درس یادگیری ماشین

پاسخ مینی‌پروژهٔ سوم

نام و نام خانوادگی	ایمان گندمی
شمارهٔ دانشجویی	۴۰۲۲۳۷۰۴
تاریخ	فروردین ۱۴۰۳

فهرست مطالب

سوال اول 3

آ. 3

ب. 9

ج 12

د. 14

سوال سوم 18

آ. 18

ب. 20

ج. 21

د. 26

ه. 28

و. 29

لینک پوشهٔ گیت‌هاب:

https://github.com/ImanGandomi/MachineLearning2024/tree/main/ml_MiniProject_3

لینک گوگل گولب:

https://colab.research.google.com/drive/1gzlxY9jaKf8ka0gAvrTyIbmCICoAcHqe?usp=sharing

سوال اول

آ.

در مرحلهٔ اول دیتاست را فراخوانی کنید و اطلاعاتی نظیر ابعاد، تعداد نمونه ها، میانگین، واریانس و همبستگی ویژگی ها را به دست آورید و نمونه های دیتاست را به تصویر بکشید (مثلا با استفاده از t-SNE). سپس، با توجه به اطلاعات عددی، آماری و بصری بدست آمده، تحلیل کنید که آیا کاهش ابعاد می تواند در این دیتاست قابل استفاده باشد یا خیر.

ابتدا کتابخانه‌های مورد نیاز را به صورت زیر فراخوانی می‌کنیم.

```
#import library
import numpy as np
import itertools
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.manifold import TSNE
from sklearn.svm import SVC
```

سپس دیتاست مورد نظر که مورد اطلاعاتی از یک نوع گل است را فراخوانی می‌کنیم. این دیتاست در کتابخانه sklearn موجود است. البته به صورت csv نیز قابل دسترسی بوده اما فراخوانی از کتابخانه انتخاب شده است.

```
from sklearn import datasets
iris = datasets.load_iris()
```

برای دسترسی به اطلاعات این دیتاست از دستورهای مختلفی می‌توان استفاده نمود. اطلاعاتی مانند تعداد نمونه‌ها، تعداد کلاس‌ها، میانگین و .. از اطلاعاتی مفیدی هستند که پیش از کار باید آن‌ها را بدانیم.

اولین کار آگاهی از کلیدهای مهم دیتاست است. با استفاده از دستور keys می‌توان موارد مهم دیتاست را دید. از مهم‌ترین این کلیدها، داده‌های مربوط به data, target, target_names, freature_names هستند. با دستور دوم، ابعاد داده‌های مربوط به ویژگی استخراج شده است که برابر 4 بوده و تعداد 150 داده نیز در این دیتاست وجود دارد. سپس لیبل‌های داده‌ها قابل مشاهده است که نشان می‌دهد اعداد 0 و 1 و 2 برای تقسیم بندی به سه کلاس مختلف هستند. اسامی سه کلاس این دیتاست به صورت setosa, versicolor, verginica است که مربوط به سه نوع گل است. اسامی مربوط به 4 ویژگی نیز به صورت زیر قابل مشاهده است که همگی به صورت واحد سانتی‌متر هستند.

```
print(iris.keys())
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename', 'data_module'])
```

```
iris['data'].shape
```

```
(150, 4)
```

```
iris['target']
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
iris['target_names']
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
iris['feature_names']
```

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

اما دانستن اطلاعات مفید دیگری نظیر میانگین، واریانس و سایر محاسباتی ریاضی نیز پیش از ورود به بخش طبقه‌بندی مفید است. برای اطلاع از این محاسباتی و برای راحتی کار با داده‌های بصورت data frame ابتدا دیتاست را به این قالب در می‌آوریم. سپس با استفاده از دستور describe اطلاعات مربوط به ویژگی‌های ریاضیاتی دیتاست استخراج می‌شود که به صورت زیر قابل ملاحظه است. میانگین، واریانس، مقادیر بیشینه و کمینه برای هر ویژگی و تعداد هر ویژگی از دیتاهای مهم هر دیتاست هستند که باید مورد توجه قرار گیرند. مشاهده می‌شود که تعداد هر چهار ویژگی برابر 150 است که نشان می‌دهد هیچ داده‌ای میس‌فیچر ندارد. همچنین با توجه به میانگین و واریانس هر ویژگی، می‌توان نتیجه گرفت که با اینکه این مقادیر کاملا یکسان نیستند و برای هر ویژگی متفاوت است اما این تفاوت فاحش نیست و به نظر گسستگی ویژگی برای دیتاست زیاد نیست. مقادیر بیشینه و کمینه نیز برای هر ویژگی قابل مشاهده است.

```
data = iris.data
feature_names = iris.feature_names

# Convert to pandas DataFrame
df_data = pd.DataFrame(data, columns=feature_names)
df_data['target'] = iris.target

# Compute statistics
statistics = df_data.describe()

# Print results
print("Summary statistics:")
print(statistics)
```

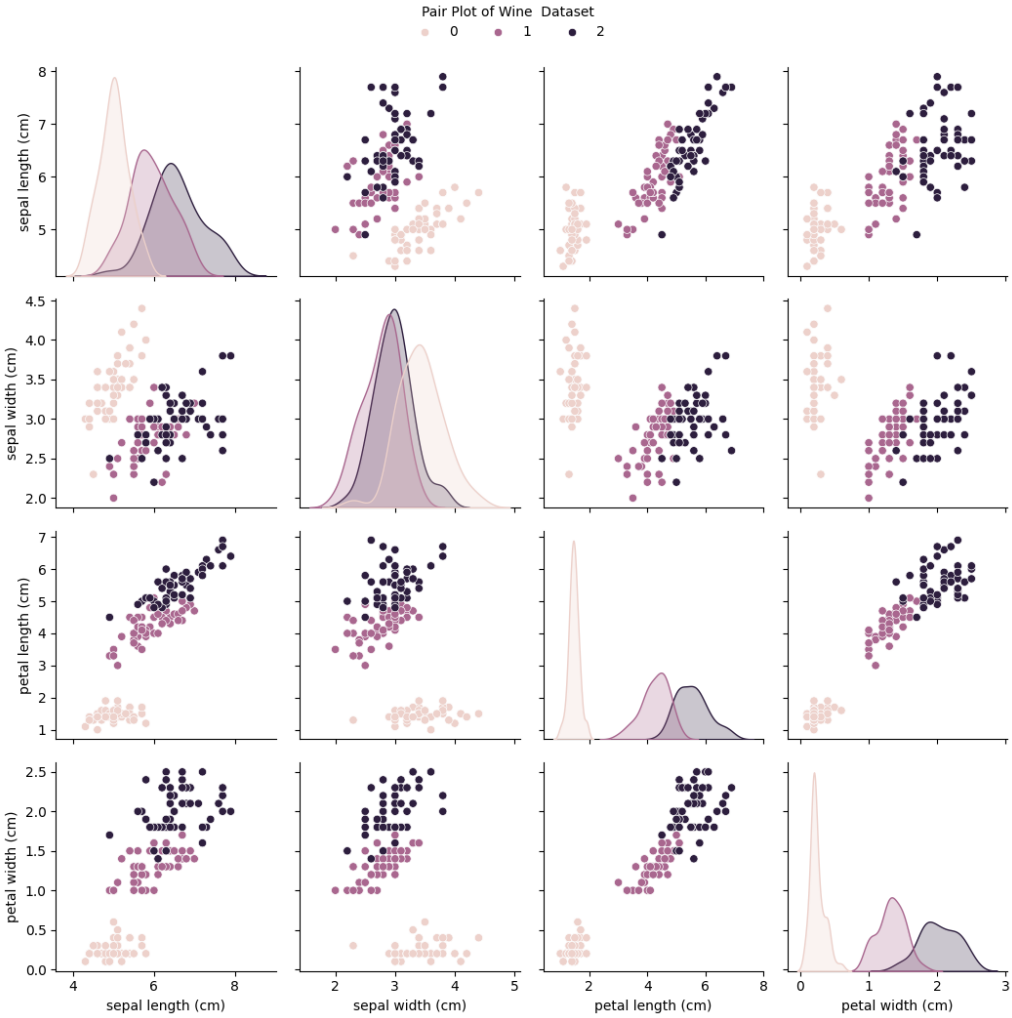
```
Summary statistics:
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)      target
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

در ادامه نیاز است که همبستگی ویژگی‌های این دیتاست نیز مورد بررسی قرار بگیرد. این کار با استفاده از کد زیر انجام می‌شود. در این دستور با استفاده از دیتاستی که به صورت دیتافریم درآوردیم پلات زیر رسم می‌شود.

```
import seaborn as sns

ax = sns.pairplot(df_data, hue='target')
sns.move_legend(
    ax, "lower center",
    bbox_to_anchor=(.5, 1), ncol=3, title="Pair Plot of Wine  Dataset", frameon=False)

plt.tight_layout()
plt.show()
```

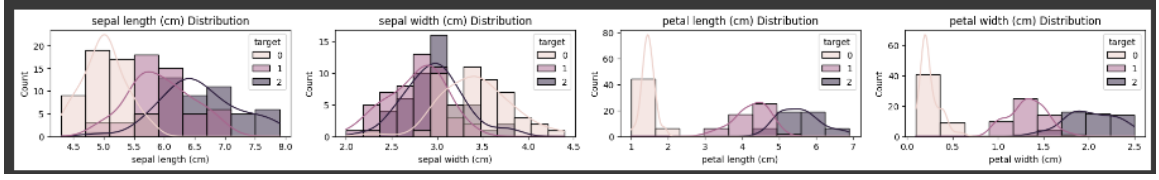
در این پلات تمام ویژگی‌ها نسبت به یکدیگر رسم شده است. در روی قطر اصلی این پلات نیز توزیع هر ویژگی برای سه کلاس مورد نظر رسم شده است. این توزیع‌ها از اهمیت بالایی برخوردارند و اطلاعات مفیدی می‌توان از آن‌ها استخراج کرد. توزیع مربوط به دو ویژگی اول یعنی sepal_length و sepal_width نشان از سختی طبقه بندی در صورت استفاده از این دو ویژگی می‌دهد چون توزیع این دو ویژگی بسیار سه کلاس بسیار نزدیک به یکدیگر بوده و سخت است که تنها با یکی از این دو ویژگی یا حتی با استفاده از هر دو کار طبقه‌بندی را انجام داد. اما دو ویژگی سوم و چهارم یعنی petal_length و petal_wigth توزیع متفاوت‌تری دارند. توزیع این دو ویژگی برای کلاس 0 کاملا از دو کلاس دیگر جداست و توزیع متفاوتی دارد. این نشان می‌دهد که با استفاده از این دو ویژگی کار طبقه‌بندی کلاس 0 از دو کلاس دیگر کار سختی نیست. حتی دو کلاس دیگر نیز خیلی توی هم نیستند. پلات‌های روی سایر قسمت‌های ماتریس پلاتی نیز این موضوع را تایید می‌کنند که ویژگی‌های 3 و 4 توزیع جداتری

برای هر کلاس دارند و کار طبقه‌بندی با استفاده از این دو ویژگی راحت‌تر ممکن است بشود و هر پلاتی که با استفاده از یکی از این دو ویژگی رسم شده است، جداپذیری را بهتر نشان داده است.



نمودار زیر به نحوی دیگر توزیع متفاوت ویژگی‌های 3 و 4 را نسبت به سایر ویژگی‌ها نشان می‌دهد.

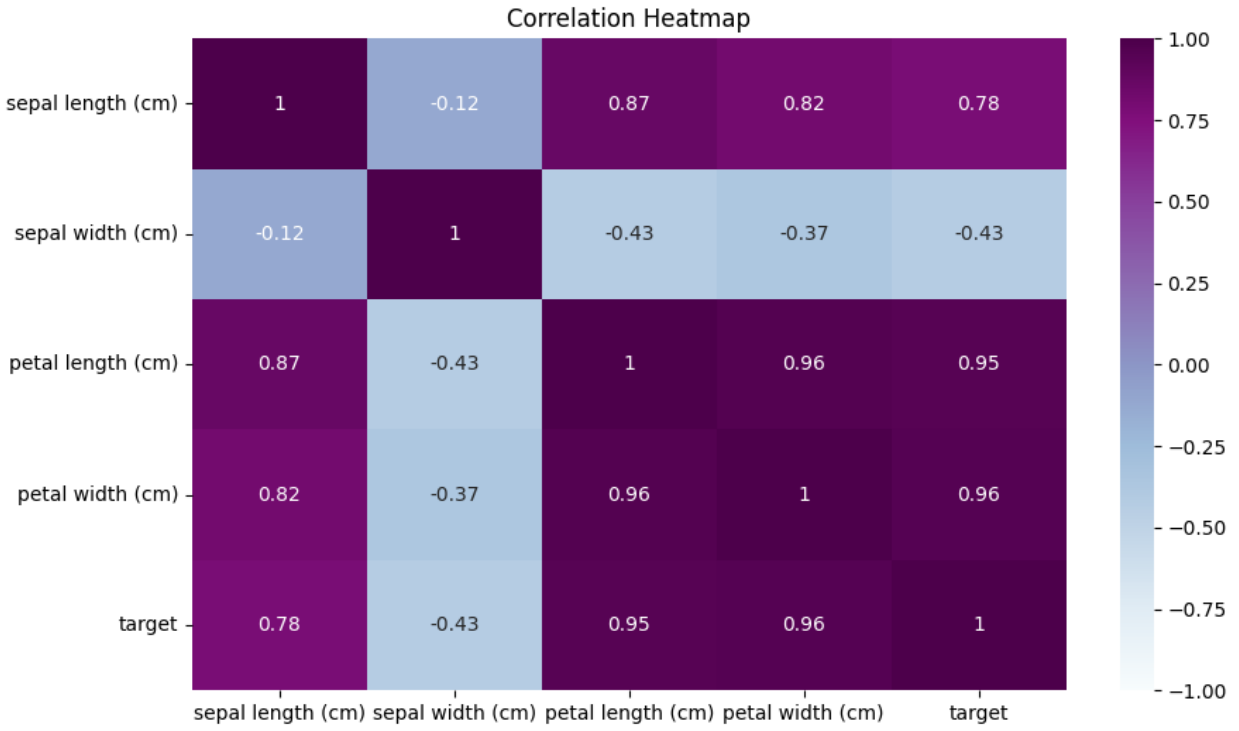
```
column_names = df_data.columns
column_names = column_names.to_list()

plt.figure(figsize=(18, 9))
for i, feature in enumerate(column_names[:-1]):
    plt.subplot(4, 4, i + 1)
    sns.histplot(data=df_data, x=feature, hue='target', kde=True)
    plt.title(f'{feature} Distribution')
plt.tight_layout()
plt.show()
```



اما برای اینکه همبستگی ویژگی‌ها را با یکدیگر به صورت عددی مورد بررسی قرار دهیم با استفاده از دستور زیر ماتریس همبستگی را پلات می‌کنیم.

```
corr = df_data.corr()
plt.figure(figsize=(10,6))
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values,
            cmap="BuPu",
            vmin=-1,
            vmax=1,
            annot=True)
plt.title("Correlation Heatmap")
plt.show()
```

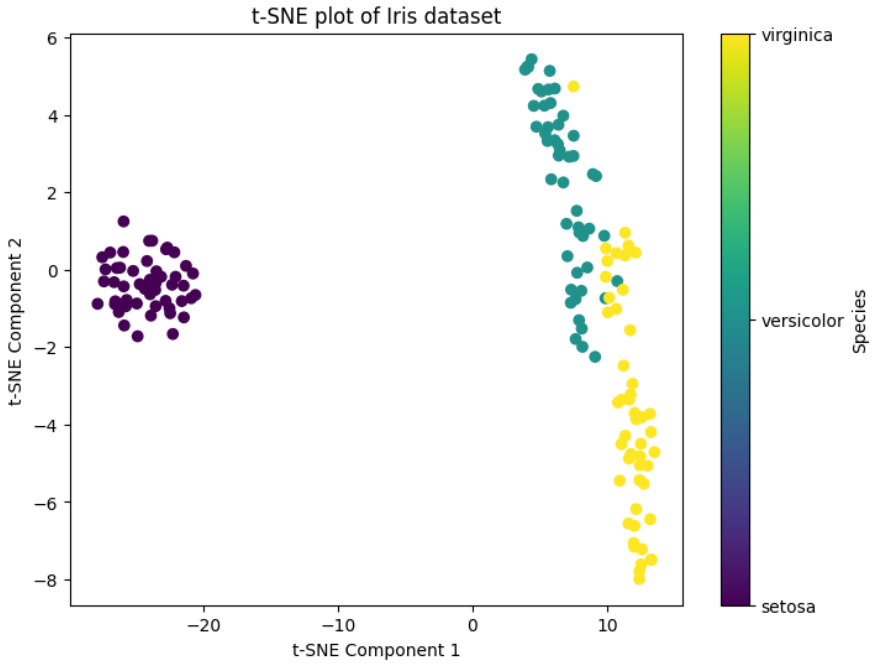
ماتریس همبستگی به صورت زیر قابل مشاهده است. مشاهده می‌شود که ویژگی 2 با سه ویژگی دیگر همبستگی کمتری دارد در حالی که سه ویژگی 1 و 3 و 4 با یکدیگر همبستگی زیادی دارند. به خصوص ویژگی‌های 3 و 4 که مقداری برابر 0.96 دارند. از این ماتریس و هم‌چنین توزیع ویژگی‌ها که پیش از این بررسی شد می‌توان پیش بینی کرد که اگر بخواهیم تنها دو ویژگی را برای طبقه‌بندی انتخاب کنیم، احتمالا دو ویژگی 2 و 3 در انتخاب اول و دو ویژگی 3 و 4 در انتخاب دوم، می‌تواند ویژگی‌های بهتری باشند.



در ادامه با توجه به اینکه 4 ویژگی داریم، اگر بخواهیم دیتاها را به تصویر بکشیم باید از کاهش بعد برای رسم در صفحه استفاده کنیم. برای اینکار از t_SNE استفاده می‌کنیم که یکی از روش‌های کاهش بعد برای رسم است. ابتدا مقادیر X و y را در دو متغیر جدا ذخیره می‌کنیم. سپس با استفاده از دستور TSNE تعداد بُعد برای رسم در صفحه را برابر 2 در نظر می‌گیریم و در نهایت با استفاده از دستور fit این کاهش بعد را برای داده‌ها X اعمال می‌کنیم. نمودار مربوط به این پلات به صورت زیر قابل مشاهده است.

```
X = iris.data
y = iris.target

tsne = TSNE(n_components=2, random_state=0)
X_tsne = tsne.fit_transform(X)

plt.figure(figsize=(8, 6))
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap='viridis')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.title('t-SNE plot of Iris dataset')
plt.colorbar(label='Species', ticks=range(3), format=plt.FuncFormatter(lambda val, loc: iris.target_names[val]))
plt.show()
```

همانطور که دیده می‌شود با کاهش بعد صورت گرفته، کلاس setosa از دو کلاس دیگر تفکیک‌پذیری بیشتری دارد و چالش طبقه‌بندی بر روی دو کلاس دیگر ممکن است ایجاد شود. با استفاده از این کاهش بعد تفکیک‌پذیری کاملا قابل مشاهده است. بنابراین می‌توان با استفاده از اطلاعات به دست آمده در تحلیل‌های انجام شده تا به اینجای کار نتیجه گرفت که لازم نیست که حتما از همهٔ ویژگی‌های این دیتاست برای کار طبقه‌بندی استفاده کنیم و بهره‌برداری از کاهش بعد به صورت انتخاب ویژگی یا ترکیب ویژگی‌ها می‌تواند بسیار نتیجهٔ مناسبی دربرداشته باشد.



ب.

در این قسمت با توجه به نتیجه‌گیری انجام شده در قسمت قبل که کار کاهش بعد می‌تواند بسیار سودمند باشد برای این مجموعه‌داده، ابا استفاده از روش PCA کار کاهش بعد دیتاست Iris را صورت داده‌ایم. این روش روشی ساده و قابل فهم است که داده‌ها را به مولفه‌های اصلی کاهش داده و کارایی محاسباتی بالایی دارد و برای مجموعه داده‌های کوچک مثل آیریس مناسب است. از دلایل دیگر برای استفاده از این روش این است که PCA بر حفظ مولفه‌هایی تمرکز دارد که بیشترین واریانس را در داده‌ها توضیح می‌دهند، که به معنای حفظ ساختار اصلی داده‌ها است. همچنین مجموعه داده آیریس به طور کلی ساده و تا حد زیادی قابل تفکیک خطی است، بنابراین PCA که یک روش خطی است، به خوبی در اینجا کار می‌کند. این کاهش بعد در سلول اول کد زیر به این صورت انجام شده است که بعد کاهش یافته با اندازهٔ 2 در نظر گرفته شده است.

در ادامه با استفاده از الگوریتم SVM، با هستهٔ خطی روی مجموعه‌دادهٔ کاهش بعد داده‌شده آموزش انجام شده است. طبقه‌بندی انجام شده به صورت زیر قابل مشاهده است که تعداد support vector های کلاس‌های 0 تا 2 برابر 1 و 14 و 12 است.

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
```

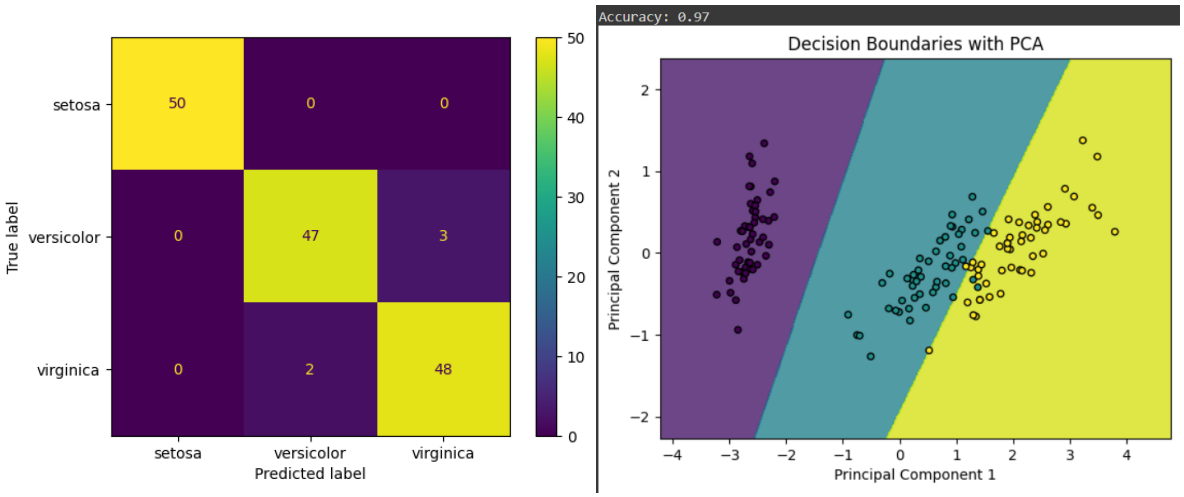
```
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import numpy as np


clf = SVC(kernel='linear')
clf.fit(X_pca, y)
```

```
SVC(kernel='linear')
```

```
clf.n_support_
```

```
array([ 1, 14, 12], dtype=int32)
```

مطابق خواسته مسئله پس از آموزش مجموعه‌داده ماتریس در هم ریختگی و مرزهای تصمیم‌گیری به صورت زیر قابل مشاهده‌اند. تمامی داده‌های مربوط به کلاس setosa به خوبی پیش‌بینی شده‌اند در حالی که پیش‌بینی دو کلاس دیگر برای چند نمونه دچار اشتباه شده است. این نتیجه از روی نمودار توزیعی مجموعه‌داده پیش و پس از کاهش بعد قابل حد بود چون دو کلاس 1 و 2 دارای توزیعی تا حدودی نزدیک هم و چالشی داشتند. داده‌هایی که برای این دو کلاس اشتباه تشخیص داده شده‌اند از روی ماتریس درهم‌ریختگی و مقایسه با نمودار دارای مرز تصمیم گیری قابل مشاهده‌اند. مقدار accuracy محاسبه‌شده برای این طبقه‌بندی برابر 0.97 محاسبه شده است.



اما برای اینکه بتوانیم مقایسه‌ای میان طبقه‌بندی صورت گرفته با استفاده از داده‌های کاهش‌بعد یافته به روش PCA با داده‌های اصلی دیتاست داشته باشیم، طبقه‌بندی‌های دیگری انجام داده‌ایم. در این طبقه‌بندی‌ها سعی شده است که کار کاهش بعد با انتخاب ویژگی صورت بگیرد. به این صورت که با توجه به این که کلا 4 ویژگی داریم، هر بار 2 ویژگی و به طور کلی 6 بار این زوج ویژگی انتخاب شده است و طبقه‌بندی صورت گرفته است.

در این قسمت سعی شده است با استفاده از کد زیر کار طبقه‌بندی روی دیتاست ناشی از انتخاب ویژگی برای 6 مرتبه صورت بگیرد. نتایج به صورت زیر قابل مشاهده است.

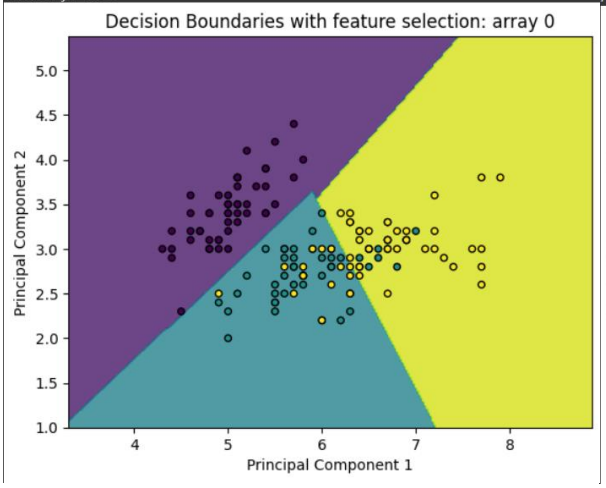
```
import numpy as np
import itertools


# Get all combinations of 2 columns out of 4
combinations = list(itertools.combinations(range(X.shape[1]), 2))

# Generate the 6 unique arrays
arrays = [X[:, combination] for combination in combinations]

# Example of accessing individual arrays
# X_12 = arrays[0]
# X_13 = arrays[1]
# X_14 = arrays[2]
# X_23 = arrays[3]
# X_24 = arrays[4]
# X_34 = arrays[5]
```

```
combinations
```

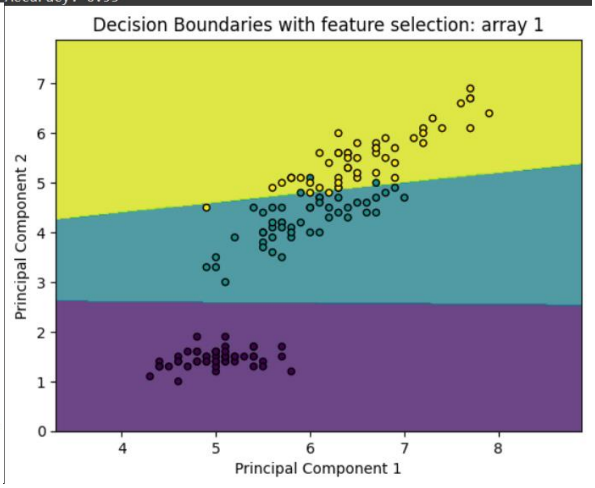
```
[(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]
```

```
for i in range(6):
  clf = SVC(kernel='linear')
  clf.fit(arrays[i], y)
  plot_decision_boundary(arrays[i], y, clf, f'Decision Boundaries with feature selection: array {i}')
  print("================================================================================")
  print()
```

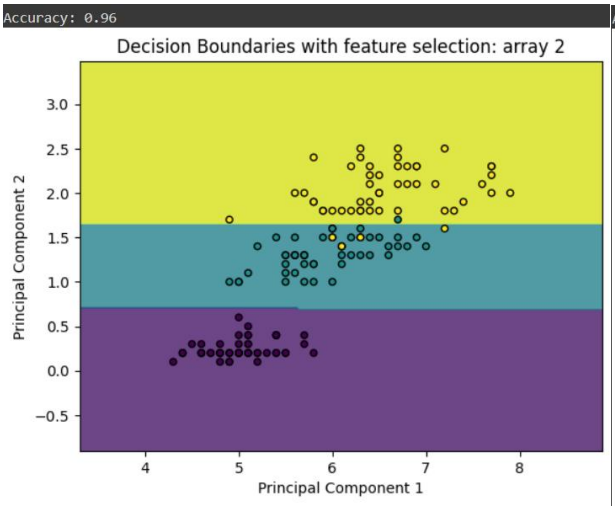
Accuracy: 0.82



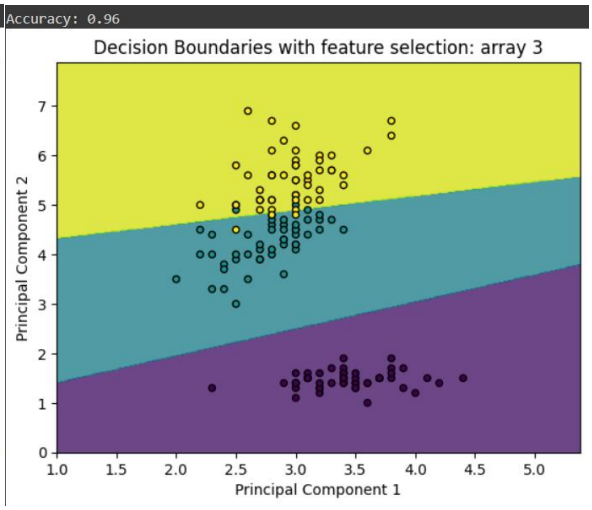
Accuracy: 0.95



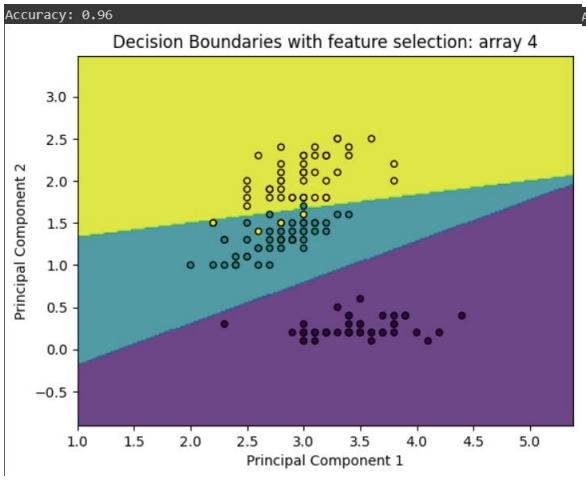
Accuracy: 0.96



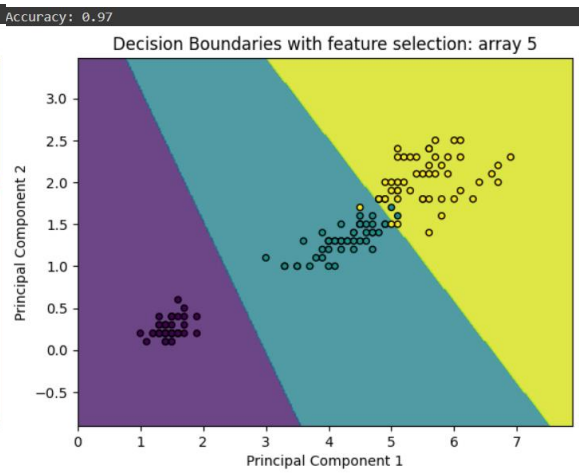
Accuracy: 0.96



Accuracy: 0.96



Accuracy: 0.97



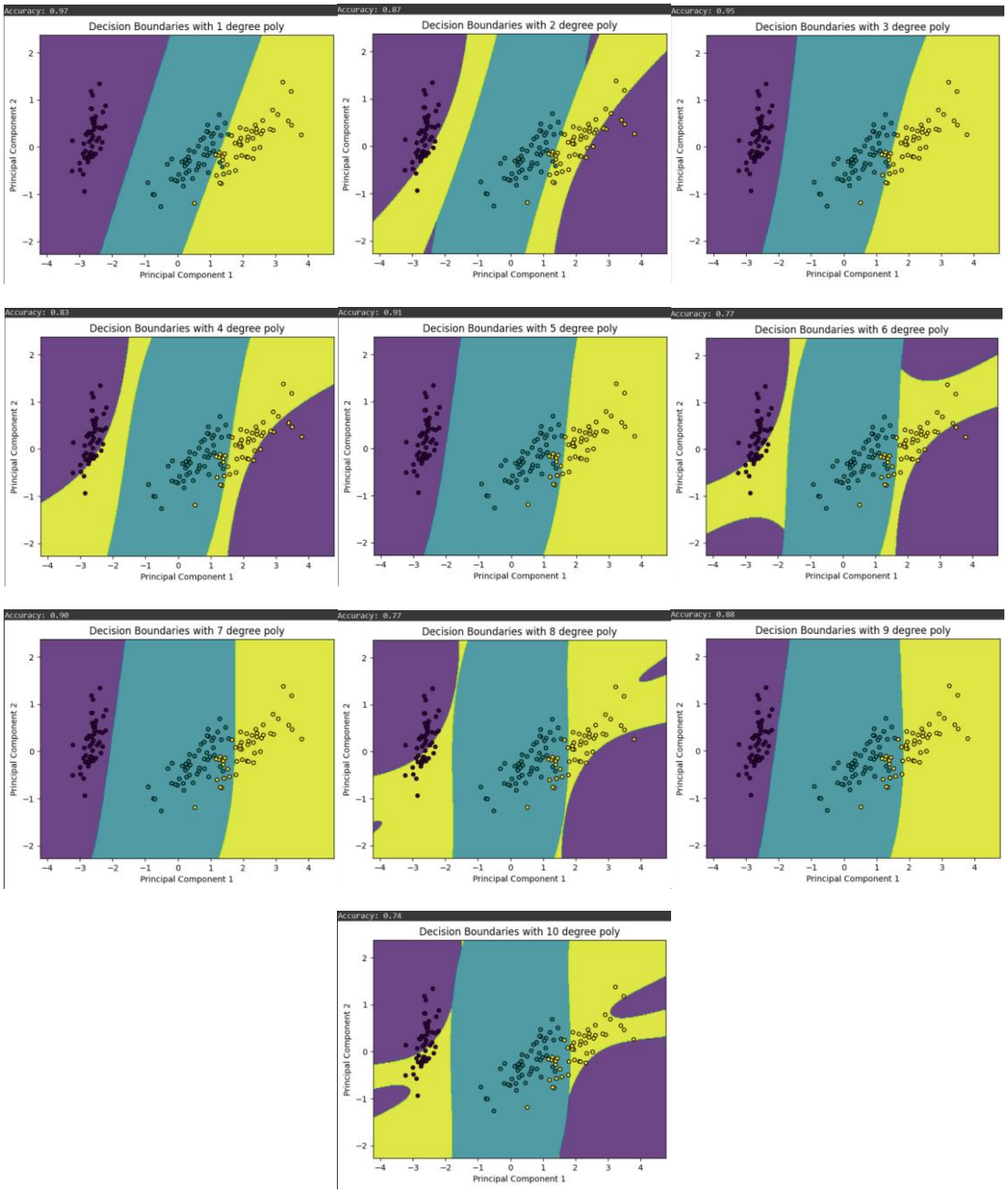
مطابق تصاویر بالا نتیجهٔ طبقه‌بندی با استفاده از انتخاب زوج ویژگی های 1و2 - 1و3 - 1و4 - 2و3 - 2و4 - 3و4 به ترتیب برابر با 0.82 - 0.95 - 0.96 - 0.96 - 0.96 - 0.97 است که بیشترن آن مربوط به انتخاب ویژگی‌های 3و4 است که پیش از این در قسمت اول سوال نیز مطرح کرده بودیم این دو ویژگی به دلیل اینکه توزیع جداپذیرتری برای سه کلاس دارند، می‌توانند بسیار مفید واقع شوند. البته این روش انتخاب ویژگی با اینکه در اینجا مقداری برابر با روش PCA نتیجه داد اما روشی با هزینهٔ محاسباتی بالاست چون اگر تعداد ویژگی‌ها بیشتر شود، باید تعداد دفعات بیشتری برای تعداد زوج ویژگی‌های انتخابی بیشتری آموزش انجام داده شود تا نتیجه احتمالا مناسب به دست آید. اما روش PCA با استفاده از اطلاعات همهٔ ویژگی‌ها بهترین زوج ویژگی را در اینجا در یک مرتبه به ما می‌دهد.

ج)

حال مانند قسمت قبل با استفاده از الگوریتم SVM اما با کرنل‌های غیرخطی و چندجمله‌ای از درجه 1 تا 10، دیتاست را آموزش می‌دهیم. این کار با استفاده از قطعه‌کد زیر صورت گرفته است. در اینجا تابعی نوشته شده است که مطابق اعداد 1 تا 10، درجه کرنل چندجمله‌ای تعیین شده، آموزش انجام داده می‌شود و محاسبهٔ معیارها و رسم مرزهای تصمیم گیری صورت می‌گیرد.

```
def plot_decision_boundaries_subplots(X, y):
    degrees = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    models = [SVC(kernel='poly', degree=degree, C=1.0) for degree in degrees]

    for model, degree in zip(models, degrees):
      model.fit(X_pca, y)
      y_pred = model.predict(X_pca)
      accuracy = accuracy_score(y, y_pred)

      plot_decision_boundary(X_pca, y, model, f'Decision Boundaries with {degree} degree poly')
      print("============================================================")
      print()


# Generate synthetic data
plot_decision_boundaries_subplots(X, y)
```

این 10 پلات به همراه مرز تصمیم‌گیری و معیارهای محاسبه‌شده به صورت زیر قابل مشاهده‌اند.



می‌توان از این خروجی‌ها به ازای چندجمله‌ای های از مرتبه 1 تا 10 اینگونه نتیجه‌گیری کرد که همواره افزودن پیچیدگی به کرنل SVM موجب بهبود نتیجه نمی‌شود و نوع دیتاست و ویژگی‌های آن بسیار دارای اهمیت است در این که از چه نوع کرنلی استفاده کنیم. در این دیتاست کرنل خطی نتیجهٔ بسیار مناسب‌تری داده است.

همچنین gif تولید شده با استفاده از کتابخانهٔ imageio از طریق این لینک قابل دسترسی است.

د.

در این قسمت از کد از کد مورد بررسی قرار گرفته در ویدیوهای حل تمرین استفاده شده است.

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

ابتدا تابع مورد نیاز برای تولید کرنل چندجمله‌ای نوشته شده است. به این صورت که با تعیین کردن مقدار توان و بایاس مورد نیاز، کرنل درست می‌شود.

این کد دو تابع SVM1 و multiclass_svm را پیاده‌سازی می‌کند که برای آموزش و استفاده از مدل‌های ماشین بردار پشتیبان (SVM) از ابتدا تا انتها استفاده می‌شود.

تابع SVM1

این تابع برای آموزش یک مدل SVM برای دسته‌بندی دودویی استفاده می‌شود. ورودی‌های این تابع عبارتند از:

این تابع ابتدا ماتریس Gram را برای محاسبه‌ی هسته ایجاد می‌کند، سپس با استفاده از کتابخانه cvxopt، مسئله بهینه‌سازی QP را حل می‌کند تا چندجمله‌ای‌ها لاگرانژی گرفته شده را به دست آورد. سپس بردار وزن (در صورت استفاده از هسته خطی) و بایاس محاسبه می‌شود. در نهایت، با استفاده از مدل آموزش دیده، پیش‌بینی‌ها برای داده‌های آزمون انجام می‌شود.

```
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
```

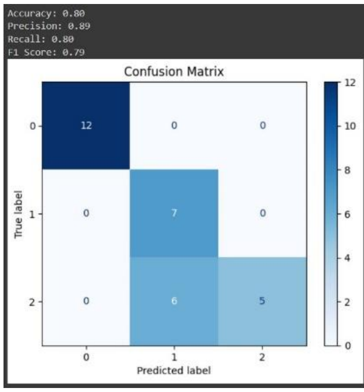
```
    sv_x = X[sv]
    sv_y = y[sv]
    numbers_of_sv=len(sv_y)
    # Bias (For linear it is the intercept):
    bias = 0
    for n in range(len(a)):
        # For all support vectors:
        bias += sv_y[n]
        bias -= np.sum(a * sv_y * K[ind[n], sv])
    bias = bias / len(a)

    # 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

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

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

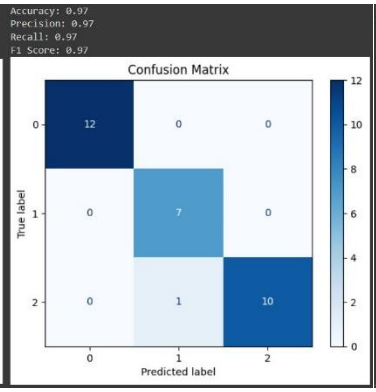
تابع multiclass_svm

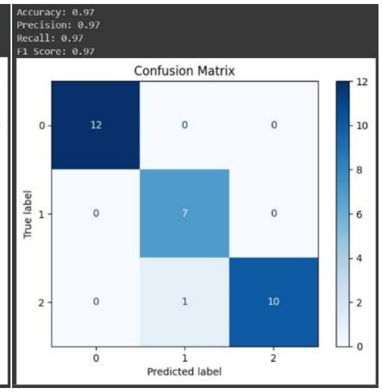
این تابع برای آموزش یک مدل SVM چندکلاسه استفاده می‌شود که با استفاده از توابع SVM دودویی (SVM1)، برای هر کلاس در مجموعه داده، یک مدل SVM دودویی آموزش داده و پیش‌بینی می‌کند.

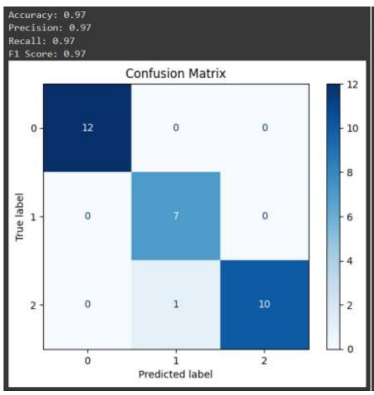
این تابع ابتدا مجموعه برچسب‌های یکتا را تعیین می‌کند، سپس برای هر کلاس در مجموعه داده، یک مدل SVM دودویی آموزش می‌دهد و پیش‌بینی می‌کند. نتایج پیش‌بینی برای هر کلاس در یک دیکشنری ذخیره می‌شود و به عنوان خروجی تابع ارائه می‌شود.

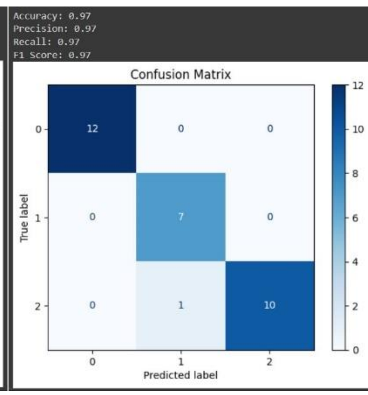
همچنین دقت الگوریتم به ازای افزایش درجه به صورت زیر است. همانطور که قابل مشاهده است به ازای افزودن پیچیدگی به کرنل‌ها در برخی از کرنل‌ها افزایش دقت در ماتریس درهم‌ریختگی قابل مشاهده است اما نمی‌توان به طور حتم گفت که افزایش پیچیدگی حتما باعث افزایش دقت می‌شود.

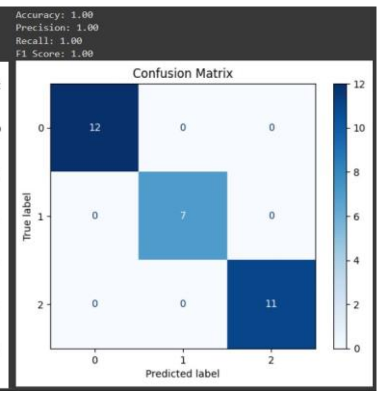

Accuracy: 0.80
Precision: 0.89
Recall: 0.80
F1 Score: 0.79
Confusion Matrix
True label
Predicted label

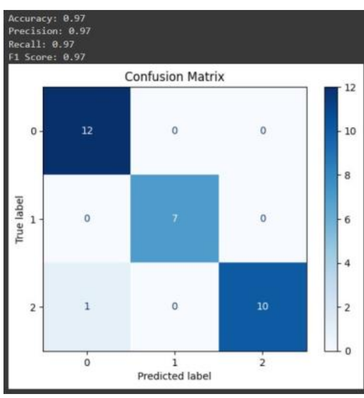

Accuracy: 0.97
Precision: 0.97
Recall: 0.97
F1 Score: 0.97
Confusion Matrix
True label
Predicted label

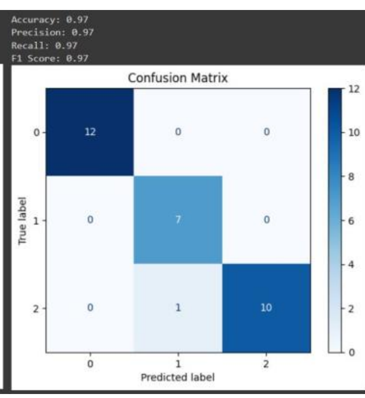

Accuracy: 0.97
Precision: 0.97
Recall: 0.97
F1 Score: 0.97
Confusion Matrix
True label
Predicted label

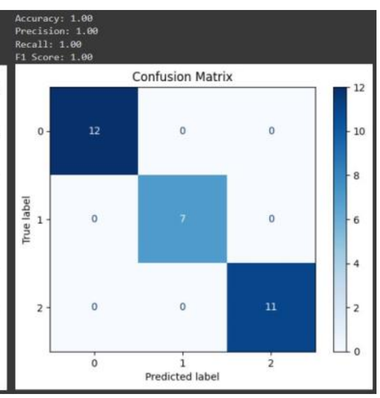

Accuracy: 0.97
Precision: 0.97
Recall: 0.97
F1 Score: 0.97
Confusion Matrix
True label
Predicted label

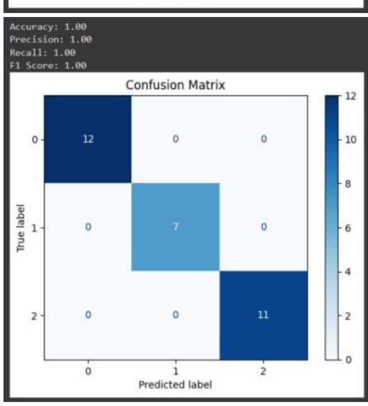

Accuracy: 0.97
Precision: 0.97
Recall: 0.97
F1 Score: 0.97
Confusion Matrix
True label
Predicted label


Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
Confusion Matrix
True label
Predicted label


Accuracy: 0.97
Precision: 0.97
Recall: 0.97
F1 Score: 0.97
Confusion Matrix
True label
Predicted label


Accuracy: 0.97
Precision: 0.97
Recall: 0.97
F1 Score: 0.97
Confusion Matrix
True label
Predicted label


Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
Confusion Matrix
True label
Predicted label


Accuracy: 1.00
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
Confusion Matrix
True label
Predicted label

```python
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 i,class_label in enumerate(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
    '''
    a=np.hstack((classifiers[0],classifiers[1],classifiers[2]))
    np.save('array_file', a)
    '''



    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)
```

هممچنین گیف تولید شده برای مرزهای تصمیم‌گیری برای حالت کدنویسی از پایه از طریق این لینک و همچنین gif معیارهای هر کدام از پیاده‌سازی‌ها با کرنل چندجمله‌ای از طریق این لینک قابل دسترسی است.

سوال سوم

آ.

توسعهٔ مدل‌های کشف و تشخیص تقلب همواره چالش‌هایی را به همراه داشته است. برخی از این چالش‌ها کلی هستند. یعنی ممکن است در تسک‌هایی که الزاما تشخیص تقلب نیستند نیز مشاهده شوند اما برخی دیگر از این چالش‌ها مختص این تسک هستند که در ادامه معرفی می‌شوند.

عدم تعادل کلاس‌ها: به طور کلی تراکنش‌ها و فعالیت‌های غیرقانونی و تقلبی بسیار کمتر از تراکنش‌های قانونی است و معمولا چیزی در حدود 0.01 کل معاملات را تشکیل می‌دهد. این عدم تعادل می‌تواند باعث شود مدلی که قصد آموزش آن را داریم به طبقهٔ دارای سمپل بیشتر بایاس پیدا کند و در تشخیص تقلب ضعیف عمل کند. همچنین در حضور عدم تعادل کلاس، استفاده از معیارهای معمول طبقه‌بندی مانند accuracy دیگر مناسب نیست. چون این معیار تعداد تشخیص‌های صحیح را به کل داده‌ها می‌سنجد و از آن‌جایی که کلاس مربوط به تقلب، تعداد نمونهٔ کمتری دارند نسبت به کلاس نرمال، حتی اگر همهٔ داداهای مربوط به کلاس تقلب به اشتباه به کلاس نرمال تعلق پیدا کند، افت چشمگیری در این معیار مشاهده نمی‌شود و این خوب نیست. می‌توان به جای این معیار از معیارهایی مثل F1-score استفاده کرد.

معمولا برای رفع مشکل عدم تعادل کلاس‌ها، از تکنیک‌هایی مانند oversampling, undersampling و یا تولید داده‌های مصنوعی برای متعادل کردن کلاس‌ها استفاده می‌شود. همچنین می‌توان به این صورت عمل کرد که اگر مدل در تشخیص کلاس دارای سمپل کمتر (اینجا کلاس تقلب) اشتباه کرد، با جریمهٔ بیشتری همراه شود.

مهندسی ویژگی: الگوهای تشخیص تقلب می‌توانند پیچیده باشند و برای تشخیص و تمایز بین کلاس تقلب و کلاس نرمال، به ویژگی‌های اصلی و روشنگرایانه‌ای نیاز داریم. از این بابت انتخاب ویژگی بسیار دارای اهمیت است. اما شناسایی ویژگی‌هایی که می‌تواند در این تشخیص کمک کننده باشد کار آسانی نیست. به خصوص در داده‌های با ابعاد بالا و ویژگی‌های فراوان.

امنیت و حریم خصوصی: معمولا داده‌های مربوط به تراکنش‌های مالی و کارت‌های اعتباری حساس هستند و مقررات و قوانین مالی و امنیتی مشمول این داده‌ها می‌شود. از این جهت دسترسی به این گونه داده‌های کار سختی است. برای ایجاد امکان استفاده از این داده‌ها کارهایی صورت می‌گیرد؛ تضمین حریم خصوصی داده‌ها و رعایت قوانین، ناشناس کردن افراد تراکنش-کننده، رمزگذاری داده‌ها و ...

حرفه‌ای شدن تکنیک‌های تقلب: کلاهبرداران به طور مداوم تکنیک‌های جدیدی را برای دور زدن سیستم‌های امنیتی و شناسایی تقلب استفاده می‌کنند. با این شرایط مدل‌هایی که بر روی داده‌های قبلی آموزش داده شده‌اند ممکن است با ظهور این الگوهای

جدید تقلب، قدیمی شوند و عملکرد مناسب قبلی را نداشته باشند. به روزرسانی مداوم این مدل‌ها بصورت آنلاین یا هر چند وقت یکبار، می‌تواند تا حدود مناسبی این مشکل را برطرف کند.

تشخیص به صورت زمان-واقعی: تشخیص تقلب از نظر زمانی دارای اهمیت زیادی است. ایجاد تاخیر زیاد در این تشخیص باعث تاخیر در پردازش‌ها می‌شود و این مناسب نیست. همچنین سرعت به تنهایی ملاک کافی نیست. نیاز است که بین سرعت و دقت تشخیص تعادل ایجاد شود. برای رفع این مشکل از الگوریتم‌های کارآمد و سریع باید استفاده کرد.

ارزیابی و اعتبارسنجی: ارزیابی مدل‌های تشخیص تقلب به دلیل نیاز به اعتبارسنجی قوی چالش‌برانگیز است. انتخاب معیارهای مناسب که عملکرد درستی از مدل بدهد و همچنین جلوگیری از برازش بیش از حد و اطمنیان از تعمیم مدل از موارد مهم هستند.

تفسیرپذیری: معمولا موسسه‌های مالی به مدل‌هایی نیاز دارند که علاوه برا دقت دارای شفافیت در اجرای عملیات تشخیص باشد. اما تفسیر مدل‌های مبتنی بر یادگیری عمیق به عنوان مثال می‌تواند مشکل باشد. برای رفع این مشکل می‌توان از روش‌هایی مثل درخت تصمیم که تفسیرپذیری دارد استفاده کرد.

اما این مقاله برای حل چالش‌های این‌چنینی برای تشخیص تقلب کارهایی انجام داده به ویژه با تمرکز بر مسائل عدم تعادل کلاس‌ها.

Oversampling با استفاده از SMOTE: برای رسیدگی به عدم تعادل در مجموعه داده، از تکنیک نمونه برداری بیش از حد اقلیت مصنوعی (SMOTE) استفاده شده است. این تکنیک نمونه‌های مصنوعی را برای کلاس اقلیت تولید می‌کند، بنابراین مجموعه داده‌ها را متعادل می‌کند و سوگیری نسبت به کلاس اکثریت را کاهش می‌دهد.

Denoising Autoencoder (DAE) : این مقاله یک رمزگذار خودکار حذف نویز برای مقابله با نویز در داده ها معرفی می کند. رمزگذار خودکار برای حذف نویز از داده های ورودی، بهبود کیفیت و استحکام ویژگی های مورد استفاده برای طبقه بندی آموزش دیده است.

استفاده از شبکه عصبی عمیق تماما متصل: برای طبقه بندی نهایی از شبکه عصبی عمیق استفاده می شود. این مدل از داده های تمیز و متعادل ارائه شده توسط مراحل قبلی برای افزایش دقت تشخیص تراکنش های تقلبی استفاده می کند.

ب.

ساختار این مقاله از دو قسمت تشکیل شده است.

Denoising Autoencoder (DAE) : مطابق تصویر زیر (جدول 2) که از مقاله آورده شده است، این قسمت شامل یک اتوانکودر با 7 لایه است.

لایه اول: لایهٔ ورودی است با 29 نورون (ویژگی های اصلی با نویز اضافه شده)

لایه‌های دوم تا ششم: لایه‌های پنهان هستند که به ترتیب دارای 22، 15، 10، 15 و 22 نورون هستند.

لایهٔ آخر: لایهٔ خروجی با 29 نورون (ویژگی های بازسازی شده)

Classifier (Deep Fully Connected Neural Network) : این قسمت نیز مطابق تصویر (جدول 3) شامل 6 لایه است.

لایه اول: لایهٔ ورودی است با 29 نورون (ویژگی های حذف شده)

لایه‌های دوم تا پنجم: لایه‌های پنهان هستند که به ترتیب دارای 22، 15، 10 و 5 نورون هستند.

لایهٔ ششم: لایهٔ خروجی با 2 نورون (احتمالات کلاس تقلب و غیر تقلب)

تابع فعال‌ساز: تابع سافت‌مکس روی لایهٔ خروجی

(تابع خطا : Cross-Entropy)

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

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.

در این بخش خواسته شده است که که مدل ارائه شده در مقاله پیاده‌سازی شود. پس از فراخوانی کتابخانه‌های مورد نیاز و لود کردن دیتاست از طریق gdown، نیاز است که ویژگی‌ها و لیبل‌ها به طور مناسب و مطابق گفته مقاله جداسازی شوند. بنابراین به جز دو ستون time و class مابقی ستون‌ها به عنوان ویژگی در نظر گرفته شده‌اند.

```
# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

file_path = "/content/creditcard.csv"
data = pd.read_csv(file_path)

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

سپس داده‌ها را به دو قسمت آموزش و تست با ضریب 0.2 تقسیم می‌کنیم.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)
```

سپس مطابق کاری که مقاله انجام داده است، ابتدا باید تعداد سمپل‌های دو کلاس را متعادل کنیم. همانطور که در قسمت قبلی توضیح داده شد، این مقاله با استفاده از روش SMOT کار متعادل کردن کلاس‌ها را انجام داده است. در این کد می‌توان استراتژی را انتخاب کرد که در تست اولیه و برای این قسمت از سوال برابر auto قرار می‌دهیم و با دستور fit بر روی داده‌ها اعمال می‌کنیم. سپس باید نویز به داده‌ها اضافه کنیم. نوع نویزی که انتخاب شده است و در مقاله نیز ذکر شده بود، نویز گوسی است. نویزفکتور برابر با 0.5 در نظر گرفته شده است.

```
# Apply SMOTE to balance the training dataset
smote = SMOTE(sampling_strategy='auto', random_state=4)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Add Gaussian noise to the training data
noise_factor = 0.5
X_noisy = X_resampled + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_resampled.shape)
```

سپس تمامی متغیرهای تعریف شده برای دیتاست‌ها را به صورت تنسور تغییر می‌دهیم. مقدار بچ‌سایز را برابر 256 قرار می‌دهیم و data_loader را می‌سازیم. این کار برای ایجاد کردن یک برنامه منظم برای ایجاد بچ‌های دیتا برای آموزش هر بچ انجام می‌گیرد. این کار را هم برای دیتاهای آموزش و هم برای دیتاهای تست انجام می‌دهیم.

```
# Convert to PyTorch tensors
X_noisy_tensor = torch.tensor(X_noisy, dtype=torch.float32).to(device)
X_resampled_tensor = torch.tensor(X_resampled, dtype=torch.float32).to(device)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32).to(device)
y_resampled_tensor = torch.tensor(y_resampled, dtype=torch.long).to(device)
y_test_tensor = torch.tensor(y_test, dtype=torch.long).to(device)

# Create DataLoader
batch_size = 256
train_dataset = TensorDataset(X_noisy_tensor, X_resampled_tensor)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

در قسمت بعدی مطابق ساختاری که در مقاله برای ایجاد بخش اتوانکودر آورده شده بود و در قسمت قبلی در مورد آن توضیحاتی ارائه شد، در اینجا اتوانکور را ایجاد کرده‌ایم که از دو بهش انکودر و دیکودر تشکیل شده است. هر کدام از این دو بخش، دارای لایه‌های تماما متصل‌اند که پس از هر لایه هم از تابع یک بخش سومی هم دارد به نام forward که برای ارجاع دیتاست ورودی و خروجی گرفتن است.

```
class DenoisingAutoencoder(nn.Module):
    def __init__(self):
        super(DenoisingAutoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(29, 22),
            nn.LeakyReLU(),
            nn.Linear(22, 15),
            nn.LeakyReLU(),
            nn.Linear(15, 10),
            nn.LeakyReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(10, 15),
            nn.LeakyReLU(),
            nn.Linear(15, 22),
            nn.LeakyReLU(),
            nn.Linear(22, 29),
            nn.LeakyReLU()
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

قدم بعدی، تعریف طبقه‌بند با استفاده از لایه‌های تمامی متصل است. 5 لایهٔ تماما متصل و پس از هر لایه از تابع relu استفاده شده است. در نهایت هم از تابع softmax استفاده شده است. سپس سه کلاس تعریف شده را باید برای انجام آموزش دادن، آماده‌سازی کنیم که در دو خط انتهایی این کد آورده شده است

```python
# Define Classifier
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.classifier = nn.Sequential(
            nn.Linear(29, 22),
            nn.LeakyReLU(),
            nn.Linear(22, 15),
            nn.LeakyReLU(),
            nn.Linear(15, 10),
            nn.LeakyReLU(),
            nn.Linear(10, 5),
            nn.LeakyReLU(),
            nn.Linear(5, 2),
            nn.Softmax(dim=1)
        )

    def forward(self, x):
        return self.classifier(x)

] # Instantiate models
autoencoder = DenoisingAutoencoder().to(device)
classifier = Classifier().to(device)
```

در گام بعدی با استفاده از تعریف تابع خطا به صورت MSELoss و همچنین بهینه‌ساز Adam، اقدام به آموزش اتوانکودر می‌کنیم. مقدار ایپاک مورد نظر را برابر 30 قرار می‌دهیم. در این آموزش سعی می‌شود که در انتهای آموزش، اتوانکودر بتواند داده نویزی ورودی به خود را بازسازی کند. خطا را در این بخش مطابق مقاله، تفاوت میان ورودی و خروجی بازسازی‌شده قرار می‌دهیم.

```python
# Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = optim.Adam(autoencoder.parameters(), lr=0.01)

autoencoder_loss = []


# Train Autoencoder
num_epochs1 = 30
for epoch in range(num_epochs1):
    autoencoder.train()
    for batch in train_loader:
        X_noisy_batch, X_resampled_batch = batch
        optimizer.zero_grad()
        outputs = autoencoder(X_noisy_batch)
        loss = criterion(outputs, X_resampled_batch)
        loss.backward()
        optimizer.step()
    print(f'Epoch [{epoch+1}/{num_epochs1}], Loss: {loss.item():.4f}')
```

```
Epoch [1/30], Loss: 0.4877
Epoch [2/30], Loss: 0.3680
Epoch [3/30], Loss: 0.3031
Epoch [4/30], Loss: 0.2892
Epoch [5/30], Loss: 0.2745
Epoch [6/30], Loss: 0.2749
Epoch [7/30], Loss: 0.2646
```

همانطور که از مقادیر خطای اتوانکور در اپیاک‌های متوالی قابل مشاهده است، خطا در حال کاهش است و هیچ‌گونه علامتی مبنی بر اورفیت در آن وجود ندارد.

در این قسمت در مرحلهٔ eval کلاس تعریف شده شده برای اتوانکودر، باید داده‌های مربوط به x های resample شده را به اتوانکودر آموزش یافته بدهیم که بتواند کار بازسازی داده‌ها را انجام دهد. سپس x های دینویز شده به صورت تنسور در آورده شده‌اند. در قدم بعدی برای آموزش داده در کلاس تعریف شده برای طبقه‌بند، لازم است که دیتالودر برای ایجاد بچ‌های مبتنی بر داده‌های دینویزشده را ایجاد کنیم.

```
# Extract denoised features using the trained autoencoder
autoencoder.eval()
with torch.no_grad():
    X_train_denoised = autoencoder(X_resampled_tensor).cpu().numpy()
    X_test_denoised = autoencoder(X_test_tensor).cpu().numpy()


# Convert denoised features to tensors
X_train_denoised_tensor = torch.tensor(X_train_denoised, dtype=torch.float32).to(device)
X_test_denoised_tensor = torch.tensor(X_test_denoised, dtype=torch.float32).to(device)

# Create DataLoader for denoised data
denoised_train_dataset = TensorDataset(X_train_denoised_tensor, y_resampled_tensor)
denoised_train_loader = DataLoader(denoised_train_dataset, batch_size=batch_size, shuffle=True)
```

تابع خطا مورد نظر برای کار طبقه‌بندی در این قسمت تابع cross entropy است که تابعی مناسب کار طبقه است. از بهینه ساز Adam در این قسمت نیز استفاده می‌کنیم. برای امکان ذخیره بهترین وزن‌ها، سلول دوم را می‌نویسیم که محل و نام ذخیره‌سازی وزن ایده‌آل است. همچنین برای امکان ذخیره‌سازی معیارها و وزن‌ها، لیست‌های خالی تعریف می‌کنیم.

```
# Define loss function and optimizer for the classifier
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(classifier.parameters(), lr=0.01)

# Track best loss for saving model
best_loss = float('inf')
best_model_path = 'best_classifier_model.pth'

# For plotting accuracy and recall
train_accuracies = []
train_recalls = []
test_accuracies = []
test_recalls = []
```

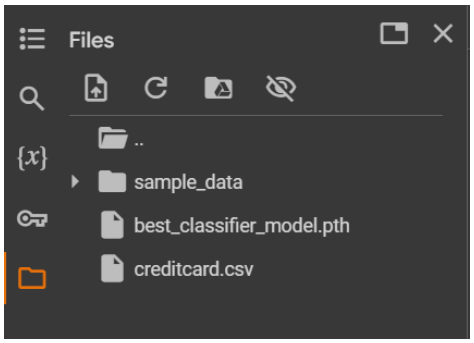
کد مورد نظر برای مرحلهٔ آموزش طبقه‌بند به صورت زیر است. مدل طبقه‌بندی (classifier) را با استفاده از داده‌های آموزشی denoised_train_loaderبرای 100 دوره آموزشی (epochs) آموزش می‌دهد و سپس آن را روی داده‌های تست ارزیابی می‌کند. بهترین مدل بر اساس کمترین میانگین خطا ذخیره می‌شود. در پایان هر دوره، مدل بر روی داده‌های تست ارزیابی

می‌شود و دقت و Recall تست محاسبه و نمایش داده می‌شود. در قسمت save the best model بر اساس خطای کمتر مطابق خواسته سوال، بهترین وزن ذخیره می‌شود.

```
num_epochs = 100

# Train Classifier
for epoch in range(num_epochs):
    classifier.train()
    running_loss = 0.0
    correct = 0
    total = 0
    y_true = []
    y_pred = []

    for batch in denoised_train_loader:
        X_denoised_batch, y_batch = batch
        optimizer.zero_grad()
        outputs = classifier(X_denoised_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total += y_batch.size(0)
        correct += (predicted == y_batch).sum().item()
        y_true.extend(y_batch.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())

    train_accuracy = 100 * correct / total
    train_recall = recall_score(y_true, y_pred)
    train_accuracies.append(train_accuracy)
    train_recalls.append(train_recall)
    avg_loss = running_loss / len(denoised_train_loader)

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}, Accuracy: {train_accuracy:.2f}%, Recall: {train_recall:.2f}')

    # Save the best model
    if avg_loss < best_loss:
        best_loss = avg_loss
        torch.save(classifier.state_dict(), best_model_path)
```

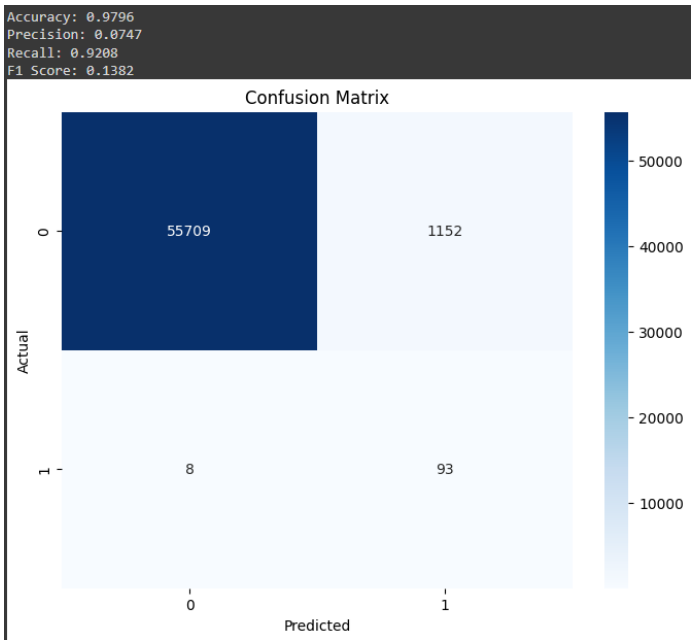
```
# Evaluate on test data
classifier.eval()
correct = 0
total = 0
y_true = []
y_pred = []

with torch.no_grad():
    for batch in test_loader:
        X_batch, y_batch = batch
        outputs = classifier(autoencoder.encoder(X_batch))
        _, predicted = torch.max(outputs.data, 1)
        total += y_batch.size(0)
        correct += (predicted == y_batch).sum().item()
        y_true.extend(y_batch.cpu().numpy())
        y_pred.extend(predicted.cpu().numpy())

test_accuracy = 100 * correct / total
test_recall = recall_score(y_true, y_pred)
test_accuracies.append(test_accuracy)
test_recalls.append(test_recall)
print(f'Test Accuracy: {test_accuracy:.2f}%, Test Recall: {test_recall:.2f}')
```

ذخیره‌سازی بهترین وزن‌های در طول دوره‌های آموزشی آموزش طبقه‌بند به این صورت در محیط colab انجام شده و قابل دسترسی است.



د.

پس از پایان آموزش نوبت به مرحلهٔ ارزیابی آن می‌رسد. برای ارزیابی از 4 معیار مختلف استفاده شده است و همچنین ماتریس در هم ریختگی برای داده‌های آموزش به صورت زیر قابل مشاهده است. کلاس ها به طور مناسبی پیش‌بینی طبقه شده‌اند. معیارهای accuracy و recall مقادیر مناسبی دارند اما معیار precision (به دلیل بالابودن نسبی عدد درایهٔ بالا راست ماتریس در هم ریختگی) مقدار مناسبی ندارد. اما باید توجه کرد که در کار تشخیص خطا و عیب، چون تشخیص درست داده‌های مربوط به کلاس تقلب و عیب از اهمیت بیشتری از تشخیص کلاس نرمال برخوردار است از این جهت، مهم‌تر است که معیار recall در اینجا بیشتر باشد. چون به طور کلی در تشخیص نرمال و تقلب، خیلی مشکل حادی پیش نمی‌آید اگر به اشتباه دادهٔ کلاس نرمال را تقلب در نظر بگیریم اما اگر داده‌های تقلب به طورت نرمال تشخیص داده شود، کل کار طبقه‌بندی زیر سوال می‌رود. بنابراین معیارهای و ماتریس در هم ریختگی قابل قبول هستند.

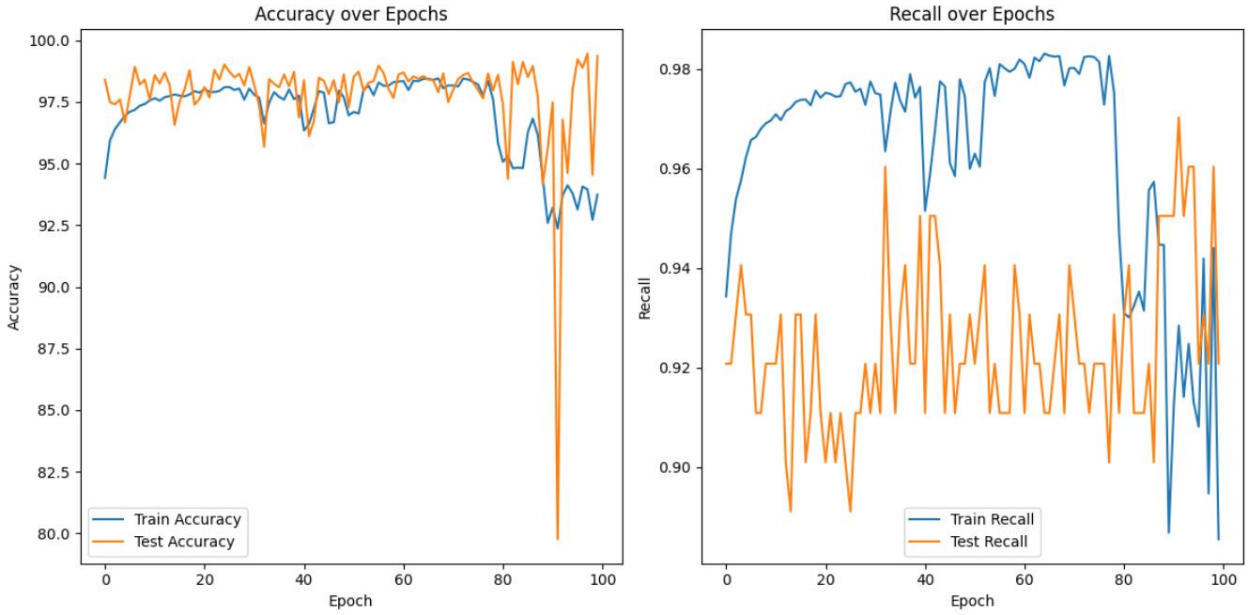


باید در نظر داشت که استفاده از معیار accuaracy در کار طبقه‌بندی دیتاهایی که با عدم تعادل کلاس‌ها همراه است، کار درستی نیست. همانطور که در بخش‌های قبلی توضیح داده شد، چون این معیار تعداد تشخیص‌های صحیح را به کل داده‌ها می‌سنجد و از آن‌جایی که کلاس مربوط به تقلب، تعداد نمونهٔ کمتری دارند نسبت به کلاس نرمال، حتی اگر همهٔ داده‌های مربوط به کلاس تقلب به اشتباه به کلاس نرمال تعلق پیدا کند، افت چشمگیری در این معیار مشاهده نمی‌شود و این خوب نیست. می‌توان به جای این معیار از معیارهایی مثل recall و F1_score استفاده کرد. معیار recall چون تمرکز روی داده‌های TP و FN دارد، می‌تواند عملکرد مناسبی را برای کارهای تشخیص عیب گزارش کند. چون اگر این معیار مناسب نباشد، یعنی داده‌های مربوط به عیب به اشتباه نرمال تشخیص داده می‌شوند. F1_score هم چون ترکیبی از recall و precision است، معیار کامل و مناسبی است.

معیارهای این قسمت از طریق classification report قابل مشاهده است که توضیحات قبلی برای این نتایج هم درست است.

```
# Calculate and print the classification report
report = classification_report(y_true, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     56861
           1       0.21      0.92      0.34       101

    accuracy                           0.99     56962
   macro avg       0.60      0.96      0.67     56962
weighted avg       1.00      0.99      1.00     56962
```



۵.

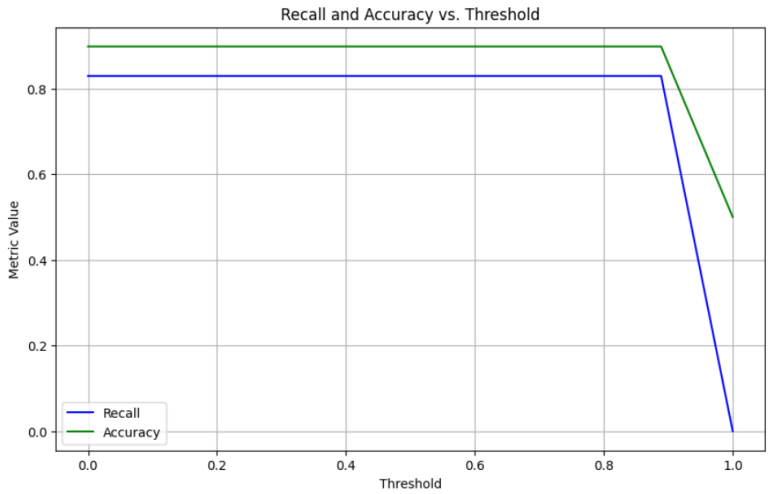
در این قسمت با توجه به فهمی که از مقاله به دست آوردم باید این کار انجام شود که یک آستانه برای مقادیر احتمالی پیش‌بینی‌شدهٔ طبقه‌بند در نظر گرفته شود. این مقدار آستانه باید از مقدار 0 تا 1 تغییر کند که بتواند یک نموداری مشابه نمودار خواسته شده در صورت سوال بشود. یک حدس دیگر که از فهم مسئله و مقاله می‌زنم این است که باید آستانه‌ای برای SMOT زده شود اما در آن صورت به نظر نمودار مشابه آن چه در مقاله است نخواهد شد چون در نمودار 6 مقاله که هیچ از SMOT استفاده نشده است باز هم این نمودار رسم شده است. به هر حال به روش اول عمل می‌کنیم.

```
def calculate_metrics(outputs , labels, thresholds):
    recalls = []
    accuracies = []
    for threshold in thresholds:
        predictions = (outputs > threshold).astype(int)
        recall = recall_score(labels, predictions)
        accuracy = accuracy_score(labels, predictions)
        recalls.append(recall)
        accuracies.append(accuracy)

    return recalls , accuracies


thresholds = np.linspace(0, 1, 10)

test_recalls, test_accuracies = calculate_metrics(y_pred, y_true, thresholds)
```

در این قسمت روی مقادیر تعریف شده از 0 تا 1 و به ازای پله‌های 0.1 روی خروجی احتمالی طبقه‌بند، آستانه در نظر گرفته شده است.

```
# Plotting the recall and accuracy for different thresholds on the same figure
plt.figure(figsize=(10, 6))

# Plot Recall
plt.plot(thresholds, test_recalls, label='Recall', color='blue')

# Plot Accuracy
plt.plot(thresholds, test_accuracies, label='Accuracy', color='green')

plt.xlabel('Threshold')
plt.ylabel('Metric Value')
plt.title('Recall and Accuracy vs. Threshold')
plt.legend()
plt.grid(True)

plt.show()
```

در نهایت به نمودار زیر می‌رسیم که درست نیست. و احتمالا قسمتی از کد اشتباهی صورت گرفته است اما روش کار و تعریف مسئله پس از چک مجدد درست محاسبه شده است. اما بلخره نمودار مناسبی رسم نشده است. در گزارش‌های تکمیلی در ادامهٔ ترم در صورت اجازهٔ استاد، نمودارهای درست قرار داده می‌شود.



و.

در این قسمت ابتدا مجددا کلاس تعریف شده برای مدل طبقه‌بند تماما متصل را تعریف می‌کنیم. همان x و y های مورد نظر را تعریف می‌کنیم.

```
# Define Classifier
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.classifier = nn.Sequential(
            nn.Linear(29, 22),
            nn.ReLU(),
            nn.Linear(22, 15),
            nn.ReLU(),
            nn.Linear(15, 10),
            nn.ReLU(),
            nn.Linear(10, 5),
            nn.ReLU(),
            nn.Linear(5, 2),
            nn.Softmax(dim=1)
        )

    def forward(self, x):
        return self.classifier(x)
```

```
# Instantiate models
classifier = Classifier().to(device)
```

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

```
scaler = StandardScaler()
X[:, -1] = scaler.fit_transform(X[:, -1].reshape(-1, 1)).flatten()
```

در قسمت بعد مجددا جداسازی داده‌های آموزش و تست را انجام می‌دهیم. در این قسمت بر خلاف قسمت قبل که باید داده-ها را دینویز کنیم، این کار را نکرده و مستقیما به تنسور تبدیل کرده و در دیتالودر میریزیم. تابع خطا و بهینه ساز نیز مطابق قبل.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)

# Convert denoised features to tensors
X_train_denoised_tensor = torch.tensor(X_train, dtype=torch.float32).to(device)
X_test_denoised_tensor = torch.tensor(X_test, dtype=torch.float32).to(device)

y_resampled_tensor = torch.tensor(y_train, dtype=torch.float32).to(device)

X_train_denoised_tensor.dtype
```

```
torch.float32
```

```
# Create DataLoader for denoised data
denoised_train_dataset = TensorDataset(X_train_denoised_tensor, y_resampled_tensor)
denoised_train_loader = DataLoader(denoised_train_dataset, batch_size=batch_size, shuffle=True)

# Define loss function and optimizer for the classifier
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(classifier.parameters(), lr=0.01)
```

در این مرحله که باید کار آموزش را انجام بدهیم، متاسفانه به دلیل اروری به صورت زیر و نبود وقت لازم، نشد که ارور را بر طرف کنم. این ارور مربوط به نوع داده‌ها در مرحلهٔ محاسبهٔ خطاست که به دلیل نبود وقت نشد که برطرف کنم. این قسمت را نیز در صورت اجارهٔ تیم حل تمرین در ادامه تکمیل و ارسال خواهم کرد.

```
num_epochs = 60

# Train Classifier
for epoch in range(num_epochs):
    classifier.train()
    running_loss = 0.0
    correct = 0
    total = 0
    y_true = []
    y_pred = []

    for batch in denoised_train_loader:
        X_denoised_batch, y_batch = batch
        optimizer.zero_grad()
        outputs = classifier(X_denoised_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-394-195d60a35d34> in <cell line: 4>()
     14         optimizer.zero_grad()
     15         outputs = classifier(X_denoised_batch)
---> 16         loss = criterion(outputs, y_batch)
     17         loss.backward()
     18         optimizer.step()

                                 3 frames
/usr/local/lib/python3.10/dist-packages/torch/nn/functional.py in cross_entropy(input, target, weight, size_average, ignore_index, reduce, reductio
   3084     if size_average is not None or reduce is not None:
   3085         reduction = _Reduction.legacy_get_string(size_average, reduce)
-> 3086     return torch._C._nn.cross_entropy_loss(input, target, weight, _Reduction.get_enum(reduction), ignore_index, label_smoothing)
   3087
   3088

RuntimeError: "nll_loss_forward_reduce_cuda_kernel_2d_index" not implemented for 'Float'
```