVC با 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.

```python
# 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)
```

```
Accuracy on test data: 0.97

Confusion Matrix of test data:
[[10  0  0]
 [ 0  3  1]
 [ 0  0 16]]
Weights: [-0.46020088  0.33751306 -0.8639836  -0.93632798]
Bias: -1.4759502026067934

Confusion Matrix of all data:
[[50  0  0]
 [ 0 45  5]
 [ 0  1 49]]
```

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_scaledرا به فضای دو بعدی تبدیل می‌کند. ` `X_scaledباید 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:

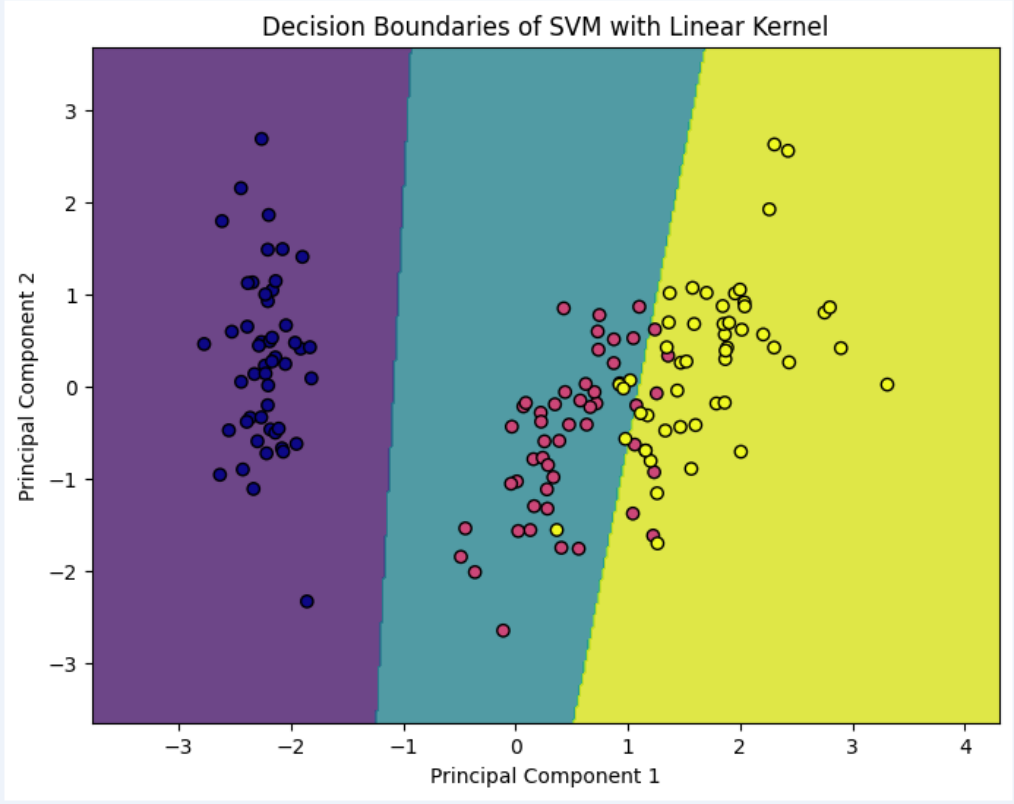
plt.figure: تعیین اندازه تصویر.

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.

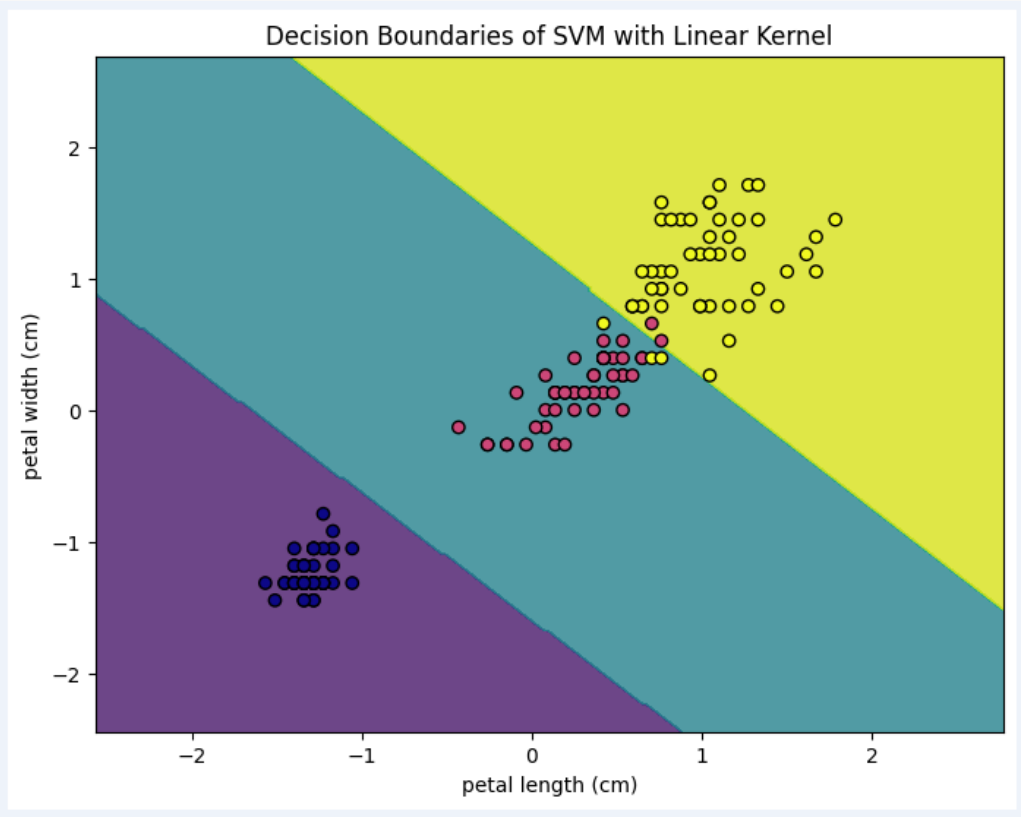
```
# Reducing the four features to two using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 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.

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 گیف به تصویر بکشید و لینک دسترسی مستقیم به فایل گیف را درون گزارش خود قرار دهید.

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 ۸۰ و قسمت آزمایش با درصد ۲۰ 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 ۱ تا ۱۰ 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 ۱ تا ۱۰ 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.

```
Accuracy with best degree: 1.0000
Degree 1: Accuracy = 1.0000
Degree 2: Accuracy = 0.9000
Degree 3: Accuracy = 0.9000
Degree 4: Accuracy = 0.9000
Degree 5: Accuracy = 0.9000
Degree 6: Accuracy = 0.9000
Degree 7: Accuracy = 0.9000
Degree 8: Accuracy = 0.9000
Degree 9: Accuracy = 0.9000
Degree 10: Accuracy = 0.8667
```

گیف با اسم q1-c در فایل موجود است.

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) Polynomial_kernel، Fit 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 گیف به تصویر بکشید پیوند دسترسی مستقیم آن را در گزارش خود قرار دهید. هم‌چنین، در مورد چالش اشباع محاسباتی در پیاده‌سازی هسته‌های با درجات بالاتر توضیحاتی ارائه کنید.

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 CVXOPTمی‌سازد:

- P:ماتریس متقارن مثبت که شامل ماتریس گرمی و بردارهای برچسب‌های کلاس است.
- q:بردار منفی یک‌ها.
- Aو b: ماتریس و بردار معادله مساوی که محدودیت‌های برچسب‌های کلاس را تعیین می‌کنند.
- Gو 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 multiplierها 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 SVM1مقادیر وزن‌ها (در صورت استفاده از هسته خطی)، بایاس، نتیجه حل مسئلهQP ، Lagrange multiplierها، بردارهای پشتیبان و پیش‌بینی‌ها را برمی‌گرداند.

بطور کلی تابع SVM1یک 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)
```

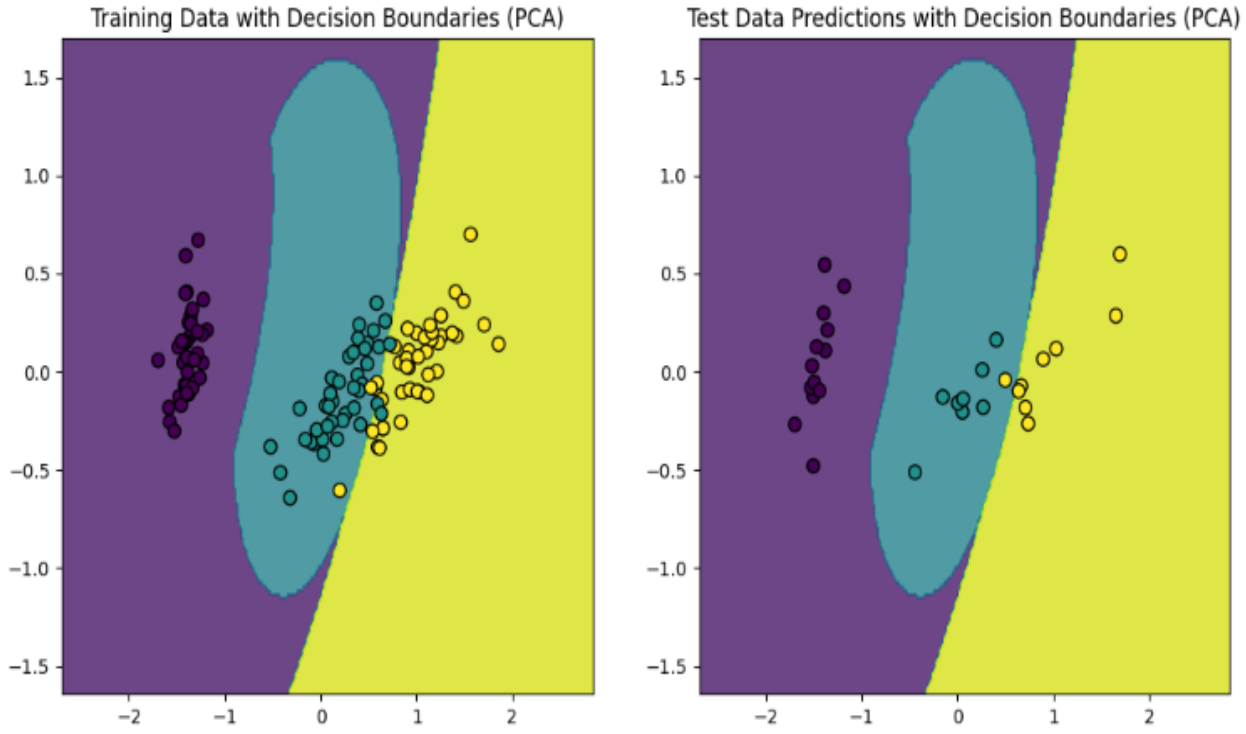
سپس مدل svm با استفاده از تمام ویژگی ها و هسته RBF آموزش می بیند و دقت ترین و تست محاسبه می شود.

```
# 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 ۹۸ و دقت تست به حدود ۹۷ 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()
```

```
Training Accuracy: 98.33%
Test Accuracy: 96.67%
```



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 ۲۰/۸۰.

```
# 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_svmپیاده‌سازی 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.

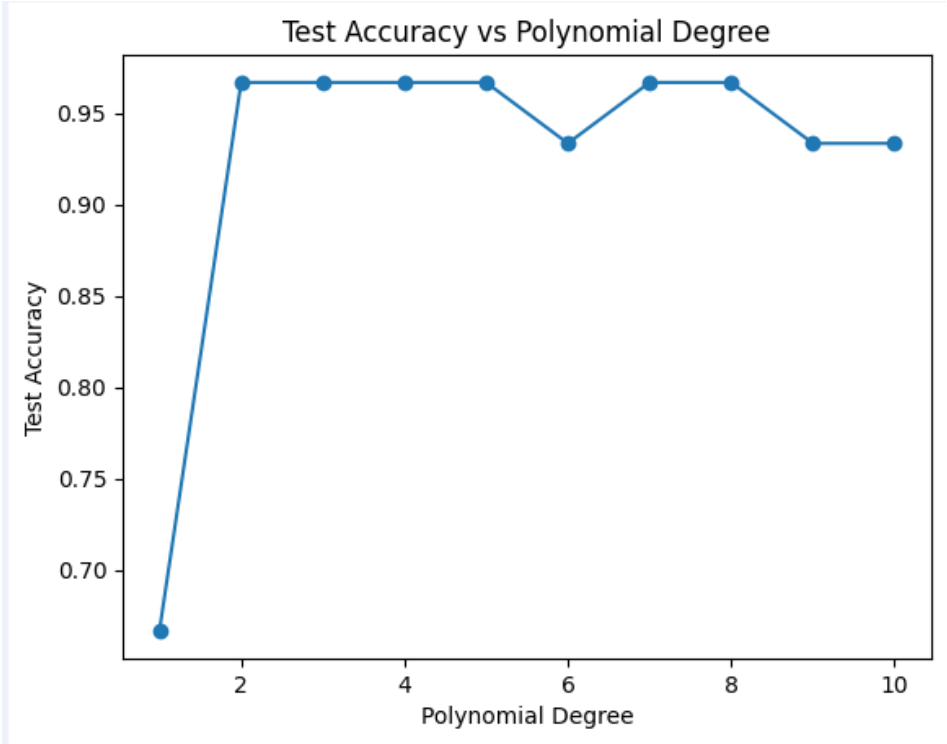
نمایش GIF ایجاد شده در محیط نوت‌بوک.

```
# 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:




```
GIF saved as 'Q1D_polynomial_kernel_degrees.gif'
Accuracy for degrees 1 to 10:
Degree 1: 0.67
Degree 2: 0.97
Degree 3: 0.97
Degree 4: 0.97
Degree 5: 0.97
Degree 6: 0.93
Degree 7: 0.97
Degree 8: 0.97
Degree 9: 0.93
Degree 10: 0.93
```

گیف با اسم q1-d در فایل موجود است.

۳ پرسش سه

مقالهٔ Credit Card Fraud Detection Using Autoencoder Neural Network برای پیاده‌سازی این قسمت در نظر گرفته شده است. پس از مطالعهٔ مقاله به سوالات زیر پاسخ دهید.

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

به طور خلاصه این مقاله به بررسی مسئله طبقه‌بندی داده‌های نامتوازن در زمینه تشخیص کلاهبرداری با کارت اعتباری می‌پردازد. برای متوازن‌سازی نمونه‌ها بین کلاس‌های اکثریت و اقلیت، از الگوریتم بیش‌نمونه‌گیری استفاده می‌شود که می‌تواند نویز ایجاد کند. در این راستا، یک الگوریتم شبکه عصبی 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 :

۱. 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.

۲. عدم توازن داده‌ها:

در بیشتر موارد، داده‌های تراکنش‌های مالی به شدت نامتوازن هستند؛ به این معنا که تعداد تراکنش‌های قانونی بسیار بیشتر از تعداد تراکنش‌های تقلبی است. این عدم توازن باعث می‌شود که مدل‌های سنتی 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.

۳. انتخاب ویژگی‌های بهینه:

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

۴. معیارهای ارزیابی مناسب:

انتخاب معیارهای مناسب برای ارزیابی عملکرد مدل‌ها در داده‌های نامتوازن یکی دیگر از 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 ٪۹۹,۹ آن شامل داده‌های عادی و تنها ٪۰,۱ آن شامل داده‌های غیرعادی است. اگر یک مدل طبقه‌بندی همه نمونه‌ها را به عنوان داده‌های عادی برچسب‌گذاری کند، دقت آن مدل ٪۹۹,۹ خواهد بود. این عدد ممکن است در نگاه اول بسیار خوب به نظر برسد، اما در واقعیت مدل نتوانسته هیچ‌یک از داده‌های غیرعادی را شناسایی کند و در شناسایی ناهنجاری‌ها کاملاً ناکارآمد است.

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

۱. استفاده از الگوریتم Oversampling :

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

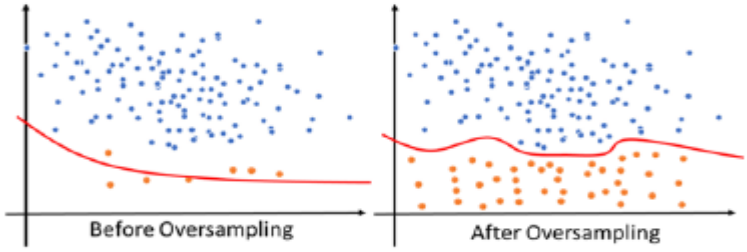


Fig. 3 Benefit of using oversampling

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.

۲. استفاده از شبکه‌های عصبی Autoencoder :

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

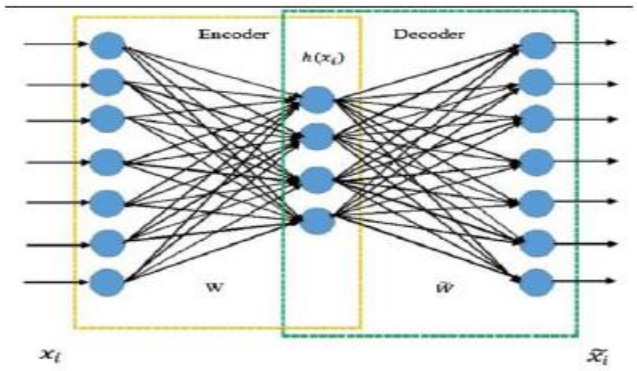


Fig. 1 architecture of autoencoder neural network

۳. استفاده از Denoising Autoencoder :

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

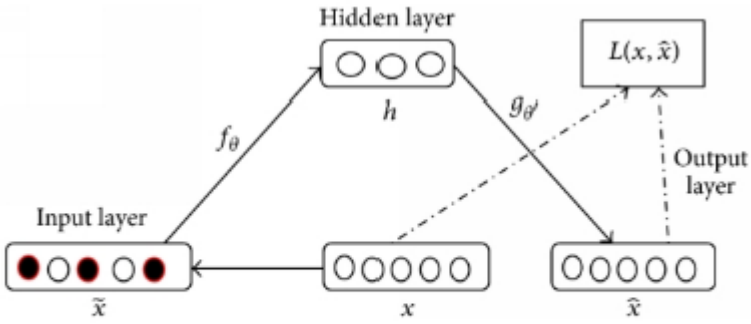


Fig. 2 Denoising autoencoder neural network

۴. 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) :

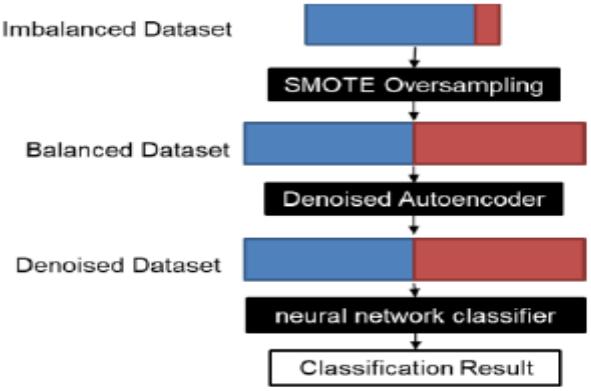


Fig. 5 Flowchart of the porcess

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

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

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

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

لایه‌ها شامل:

لایه ورودی با داده‌های نویزدار

چندین لایه کاملاً متصل با تعداد نورون‌های متغیر

استفاده از تابع زیان مربع برای بهینه‌سازی

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

Table 2. Model design for denoised autoencoder

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

۲. طبقه‌بند:

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

لایه‌ها شامل:

لایه ورودی با داده‌های نویزگیری شده

چندین لایه کاملاً متصل با تعداد نورون‌های متغیر

استفاده از تابع زیان انتروپی متقاطع SoftMax برای طبقه‌بندی نهایی

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

Table 3. Model design for classifier

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

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.

در ابتدا، از مجموعه داده‌ای که شامل ۲۸۳۱۵ تراکنش کارت اعتباری است و ۰٫۵٪ از آن‌ها به عنوان کلاهبرداری دسته‌بندی شده‌اند، استفاده کرده است. سپس با استفاده از بیش‌نمونه‌گیری، مجموعه داده را به یک مجموعه داده متعادل تبدیل کرده است. بعد از آن، از 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 ۲۰۱۳ توسط دارندگان کارت اروپایی انجام شده است. این مجموعه داده شامل تراکنش‌هایی است که در دو روز اتفاق افتاده است، ما ۴۹۲ تقلب را از بین ۲۸۴۸۰۷ تراکنش داریم. این مجموعه داده بسیار نامتوازن است، کلاس مثبت (تقلب‌ها) حدود ۰٫۱۷۲٪ 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 ۱ را در صورت وقوع تقلب و در غیر این صورت صفر می‌گیرد. (۲۸۴۸۰۷ ردیف و ۳۱ 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()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	...	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15	...	1.654067e-16	-3.568593e-16
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	...	7.345240e-01	7.257016e-01
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	...	-3.483038e+01	-1.093314e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	...	-2.283949e-01	-5.423504e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	...	-2.945017e-02	6.781943e-03
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	...	1.863772e-01	5.285536e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	...	2.720284e+01	1.050309e+01

8 rows × 31 columns

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 `data` DataFrameرا 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.

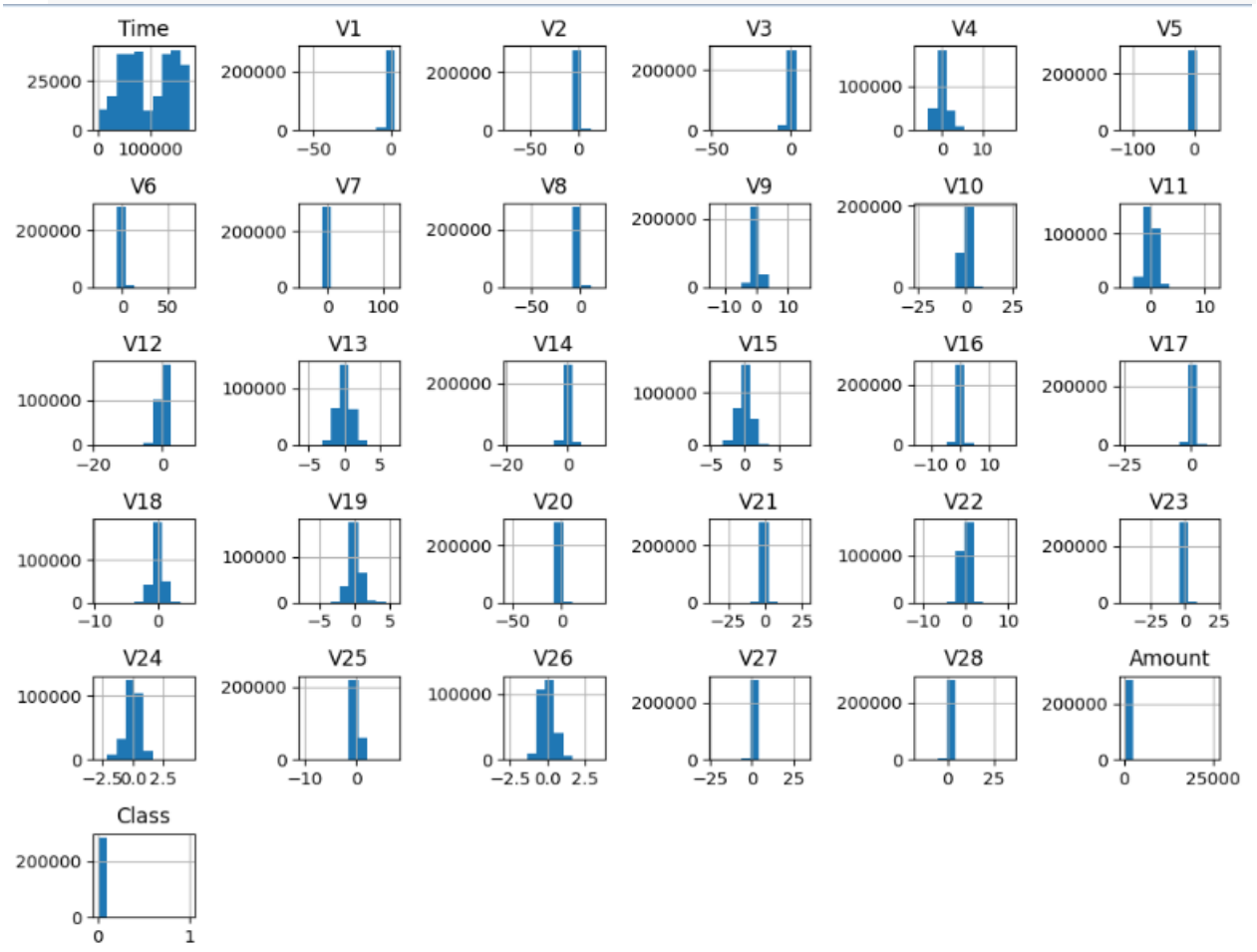
data.info(): اطلاعات کلی درباره DataFrame را نمایش می‌دهد که شامل تعداد ردیف‌ها، ستون‌ها، نوع داده هر ستون و میزان حافظه مصرفی است.

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)
```

```
raw dataset shape: (284807, 31)
X: (284807, 29)
y:(284807,)
Class
0    284315
1       492
Name: count, dtype: int64
```

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
```

```
((568630, 29), (568630,), (454904, 29), (113726, 29))
```

در ادامه به دیتا نویز گوسی با میانگین صفر وانحراف معیار ۰,۵ 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.

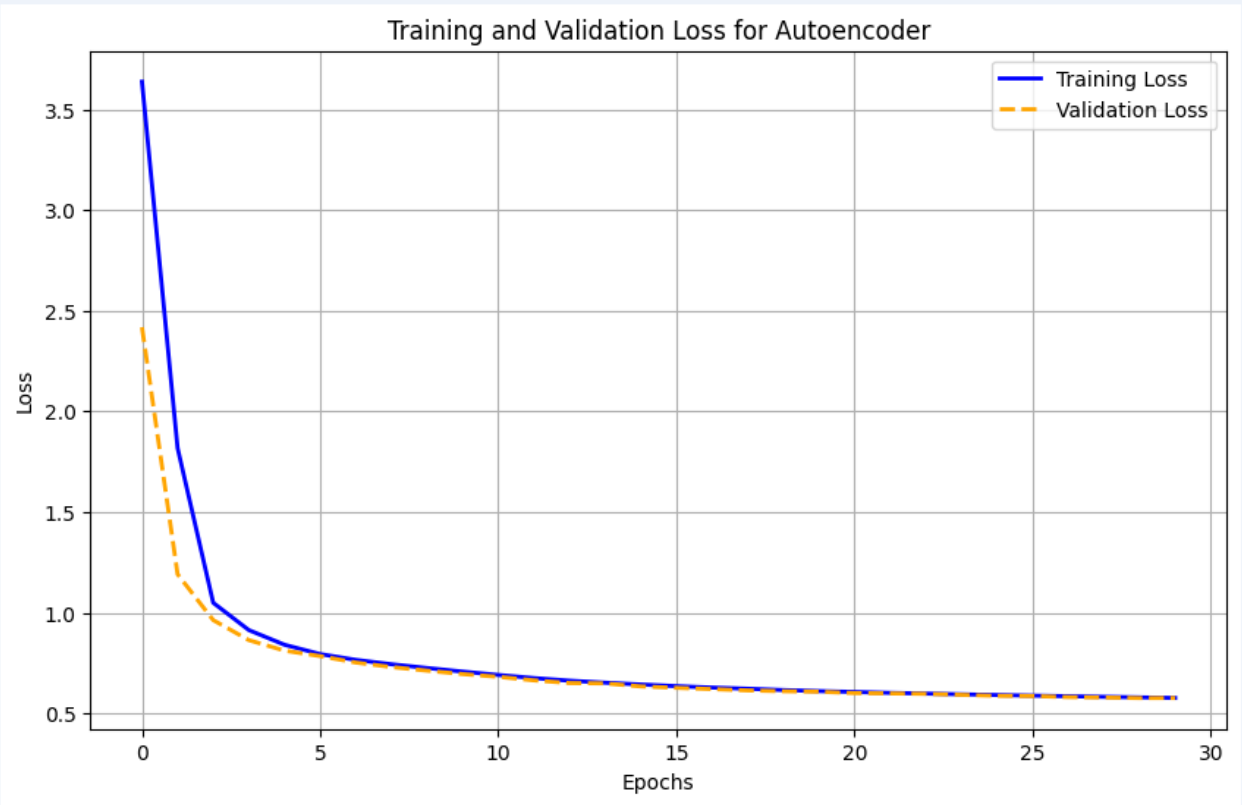
ایپاک و بقیه تنظیمات برای آموزش در کد زیر مشخص هستند.

```
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، reluو softmax ساخته می‌شود.

لایه ورودی مدل دارای ۱۰ ویژگی است و لایه خروجی دو نورون با تابع فعال‌سازی softmaxبرای طبقه‌بندی دوکلاسه دارد.

کامپایل مدل:

مدل با استفاده از بهینه‌ساز Adam و نرخ یادگیری ۰,۰۰۰۱ 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_lossمشاهده می‌شود، ذخیره می‌کند.

EarlyStopping آموزش را زمانی متوقف می‌کند که بهبود در val_loss به مدت ۱۰ دوره مشاهده نشود و بهترین وزن‌ها را بازیابی می‌کند.

آموزش مدل:

مدل با استفاده از داده‌های آموزشی رمزگذاری شده X_train_encodedو برچسب‌های یک‌داغ y_train_onehotبه مدت ۵۰ دوره آموزش داده می‌شود.

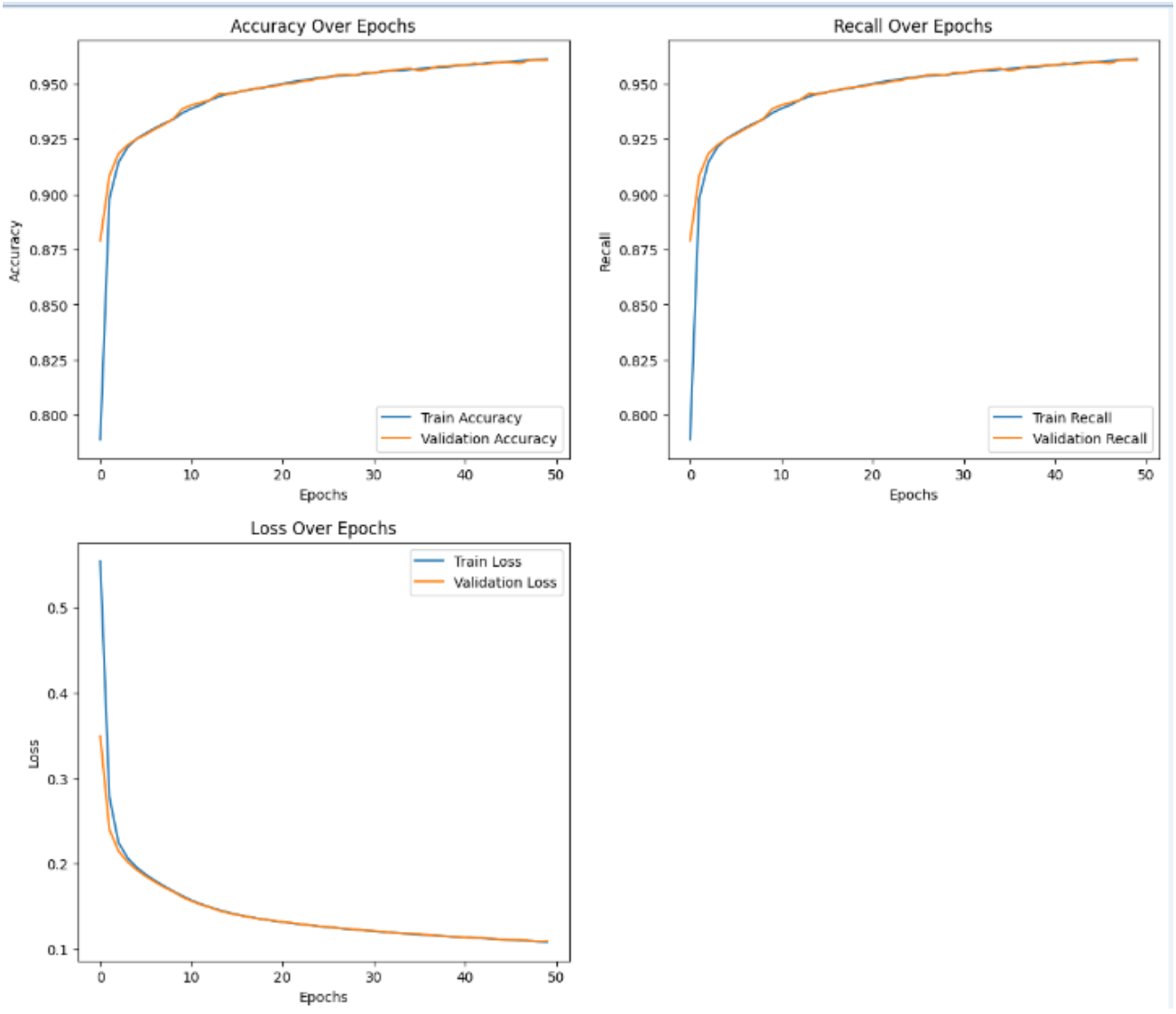
اندازه بسته‌ها ۲۰۰ و ۲۰٪ از داده‌ها برای اعتبارسنجی استفاده می‌شود.

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')
])
```

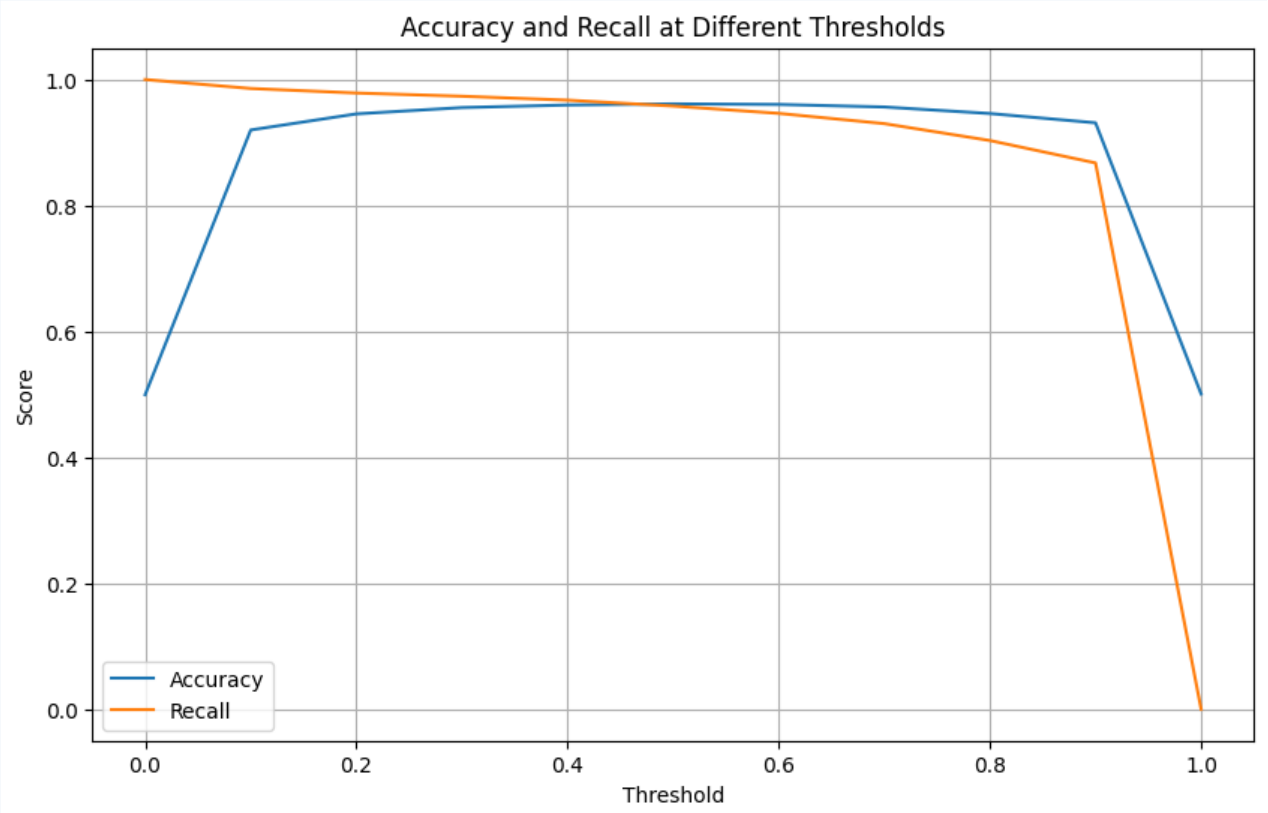
```
classifier.compile(optimizer=Adam(learning_rate=0.0001),
loss='categorical_crossentropy', metrics=['accuracy',Recall()])
# Define the ModelCheckpoint and EarlyStopping callbacks
checkpoint = ModelCheckpoint('best_model.keras', monitor='val_loss',
save_best_only=True, mode='min')
early_stopping = 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 یادآوری recallرا 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 ۰,۰ تا ۱,۰ با گام‌های ۰,۱ 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 x۶۱۰ 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)
```

```
3554/3554 [==============================] - 5s 1ms/step
Accuracy: 0.9613017251991629
Precision: 0.964386378543192
Recall: 0.9578828749515794
F1 Score: 0.9611236252815689
Confusion Matrix:
 [[54923  2009]
 [ 2392 54402]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96     56932
           1       0.96      0.96      0.96     56794

    accuracy                           0.96    113726
   macro avg       0.96      0.96      0.96    113726
weighted avg       0.96      0.96      0.96    113726
```

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 ۹۵ درصد داده‌ها به کلاس ۰ و ۵ درصد به کلاس ۱ تعلق داشته باشند، مدلی که همیشه کلاس ۰ را پیش‌بینی می‌کند، ۹۵ درصد دقت خواهد داشت، ولی عملاً هیچ نمونه‌ای از کلاس ۱ 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.

فرمول: $\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$

TP: True Positives (مثبت‌های واقعی)

FP: False Positives (مثبت‌های نادرست)

یادآوری:

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

فرمول: $\frac{TP}{TP+FN} = Recall$

FN: False Negatives (منفی‌های نادرست)

F1 Score:

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

فرمول: $\frac{Precision \times Recall}{Precision+Recall} \times 2 = F1\ Score$

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

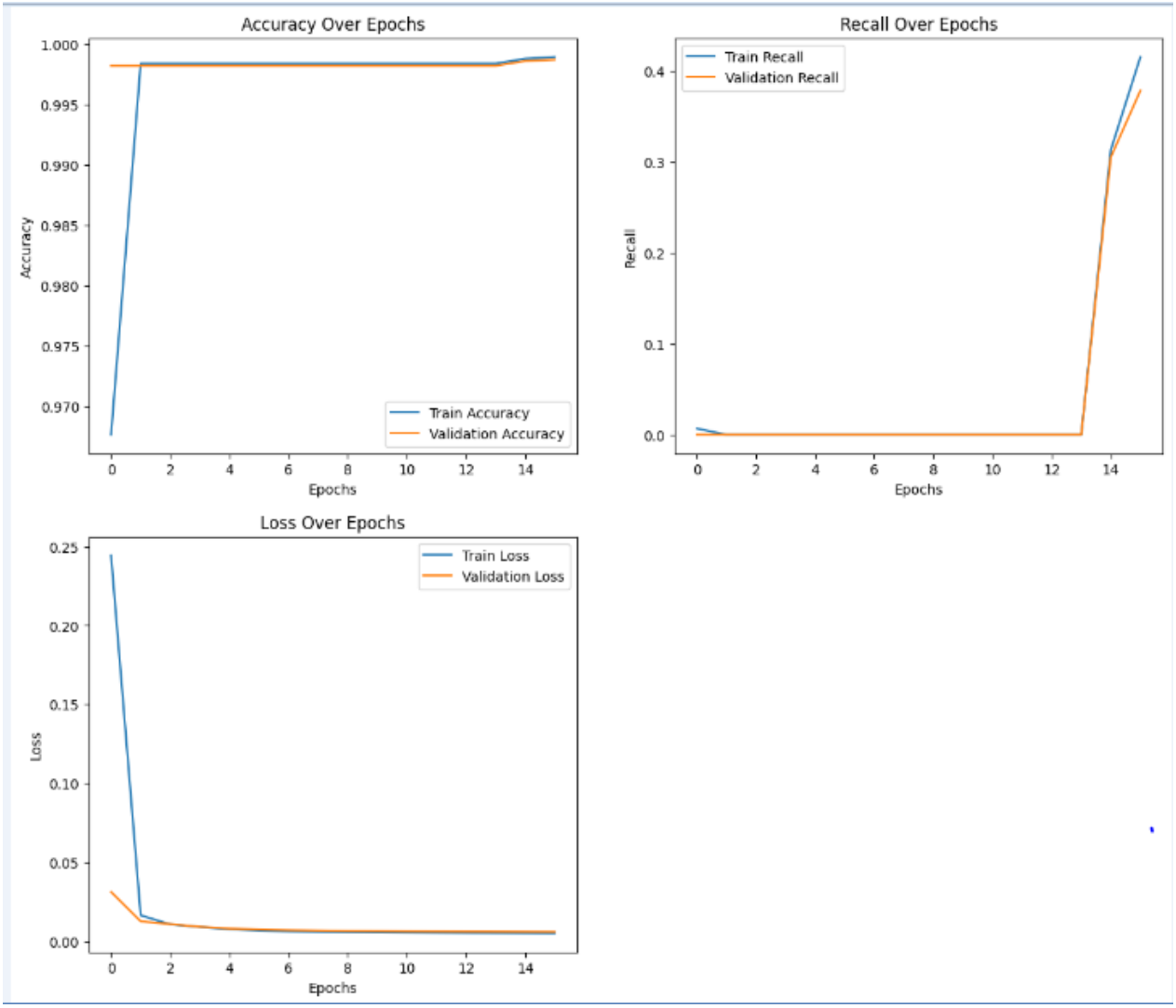
ROC Curve و AUC (Area Under the Curve):

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

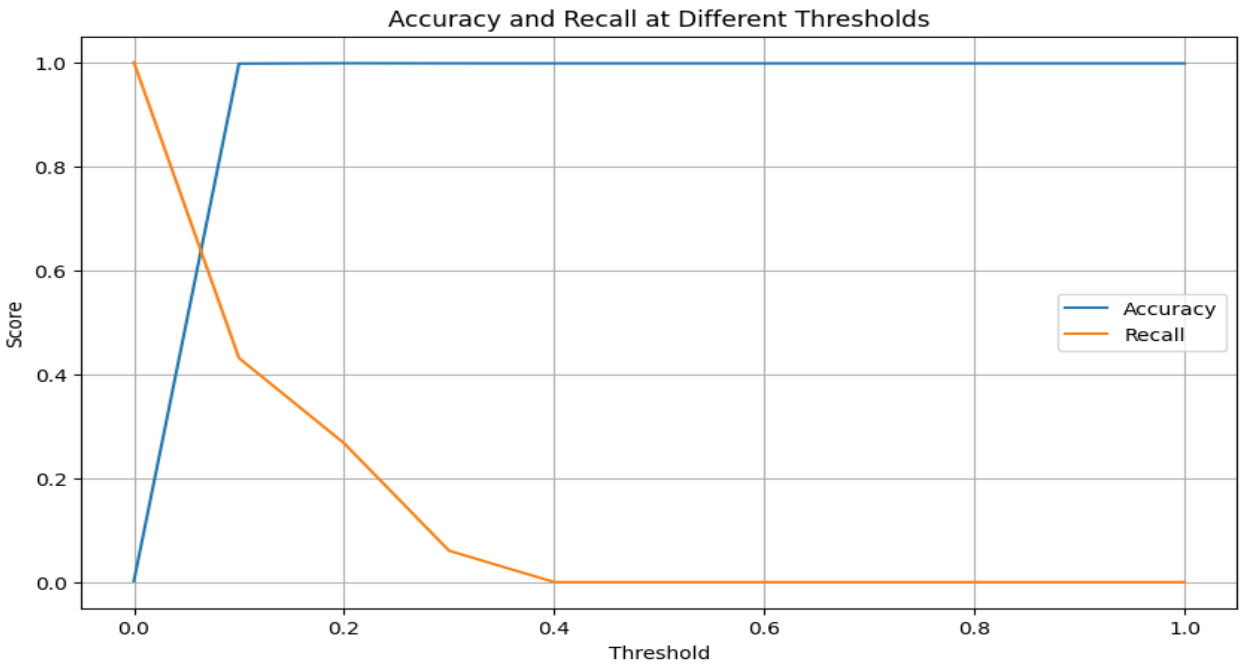
AUC یک مقدار عددی بین ۰ و ۱ است که عملکرد کلی مدل را در تمایز بین کلاس‌ها نشان می‌دهد.
AUC بالاتر به معنای عملکرد بهتر است.

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:




```
1781/1781 [==============================] - 3s 2ms/step
Accuracy: 0.9979635546504687
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
Confusion Matrix:
 [[56846     0]
 [  116     0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56846
           1       0.00      0.00      0.00       116

    accuracy                           1.00     56962
   macro avg       0.50      0.50      0.50     56962
weighted avg       1.00      1.00      1.00     56962
```



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

```
1781/1781 [==============================] - 2s 1ms/step
1781/1781 [==============================] - 3s 2ms/step
Confusion Matrix:
[[56843     3]
 [  109     7]]

Classification Report:
              precision    recall  f1-score   support

     Class 0       1.00      1.00      1.00     56846
     Class 1       0.70      0.06      0.11       116

    accuracy                           1.00     56962
   macro avg       0.85      0.53      0.56     56962
weighted avg       1.00      1.00      1.00     56962
```

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