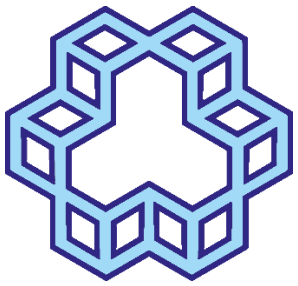


دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده مهندسی برق

درس یادگیری ماشین

استاد دکتر علیاری

سیدمحمدرضا حسینی

شماره دانشجویی: ۴۰۲۰۴۵۸۴

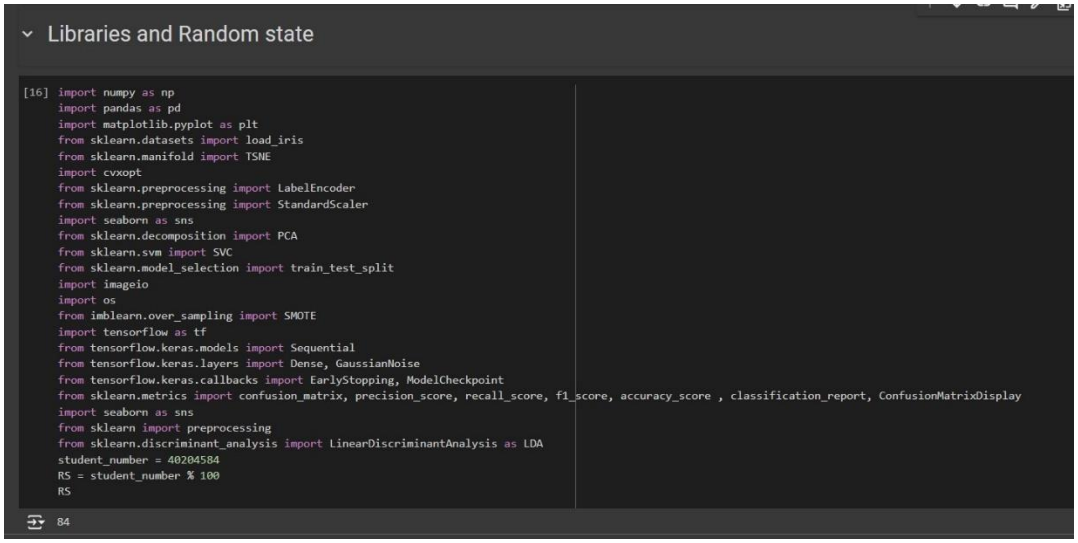
گرایش: سیستم های الکترونیک دیجیتال

مینی پروژه شماره ۳

Google Colab

Github

در ابتدای کد import های مورد نیاز را انجام می دهیم.



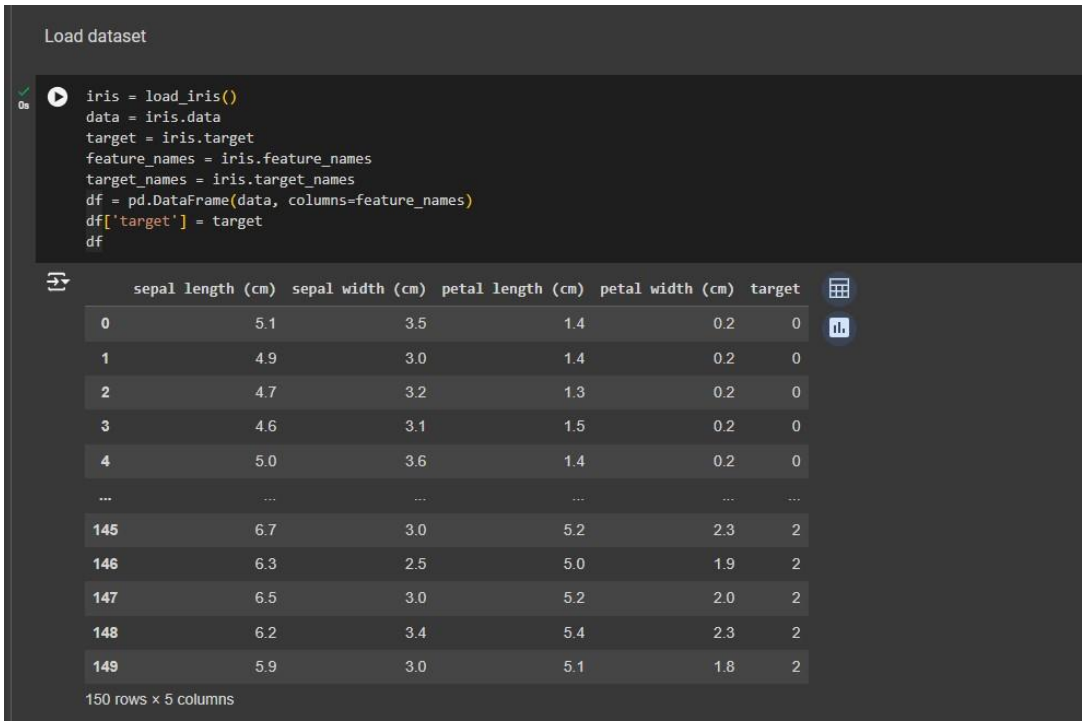
Libraries and Random state

```
[16] import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.datasets import load_iris
     from sklearn.manifold import TSNE
     import cvxopt
     from sklearn.preprocessing import LabelEncoder
     from sklearn.preprocessing import StandardScaler
     import seaborn as sns
     from sklearn.decomposition import PCA
     from sklearn.svm import SVC
     from sklearn.model_selection import train_test_split
     import imageio
     import os
     from imblearn.over_sampling import SMOTE
     import tensorflow as tf
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense, GaussianNoise
     from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
     from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score , classification_report, ConfusionMatrixDisplay
     import seaborn as sns
     from sklearn import preprocessing
     from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
     student_number = 40204584
     RS = student_number % 100
     RS

84
```

۱. سوال اول

۱/۱. آ

دیتاست مورد نظر را اضافه می کنیم.

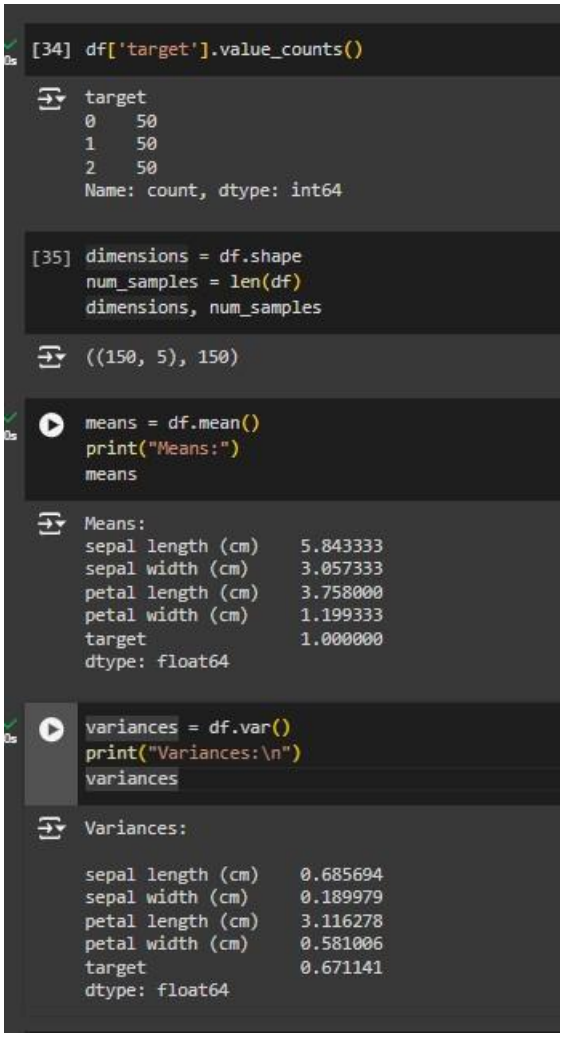


Load dataset

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

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

150 rows × 5 columns

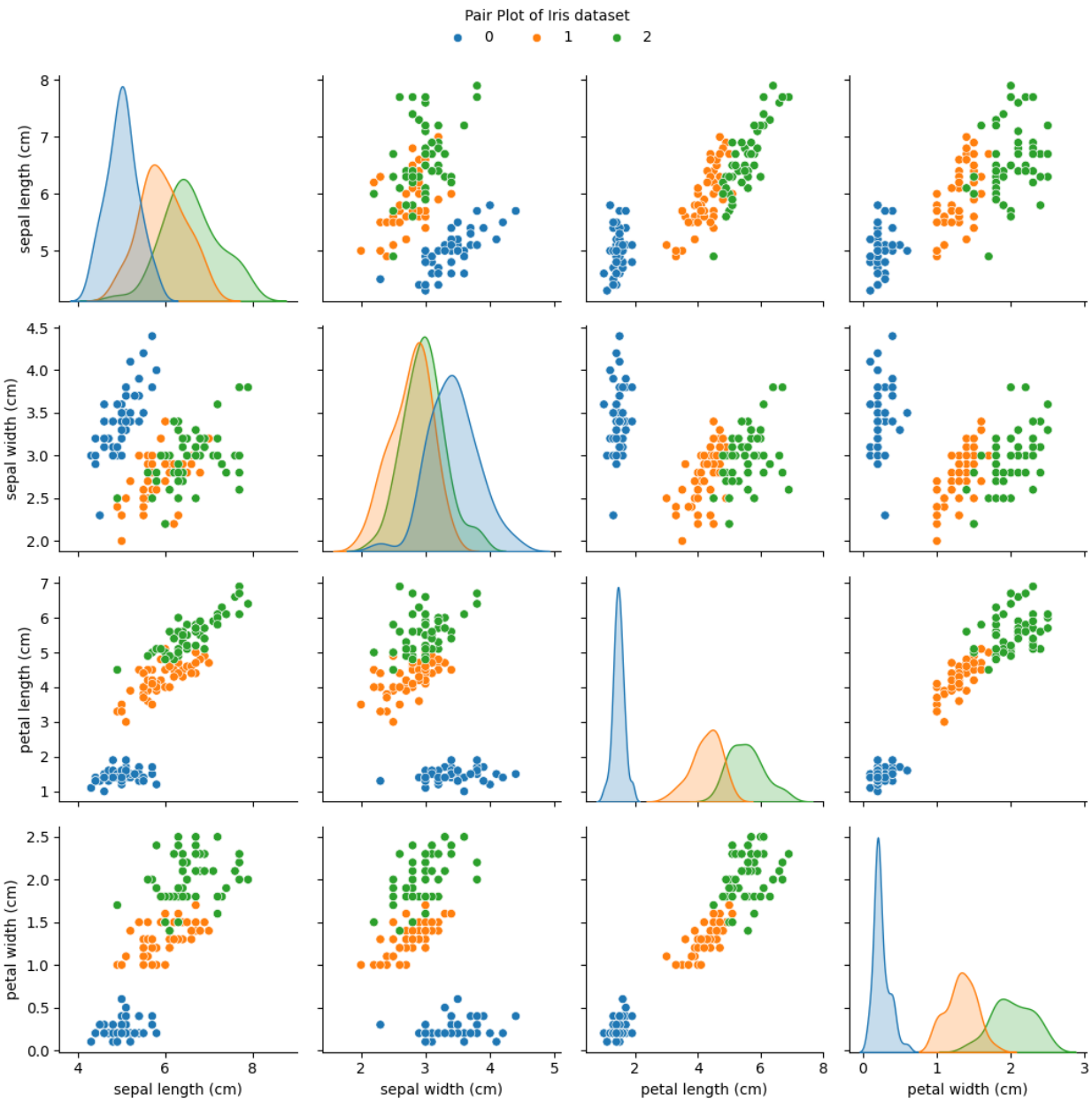


```
[34] df['target'].value_counts()

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

[35] dimensions = df.shape
     num_samples = len(df)
     dimensions, num_samples

((150, 5), 150)

     means = df.mean()
     print("Means:")
     means

Means:
sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
target               1.000000
dtype: float64

     variances = df.var()
     print("Variances:\n")
     variances

Variances:

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

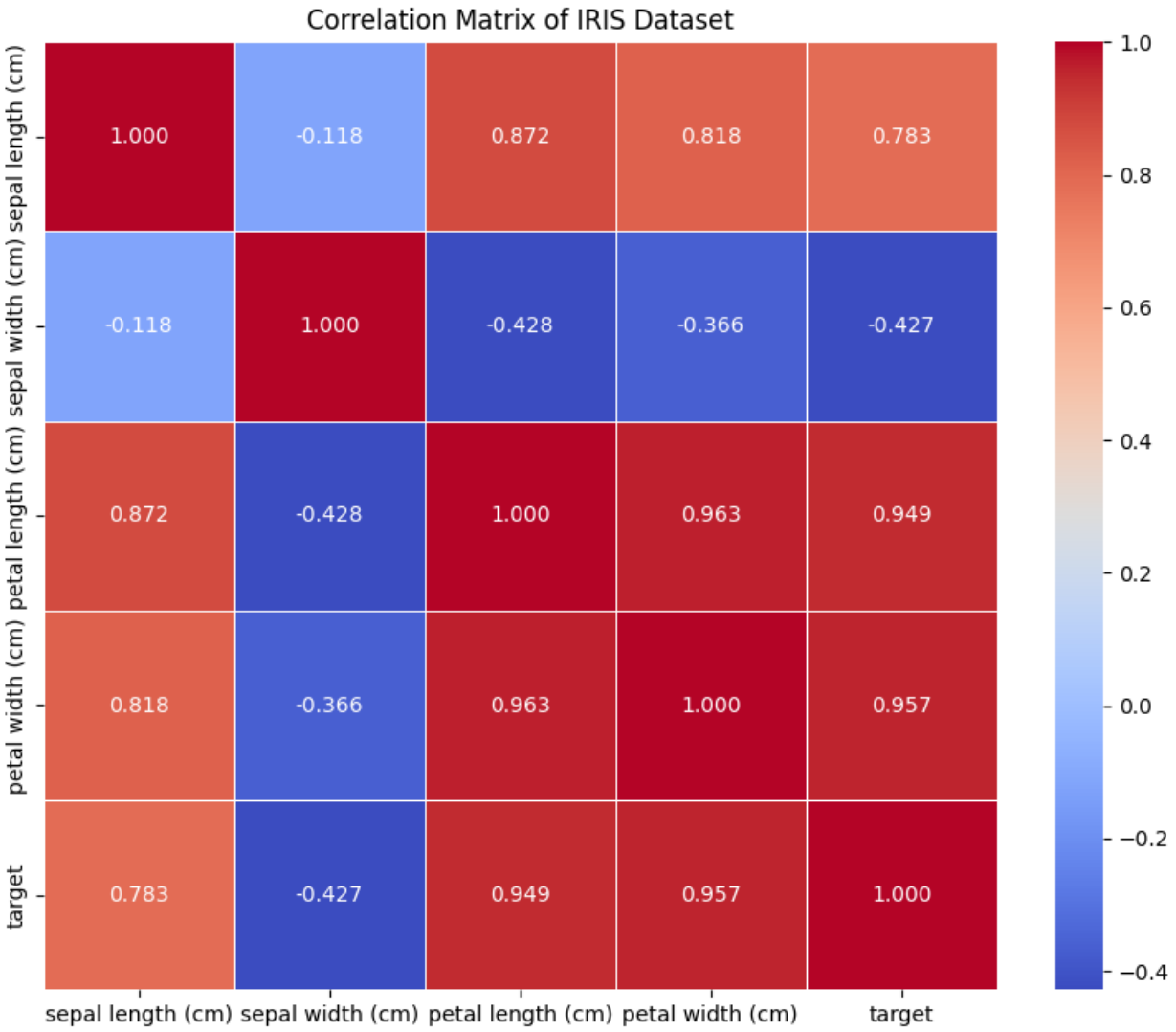
همانطور که مشاهده می شود سه کلاس داریم که از هر کدام ۵۰ دیتا (مجموعا ۱۵۰ تا) داریم و ۴ ویژگی داریم.

۴ ویژگی شامل طول و عرض sepal و petal هستند.

میانگین و واریانس هر یک از ویژگی ها را در کد بالا مشاهده می کنید.



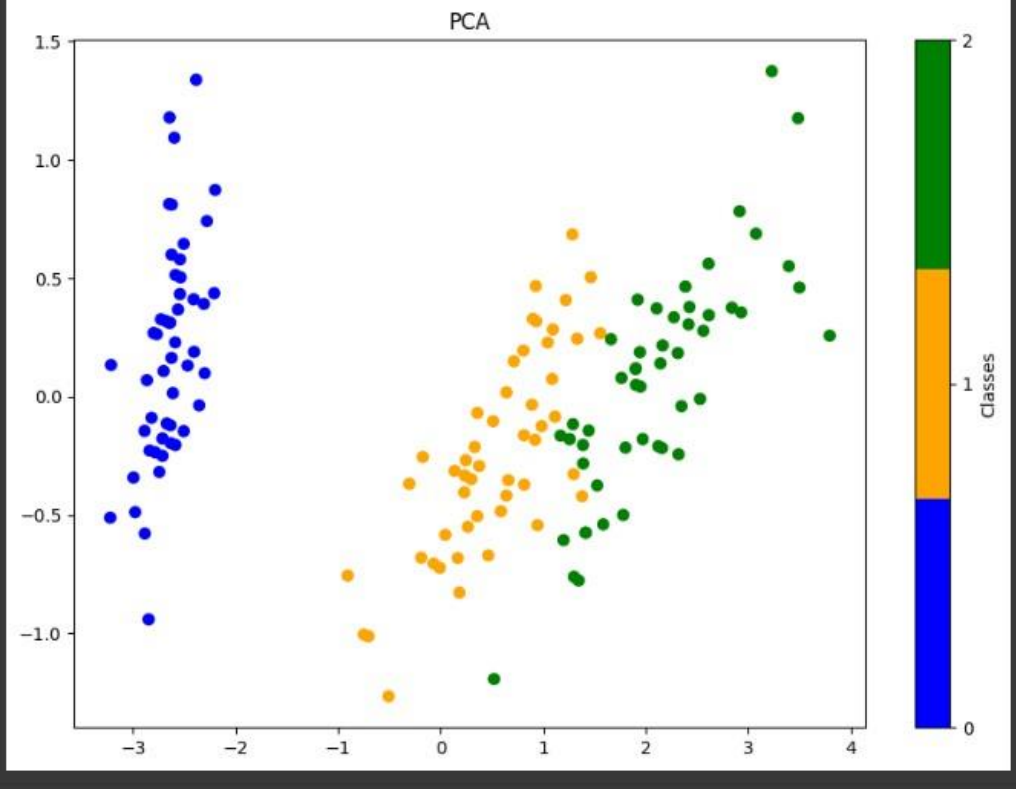
پراکندگی داده ها را در شکل بالا مشاهده می کنید. همانطور که مشاهده می شود، ویژگی های petal length و petal width بهترین جداسازی را در کلاس ها انجام می دهند در حالی که در ویژگی sepal width کلاس ها همپوشانی زیادی دارند.



ماتریس همبستگی ۴ ویژگی را مشاهده می کنید. از آنجا که به جز sepal width، سایر ویژگی ها با هم همبستگی بالایی دارند، بنظر می رسد که می توان از این ۳ ویژگی ۱ ویژگی بدست آورد و تعداد ویژگی ها را به کمک کاهش ابعاد از ۴ به ۲ رساند. (البته همبستگی sepal length با سایر ویژگی ها نسبتا کمتر است.)

بنابراین یک PCA با n_components=2 (تعداد ویژگی نهایی = ۲) می زنیم.

```
pca = PCA(n_components=2, random_state = RS)
pca_result = pca.fit_transform(data)
custom_colors = ['blue', 'orange', 'green']
custom_cmap = ListedColormap(custom_colors)
plt.figure(figsize=(10, 7))
scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1], c=target, cmap=custom_cmap, marker='o')
cbar = plt.colorbar(scatter, ticks=[0, 1, 2], label='Classes')
cbar.ax.set_yticklabels(['0', '1', '2'])
plt.title('PCA')
plt.show()
```



همانطور که مشاهده می شود، مطابق انتظار داده های کلاس ۰ به خوبی از سایرین جدا شده است اما داده های کلاس ۱ و ۲ کمی همپوشانی دارند.

۱/۲. ب

از LDA استفاده می‌کنیم زیرا این روش برای مسائل نظارت‌شده (supervised) مناسب است، در حالی که PCA برای مسائل بدون نظارت (unsupervised) استفاده می‌شود. LDA تلاش می‌کند فاصله بین کلاس‌ها را بیشتر و واریانس درون کلاس‌ها را کمتر کند تا کلاس‌ها بهتر جدا شوند. اما PCA به دنبال پیدا کردن محورهایی با بیشترین واریانس در داده‌ها است که ممکن است برای تمایز بین کلاس‌ها بهینه نباشد.

به بیان ساده، LDA برای تمایز بهتر بین کلاس‌ها در مسائل نظارت‌شده استفاده می‌شود، در حالی که PCA بیشتر برای کاهش ابعاد داده‌ها بدون توجه به برچسب‌های کلاس‌ها کاربرد دارد.

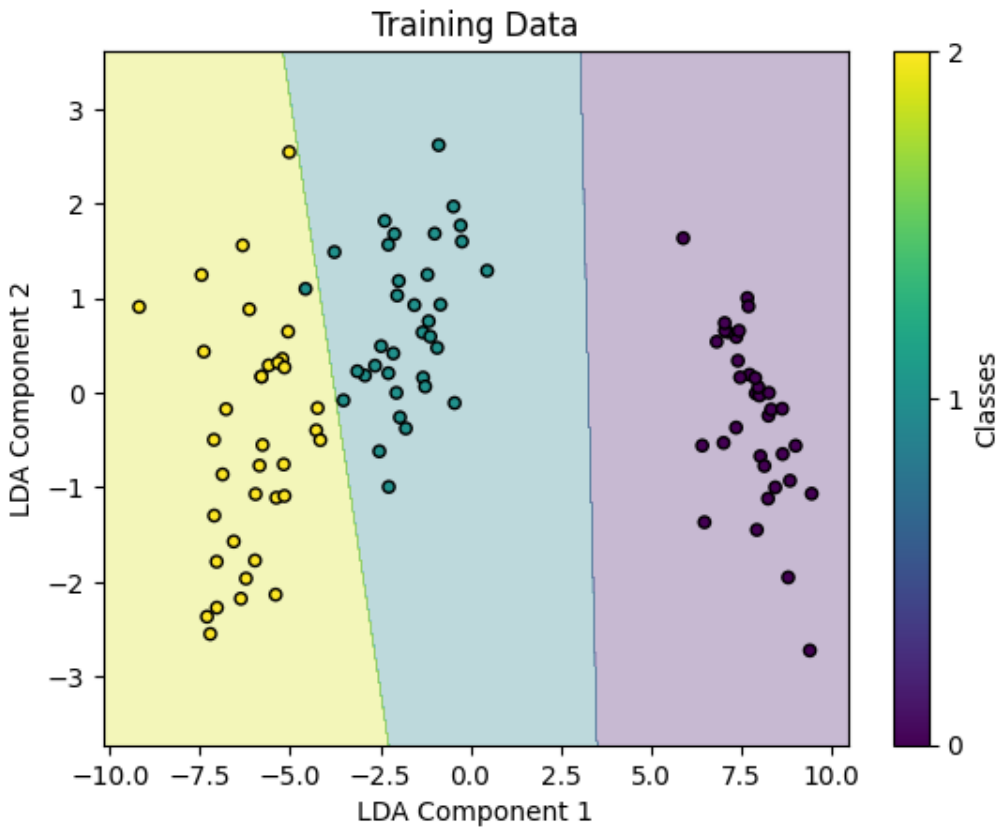
```python
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.3, random_state=RS, stratify=target)

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

lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)

svm = SVC(kernel='linear')
svm.fit(X_train_lda, y_train)
y_test_pred = svm.predict(X_test_lda)

def plot_decision_boundaries(X, y, model, title='Decision Boundaries'):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
    scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolor='k', s=20)
    plt.colorbar(scatter, ticks=range(len(target_names)), label='Classes')
    plt.title(title)
    plt.xlabel('LDA Component 1')
    plt.ylabel('LDA Component 2')
    plt.show()

plot_decision_boundaries(X_train_lda, y_train, svm, title='Decision Boundaries for SVM with Linear Kernel (Training Data)')

plot_decision_boundaries(X_test_lda, y_test, svm, title='Decision Boundaries for SVM with Linear Kernel (Test Data)')
```

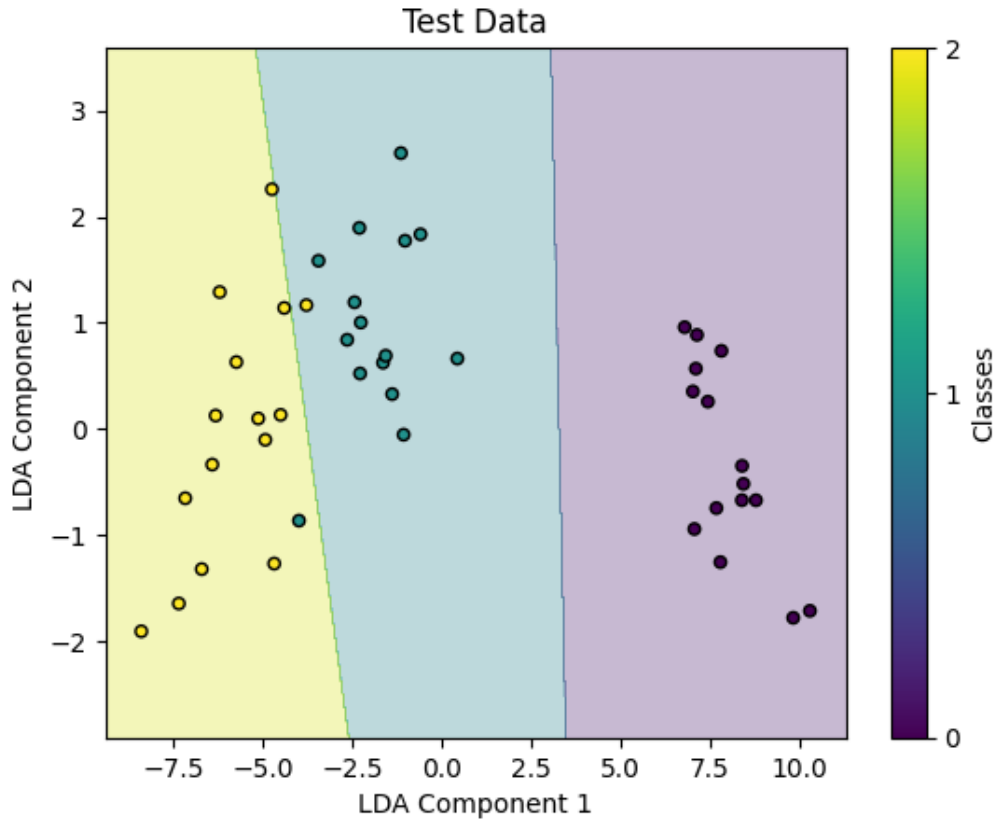
در کد بالا ابتدا دیتا را به دو بخش تست و آموزش تقسیم کردیم و اسکیل کردیم. سپس LDA زدیم. سپس یک smv خطی را آموزش دادیم. سپس نقاط و خطوط تصمیم گیری (به کمک تابع تعریف شده) را برای داده تست و آموزش رسم کردیم.

نتیجه را در تصاویر زیر می بینید:

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

مدل ما برای داده تست دقت ۹۶ درصدی دارد.


Training Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0


Test Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

۱/۳. ج

با توجه به قابل قبول بودن نتیجه قسمت قبل، از همان روش کاهش بعد استفاده می کنیم.

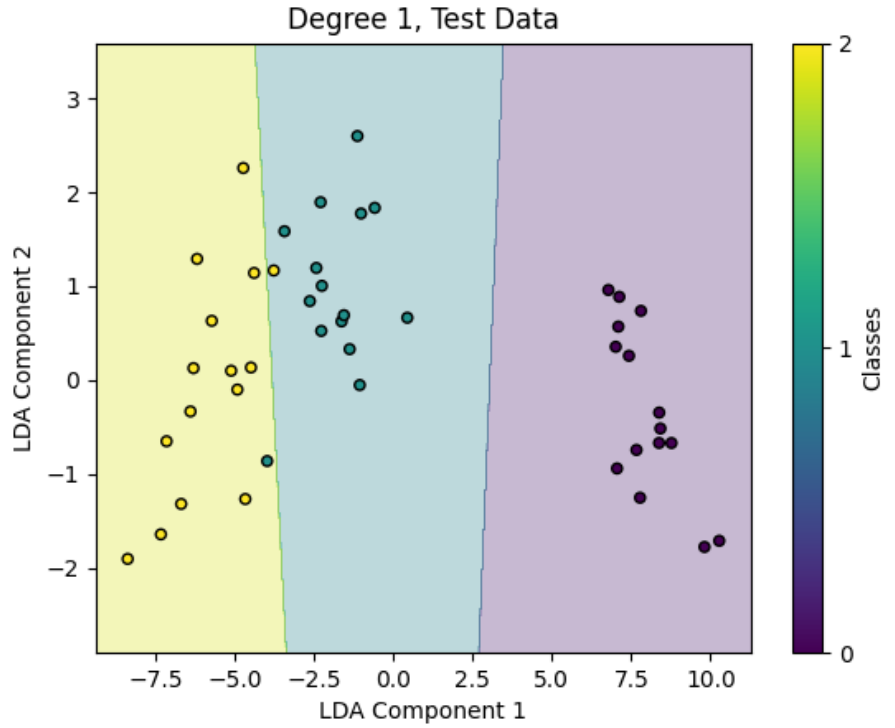
مسئله را با درجات ۱ تا ۱۰ پیاده سازی کرده و accuracy را گزارش می کنیم.

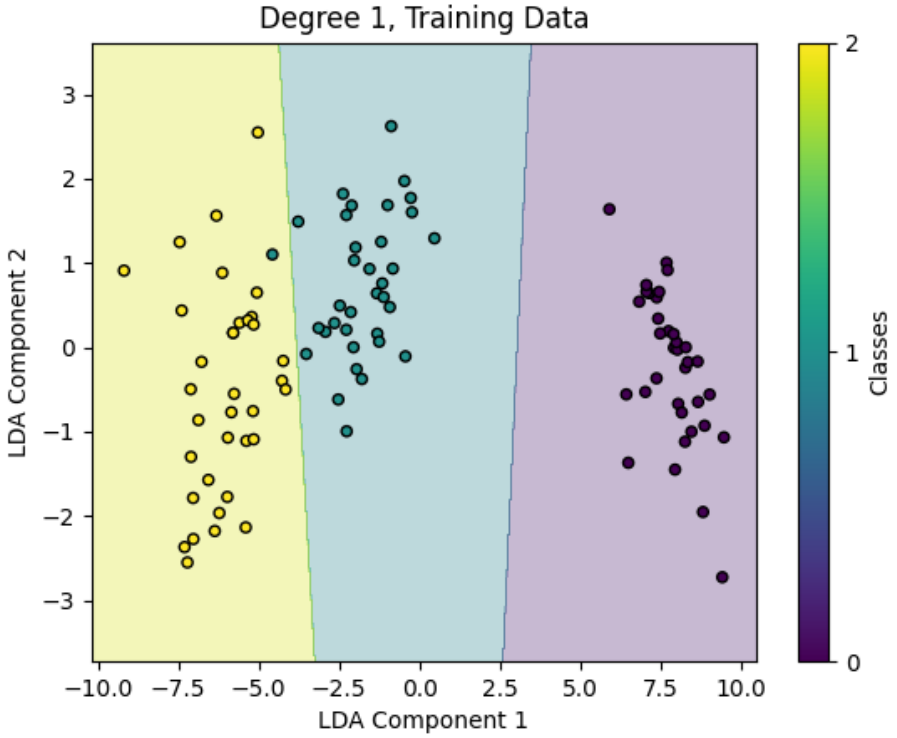
```
def plot_decision_boundaries_and_save(X, y, model, degree, dataset_type='Training'):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01),
                         np.arange(y_min, y_max, 0.01))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
    scatter = plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolor='k', s=20)
    plt.colorbar(scatter, ticks=range(len(target_names)), label='Classes')
    plt.title(f'Degree {degree}, {dataset_type} Data')
    plt.xlabel('LDA Component 1')
    plt.ylabel('LDA Component 2')
    filename = f'Degree_{degree}_{dataset_type}.png'
    plt.savefig(filename)
    plt.close()
    return filename

for degree in range(1, 11):
    svm_poly = SVC(kernel='poly', degree=degree)
    svm_poly.fit(X_train_lda, y_train)
    y_test_pred = svm_poly.predict(X_test_lda)

    accuracy = accuracy_score(y_test, y_test_pred)
    results.append((degree, accuracy))

    train_plot_file = plot_decision_boundaries_and_save(X_train_lda, y_train, svm_poly, degree, dataset_type='Training')
    images.append(imageio.imread(train_plot_file))

    test_plot_file = plot_decision_boundaries_and_save(X_test_lda, y_test, svm_poly, degree, dataset_type='Test')
    images.append(imageio.imread(test_plot_file))

gif_filename = 'decision_boundaries.gif'
imageio.mimsave(gif_filename, images, duration=2)

for degree, accuracy in results:
    print(f'Degree: {degree}, Accuracy: {accuracy:.2f}')
```

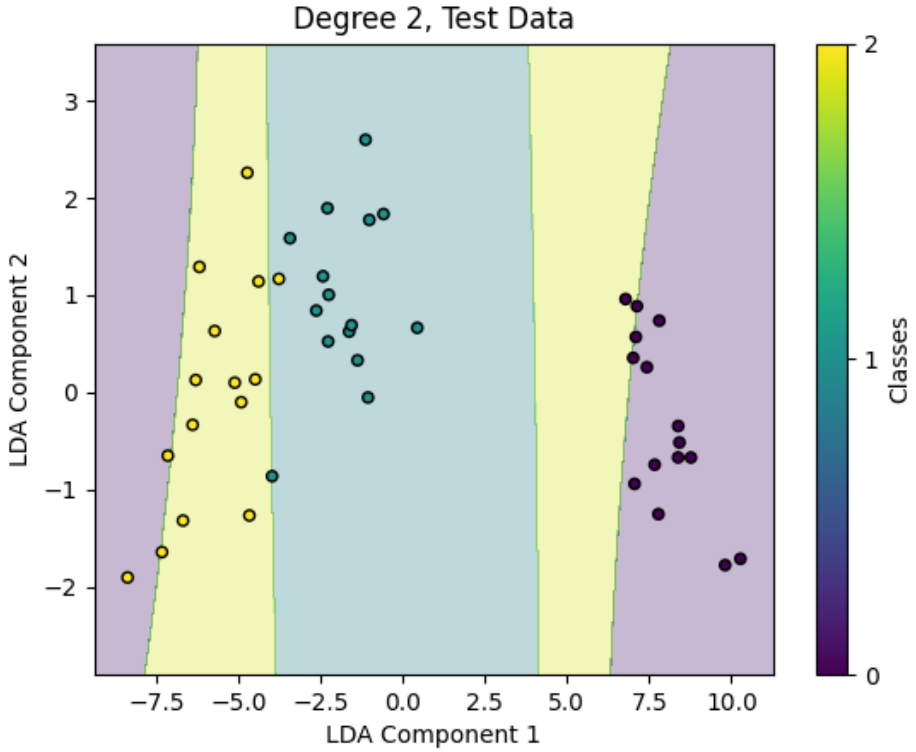
در کد بالا برای درجات ۱ تا ۱۰ یک svc را اموزش می دهیم و به کمک تابع نوشته شده یک تصویر از مرزهای تصمیم گیری ذخیره می کنیم. در نهایت به کمک این تصاویر یک gif می سازیم. سپس accuracy ها را نمایش می دهیم.

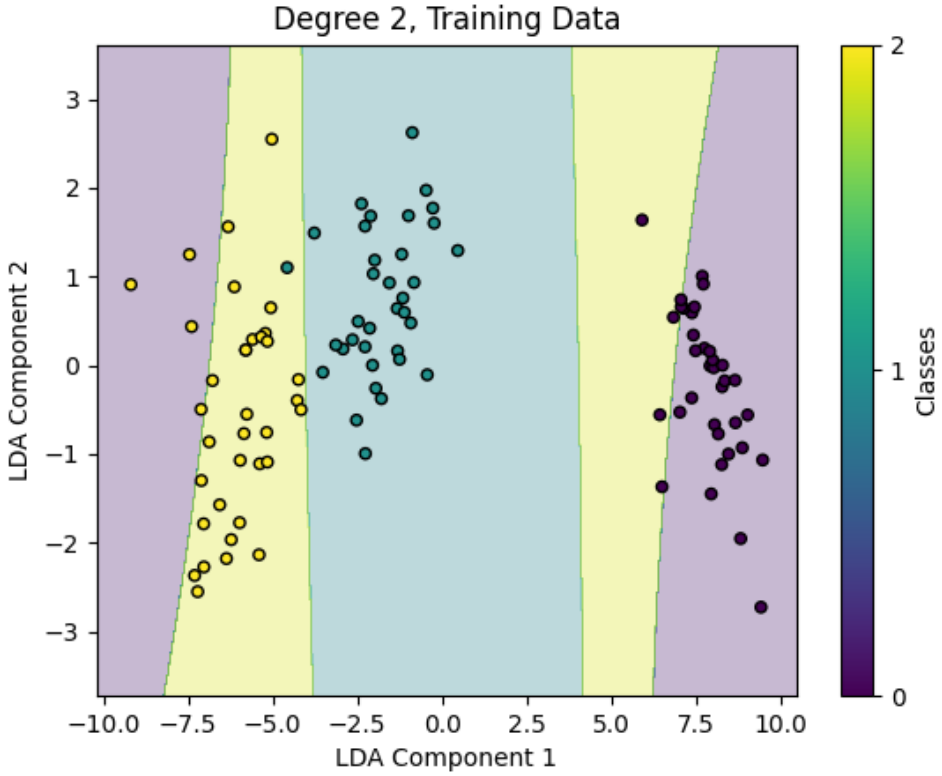
```
Degree: 1, Accuracy: 0.96
Degree: 2, Accuracy: 0.89
Degree: 3, Accuracy: 0.98
Degree: 4, Accuracy: 0.87
Degree: 5, Accuracy: 0.91
Degree: 6, Accuracy: 0.80
Degree: 7, Accuracy: 0.87
Degree: 8, Accuracy: 0.71
Degree: 9, Accuracy: 0.84
Degree: 10, Accuracy: 0.73
```

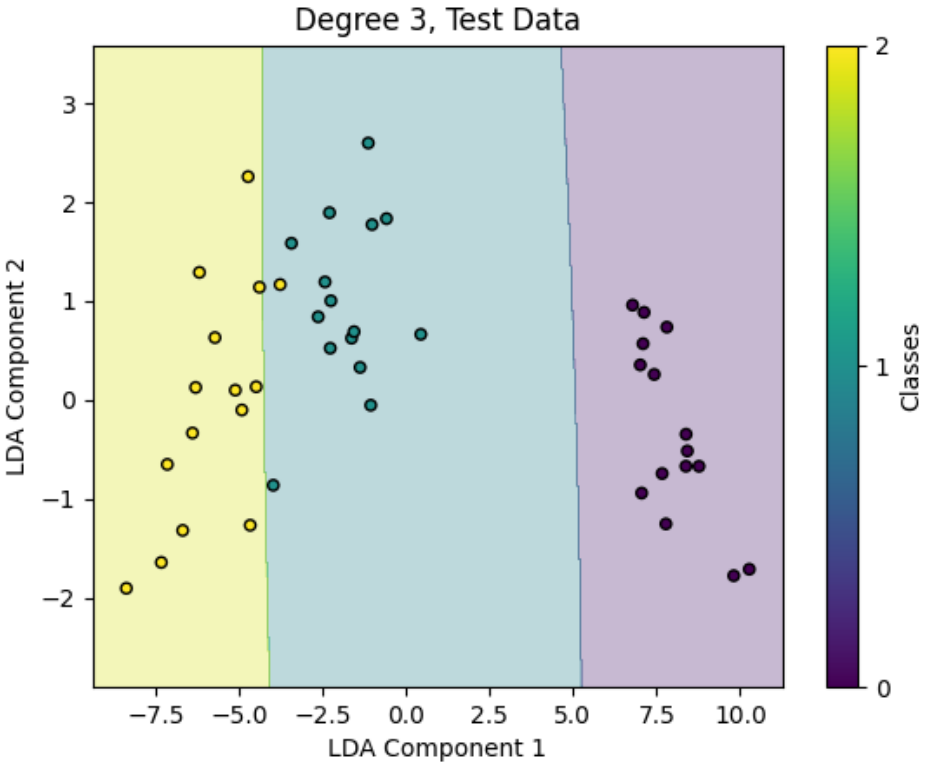
مشاهده می شود که دقت درجه ۳ از همه بالاتر است اما نمی خواهیم پیچیده سازی کنیم بنابراین دقت درجه ۱ کافی است.

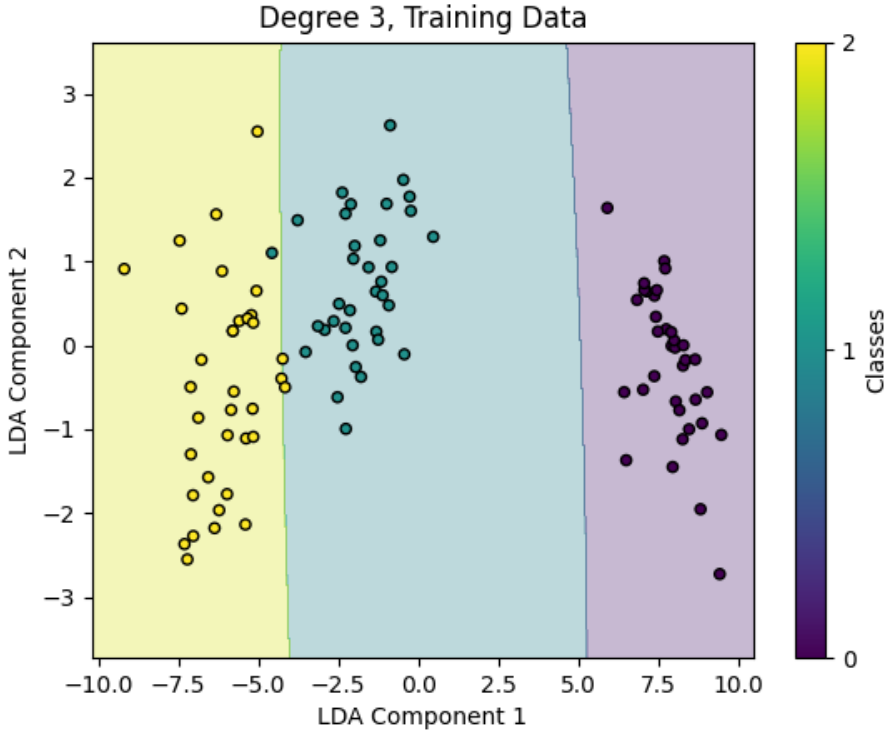

Degree 1, Test Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

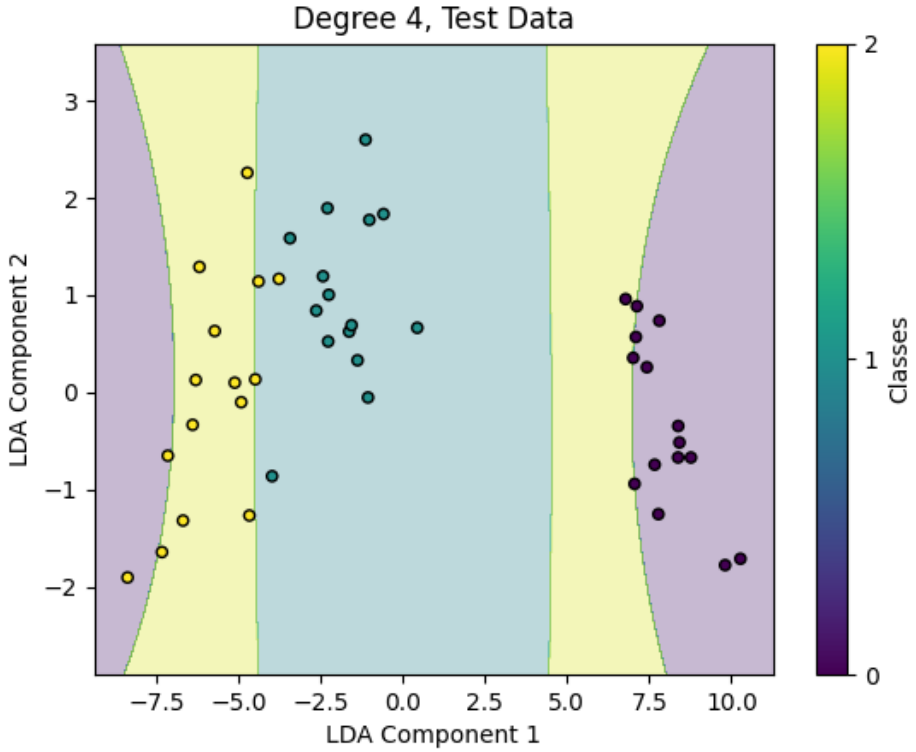

Degree 1, Training Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

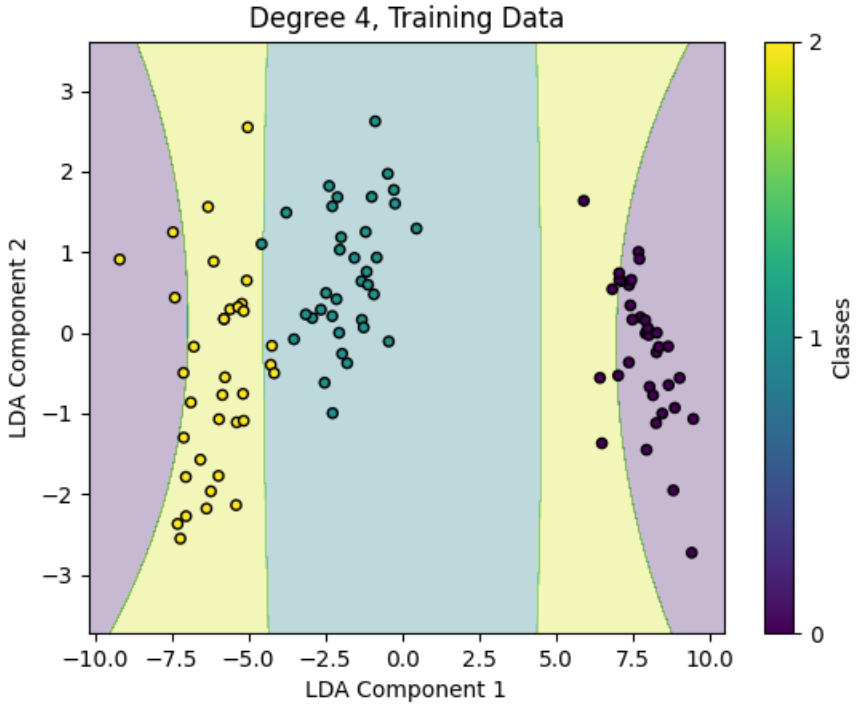

Degree 2, Test Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

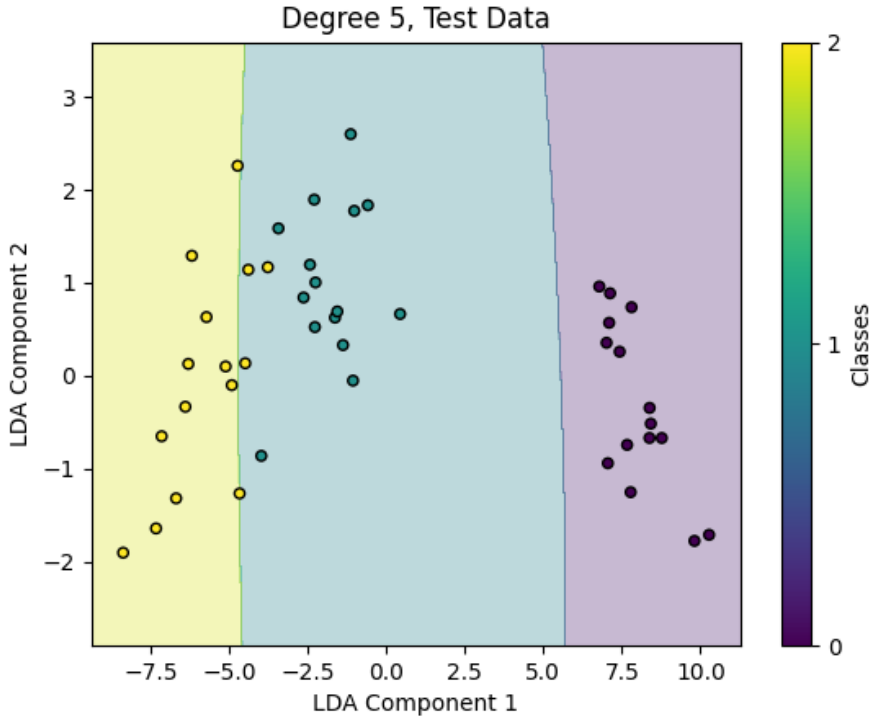

Degree 2, Training Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

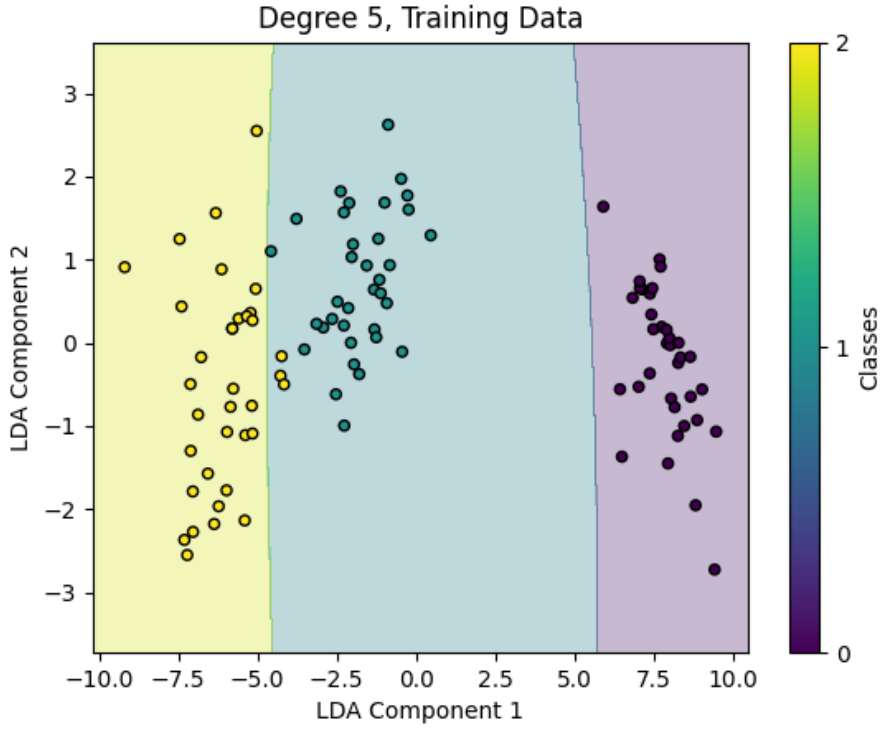

Degree 3, Test Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

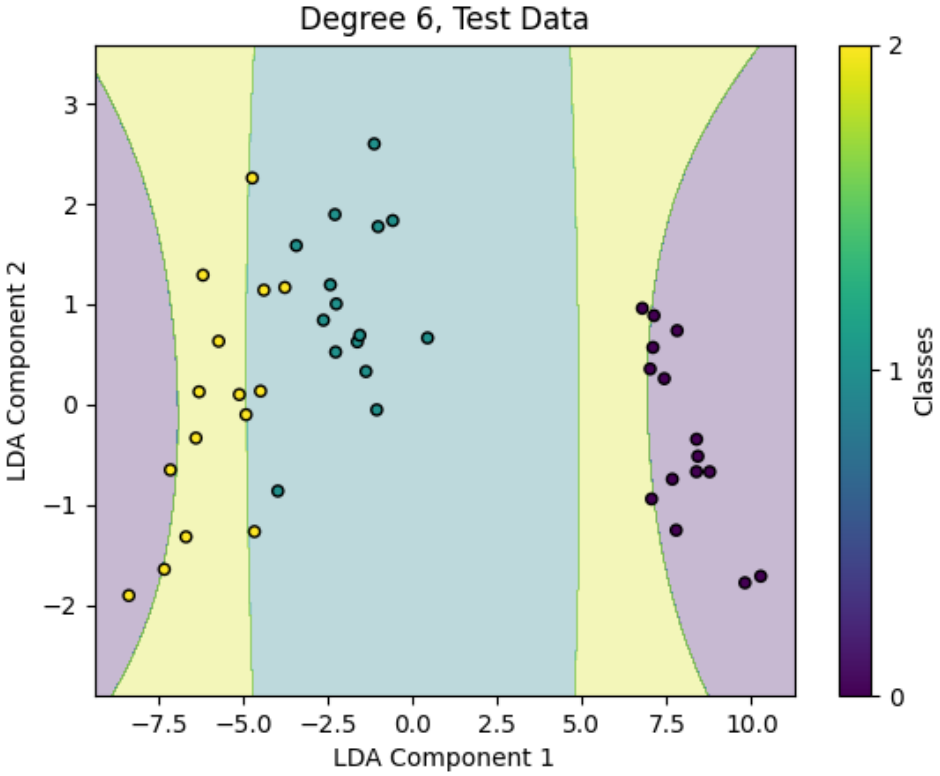

Degree 3, Training Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

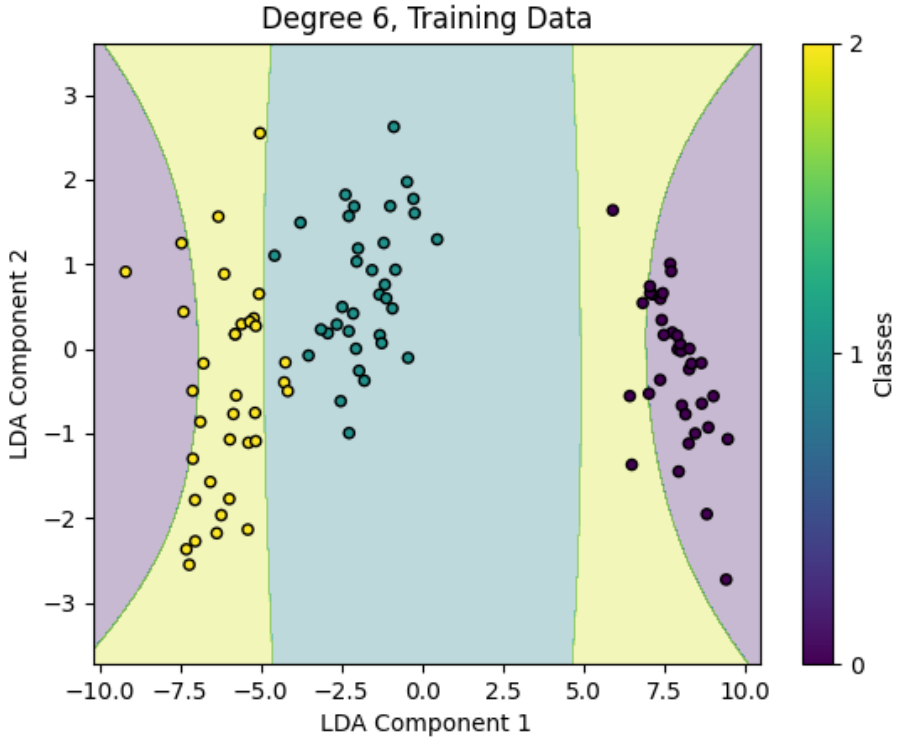

Degree 4, Test Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

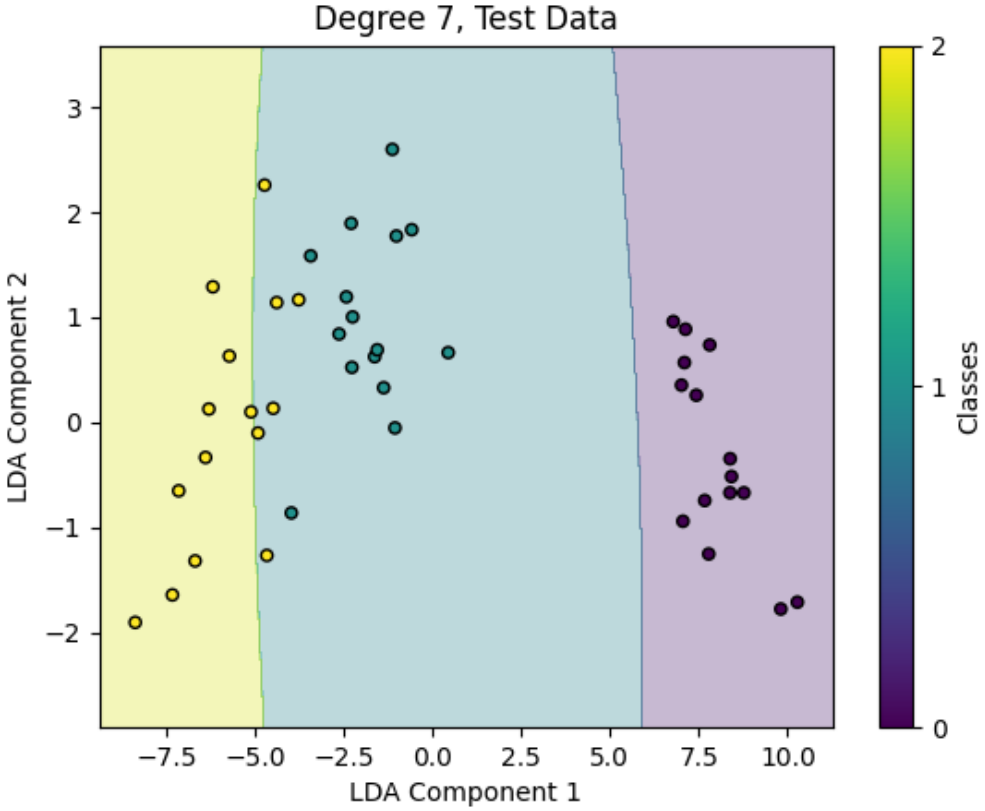

Degree 4, Training Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

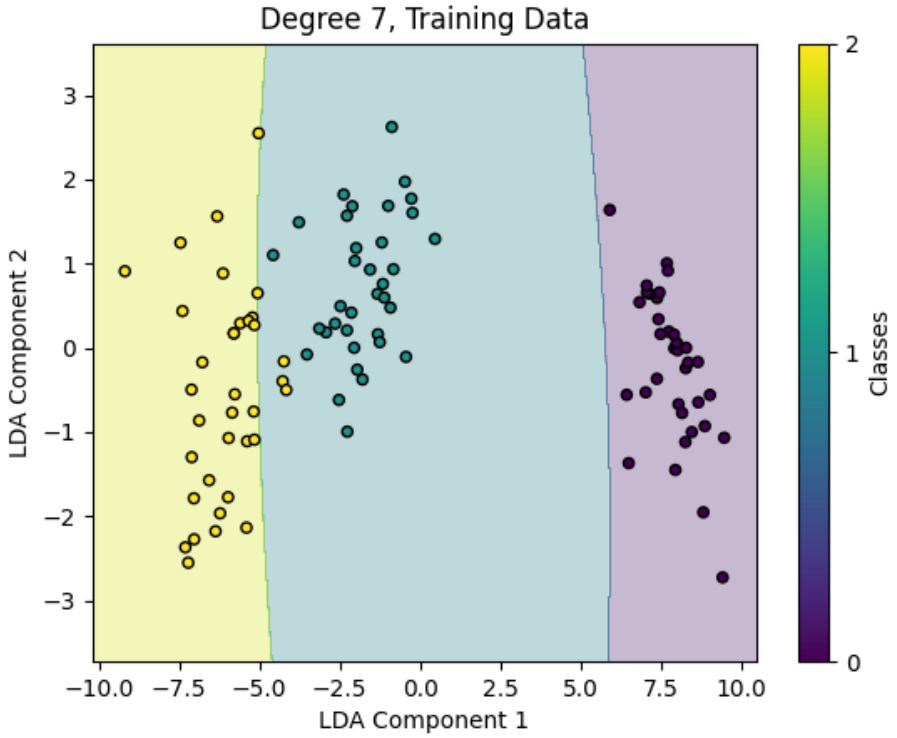

Degree 5, Test Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

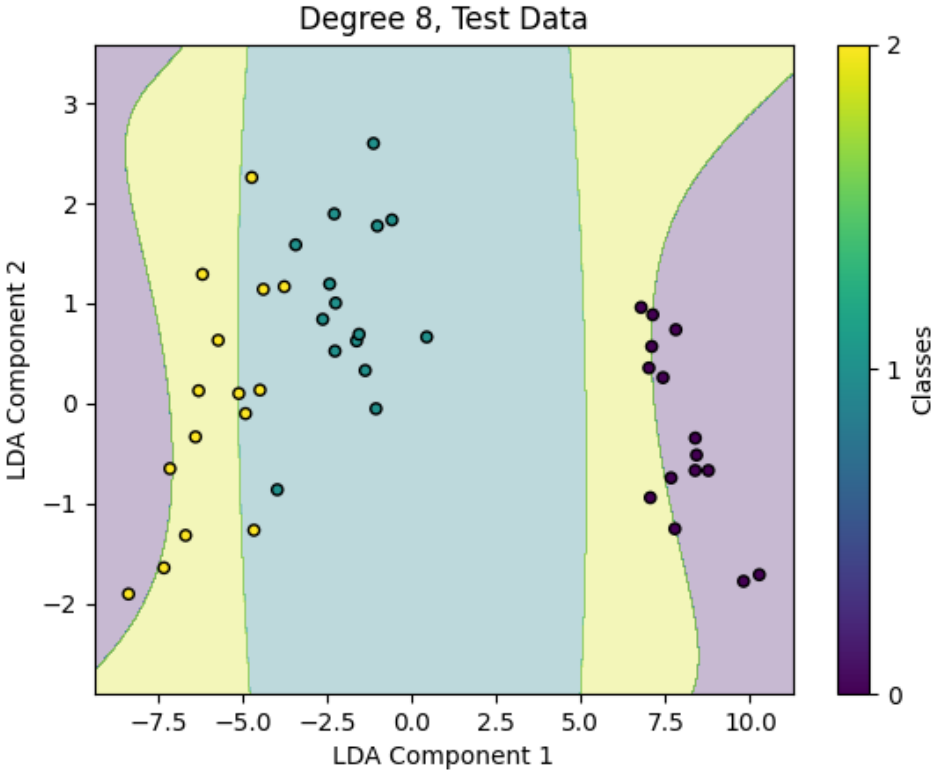

Degree 5, Training Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

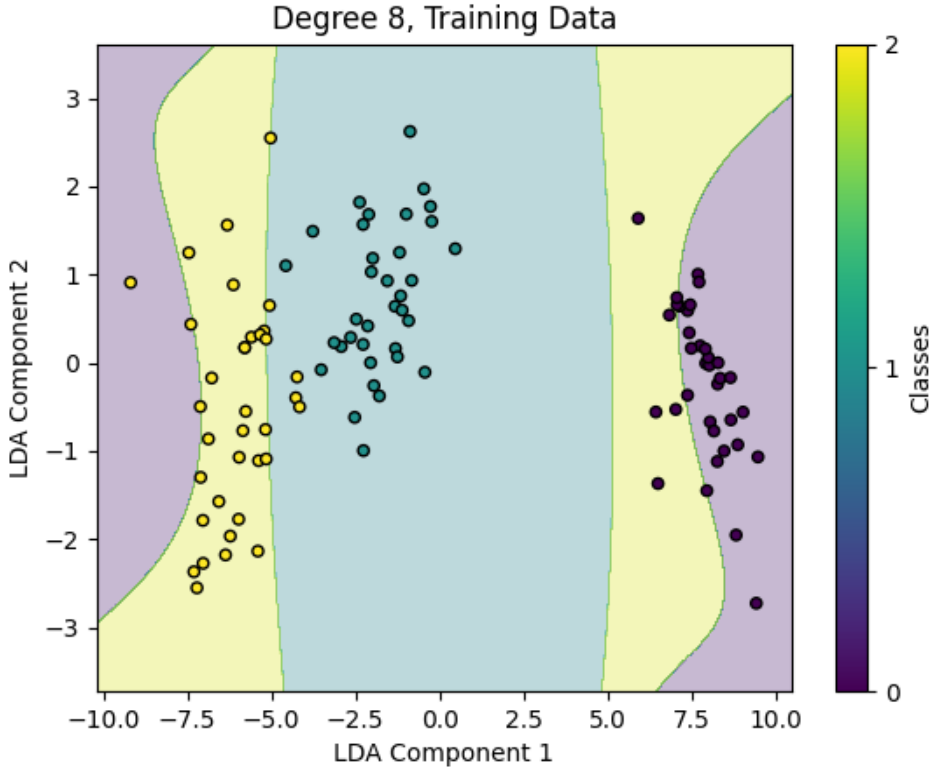

Degree 6, Test Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

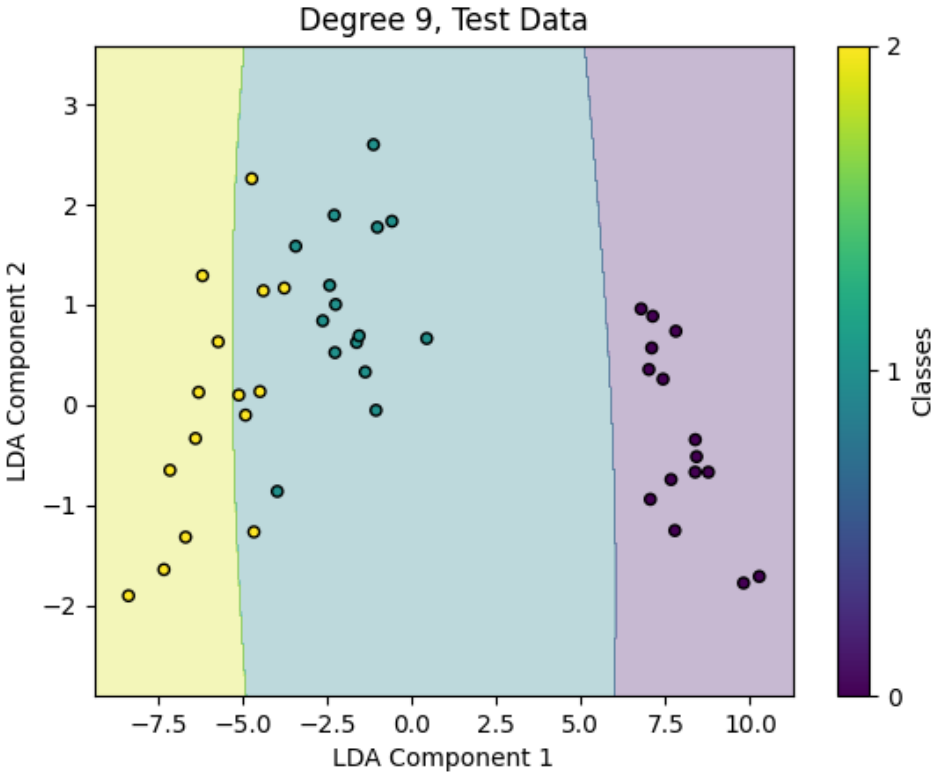

Degree 6, Training Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

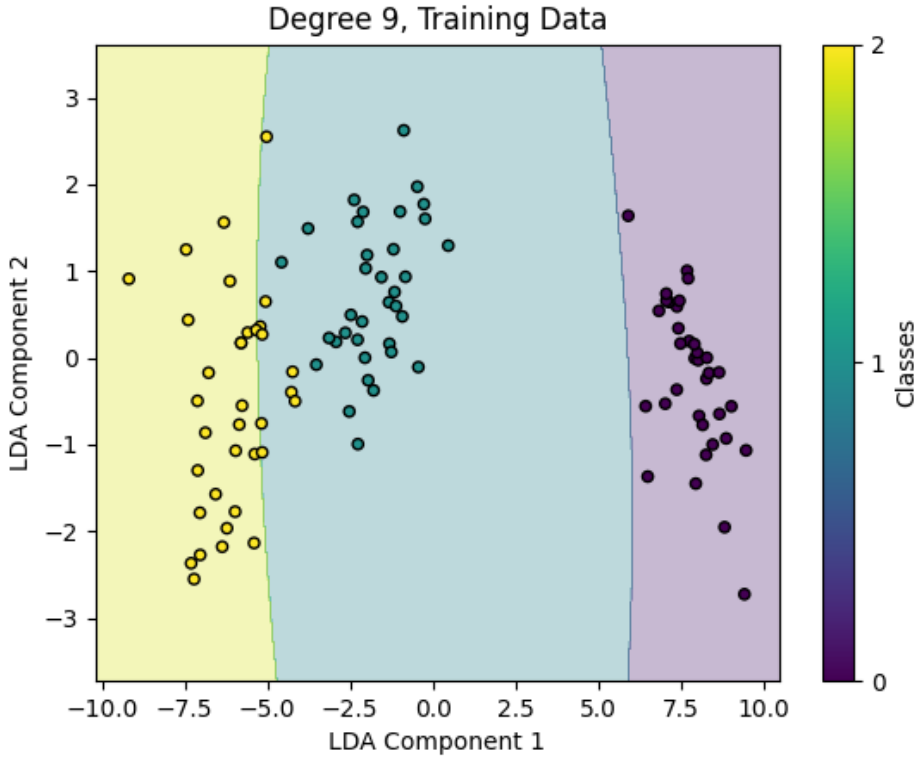

Degree 7, Test Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

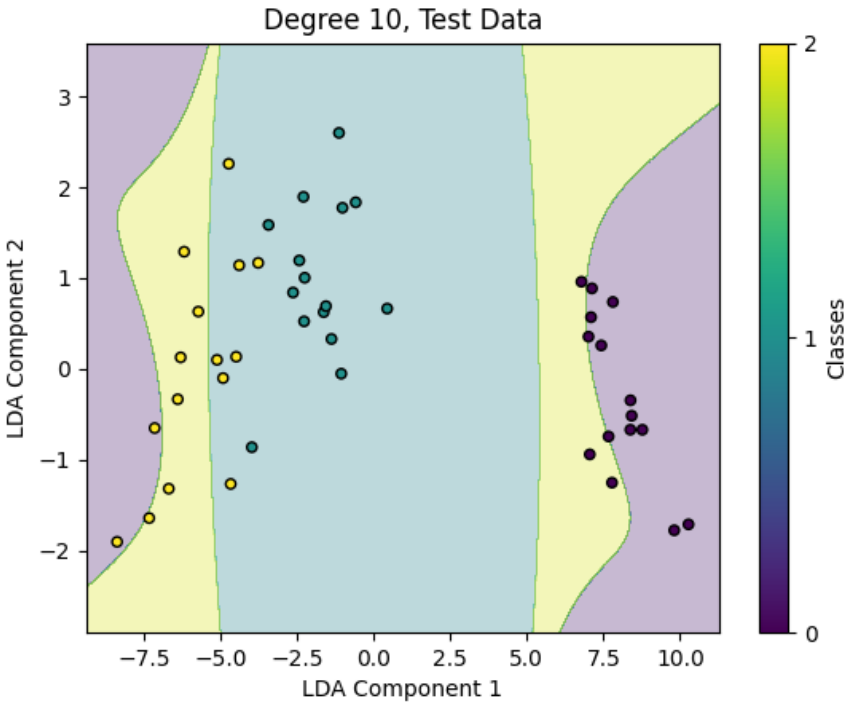

Degree 7, Training Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0

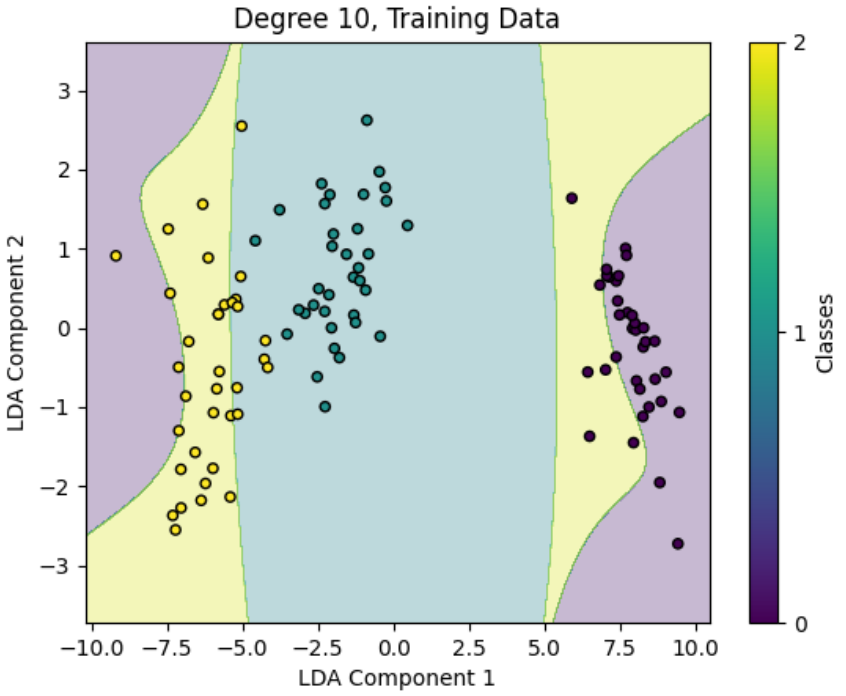

Degree 8, Test Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0


Degree 8, Training Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0


Degree 9, Test Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0


Degree 9, Training Data
LDA Component 2
LDA Component 1
Classes
3
2
1
0
−1
−2
−3
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
2
1
0





مرزهای تصمیم گیری را برای درجات مختلف مشاهده می کنید.

لینک GIF ساخته شده از تصاویر بالا

۱/۴. **د**

کلاس SVM تعریف شده:

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

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

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

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

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

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

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

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

        sv = alphas > 1e-10
        ind = np.arange(len(alphas))[sv]
```

```
        except ValueError as e:
            print("Solver failed due to rank deficiency.")
            return False

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

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

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

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

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

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

در ادامه، کاربرد هر یک از متدها توضیح داده شده است:

۱. **__init__**:

- سازنده کلاس است که پارامترهای اولیه مدل را تنظیم می‌کند.
- kernel: نوع هسته که در اینجا پیش‌فرض 'polynomial' است.
- C: پارامتر تنظیم(regularization parameter).
- gamma: پارامتر برای هسته چندجمله‌ای.
- degree: درجه هسته چندجمله‌ای.

۲. **polynomial_kernel**:

- محاسبه هسته چندجمله‌ای

۳. **fit**:

- آموزش مدل SVM با استفاده از داده‌های آموزشی X و برچسب‌ها y
- محاسبه ماتریس هسته K.
- تنظیم ماتریس‌های لازم برای حل مسئله بهینه‌سازی
- حل مسئله بهینه‌سازی برای به دست آوردن ضرایب آلفا.(alphas)
- شناسایی بردارهای پشتیبان.(support vectors)
- محاسبه مقدار بایاس (b) و بردار وزن (w) برای هسته خطی.
- خروجی True اگر آموزش موفقیت‌آمیز باشد و False در صورت بروز مشکل در حل مسئله بهینه‌سازی.

۴. **project**:

- محاسبه تابع تصمیم برای نمونه‌های داده X
- اگر هسته خطی باشد، از بردار وزن w و بایاس b استفاده می‌کند.
- در غیر این صورت، از بردارهای پشتیبان و ضرایب آلفا برای محاسبه تابع تصمیم استفاده می‌کند.

۵. **predict**:

- پیش‌بینی برچسب‌های نمونه‌های داده X با استفاده از تابع تصمیم.
- خروجی علامت تابع تصمیم (یعنی +۱ یا -۱) است.

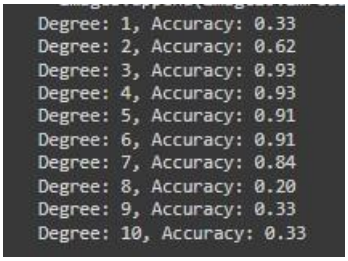
حال لازم است که با درجات مختلف آموزش دهیم:

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

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

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

از تابع بالا برای نمایش مرز تصمیم و ذخیره تصویر آن استفاده می شود. (مانند قسمت ج)

```python
results = []
images = []

for degree in range(1, 11):
    svm_models = []

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

    filename = plotSVC(X_train_lda, y_train, svm_models, degree, 'Training')
    images.append(imageio.imread(filename))
    filename = plotSVC(X_test_lda, y_test, svm_models, degree, 'Testing')
    images.append(imageio.imread(filename))

    y_test_pred = np.zeros(len(y_test))

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

    accuracy = accuracy_score(y_test, y_test_pred)
    results.append((degree, accuracy))

for degree, accuracy in results:
    print(f'Degree: {degree}, Accuracy: {accuracy:.2f}')

gif_filename = 'decision_boundaries2.gif'
imageio.mimsave(gif_filename, images, duration=1000)
```

کد بالا همان کد قسمت قبل است که برای درجات ۱ تا ۱۰ آموزش می دهد و پیشبینی را انجام می دهد و در نهایت یک gif ذخیره می کند. (مانند قسمت قبل)



در تصویر بالا Accuracy را برای درجات مختلف مشاهده می کنید. با توجه به اعداد بدست آمده در این حالت به نظر می رسد که حالت بهینه، درجه ۳ است.

۲۰ تصویر حاصل از مرزهای تصمیم گیری برای داده test و train در ۱۰ درجه مختلف در gif زیر موجود است. (چون در gif واضح هستند دیگر مانند قسمت قبل به صورت جدا در گزارش قرار داده نشدند.)

لینک gif حاصل از این ۲۰ تصویر مرزهای تصمیم گیری

۲. سوال سوم

۲/۱. آ

این مقاله به بررسی مسئله طبقه‌بندی داده‌های نامتوازن در تشخیص کلاهبرداری با کارت اعتباری می‌پردازد. برای متوازن‌سازی نمونه‌ها بین کلاس‌های اکثریت و اقلیت از الگوریتم بیش‌نمونه‌گیری استفاده می‌شود که ممکن است نویز ایجاد کند. مقاله یک الگوریتم شبکه عصبی خودرمزگذار رفع نویز (DAE) پیشنهاد می‌کند که علاوه بر بیش‌نمونه‌گیری، نویز را نیز رفع کرده و داده‌ها را طبقه‌بندی می‌کند. آزمایش‌ها نشان می‌دهند که این الگوریتم دقت طبقه‌بندی کلاس اقلیت را بهبود می‌بخشد و نسبت به روش‌های سنتی عملکرد بهتری دارد.

چالش‌های اصلی:

۱. **پروفایل رفتارهای تقلبی پویا** :تغییرات مستمر رفتارهای تقلبی که تشخیص را دشوار می‌سازد.

۲. **عدم توازن داده‌ها** :تعداد تراکنش‌های قانونی بسیار بیشتر از تقلبی‌ها است که دقت مدل‌های سنتی را کاهش می‌دهد.

۳. **انتخاب ویژگی‌های بهینه** :انتخاب نادرست ویژگی‌ها می‌تواند عملکرد مدل را کاهش دهد.

۴. **معیارهای ارزیابی مناسب** :معیارهای سنتی دقت نمی‌توانند عملکرد مدل را در داده‌های نامتوازن به خوبی نشان دهند.

روش‌های پیشنهادی:

۱. **Oversampling**:افزایش نمونه‌های کلاس اقلیت با استفاده از تکنیک SMOTE برای بهبود دقت تشخیص تراکنش‌های تقلبی.

۲. **شبکه‌های عصبی** :**Autoencoder** کاهش ابعاد داده‌ها و بازسازی آنها برای تشخیص بهتر الگوهای تقلب.

۳. **Denoising Autoencoder**:حذف نویز از داده‌های آموزشی برای بهبود دقت دسته‌بندی.

۴. **استفاده از معیارهای ارزیابی مختلف** :استفاده از معیارهایی مانند نرخ بازشناسی (Recall Rate) برای ارزیابی عملکرد مدل‌ها به جای معیار سنتی دقت.

نتیجه‌گیری:

استفاده از این روش‌ها به بهبود دقت و بازشناسی مدل تشخیص تقلب کمک کرده و چالش‌های موجود را به خوبی مدیریت می‌کند.

۲/۲. ب

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 و برای فرآیند نویزگیری طراحی شده است.
- داده‌های آموزشی با نویز گوسی وارد این شبکه می‌شوند و مدل خود رمزگذار آموزش می‌بیند تا نویز را از داده‌ها حذف کند.
- لایه‌ها شامل:
  - لایه ورودی با داده‌های نویزدار
  - چندین لایه کاملاً متصل با تعداد نورون‌های متغیر
  - استفاده از تابع زیان مربع برای بهینه‌سازی

۲. **طبقه‌بند:**

- این بخش شامل یک شبکه عصبی کاملاً متصل عمیق (Deep Fully Connected Neural Network) با ۶ لایه است.
- داده‌های نویزگیری شده 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، دقت طبقه‌بندی را بهبود بخشیده و مشکلات داده‌های نامتوازن را برطرف کرده است.

۲/۳. ج

ابتدا دیتا را دانلود و آماده می کنیم.

```
[4] !gdown 124_jH5kuOfoVdfFpV9qav4kVXYcuoliz
    data = pd.read_csv('creditcard.csv')
```

```
Downloading...
From (original): https://drive.google.com/uc?id=124_jH5kuOfoVdfFpV9qav4kVXYcuoliz
From (redirected): https://drive.google.com/uc?id=124_jH5kuOfoVdfFpV9qav4kVXYcuoliz&confirm=t&uuid=4ff5fbc4-6e2d-444b-94db-4a7b041a
To: /content/creditcard.csv
100% 151M/151M [00:01<00:00, 106MB/s]
```

```
# Handle missing values (if any)
data = data.dropna()
# Normalize the Amount Feature
scaler = StandardScaler()
data['Amount'] = scaler.fit_transform(data[['Amount']])
# Split the data into features and labels
features = data.drop(['Class'], axis=1)
labels = data['Class']
features = features.drop(['Time'] , axis = 1)
# Chech how many classes are in Labels
labels.value_counts()
```

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

```
[17] features.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	...	V20	V21	
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	...	0.251412	-0.018307	0.2
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	...	-0.069083	-0.225775	-0.6
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	...	0.524980	0.247998	0.7
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	...	-0.208038	-0.108300	0.6
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	...	0.408542	-0.009431	0.7

5 rows × 29 columns

بعد از دانلود کردن دیتا و سیو کردن آن به صورت دیتافریم اول از همه تمام سطر هایی که داده ای در آن وجود ندارد حذف میکنیم سپس ویژگی Amount را نرمالایز میکنیم با دستور standardscalar سپس ویژگی classرا جدا کرده و آنرا به صورت label ذخیره میکنیم و بقیه ویژگی ها را به صورت features ذخیره میکنیم و ویژگی time را نیز از آنها حذف میکنیم. دیتای ما حاوی ۲ کلاس است.

```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=RS)

# Oversample the training data using SMOTE
smote = SMOTE(random_state=RS)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print(y_train.value_counts())
y_train_resampled.value_counts()
```

```
Class
0    227441
1       404
Name: count, dtype: int64
Class
0    227441
1    227441
Name: count, dtype: int64
```

داده ها را با دستور train test split از هم جدا میکنیم با نسبت ۲. سپس روش smote را بر روی داده های train استفاده میکنیم.

```python
# Denoising Autoencoder
dae = Sequential([
    GaussianNoise(0.1, input_shape=(X_train_resampled.shape[1],)),
    Dense(29, activation='relu'),
    Dense(22, activation='relu'),
    Dense(15, activation='relu'),
    Dense(10, activation='relu'),
    Dense(15, activation='relu'),
    Dense(22, activation='relu'),
    Dense(29, activation='relu')
])
dae.compile(optimizer='adam', loss='mse')

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

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

سپس بخش DAE است.

ورودی آن داده های oversampling با نویزگوسی با انحراف معیار ۱ میباشد و optimizer و تابع اتلاف نیز معلوم هستند .

سپس داده های را فیت کرده و predict می کنیم.

در قسمت earlystopping در واقع ما داده های validation (که ۲۰ درصد داده های train هستند) خود را معیاری برای توقف آموزش مدل قرار دادیم بدین صورت که اگر تا ۱۰ epoch، validation loss ما بهتر نشود مدل متوقف میشود و وزن های مدل ها نیز save شده اند تا بهترین مدل نیز انتخاب شود

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

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

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

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

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

مدل ما به شکل بالا می باشد.

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

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

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

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

```python
# Load the saved model
model = load_model('best_model.h5')

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

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

۲/۴. د

دقت در مجموعه داده های نامتعادل:

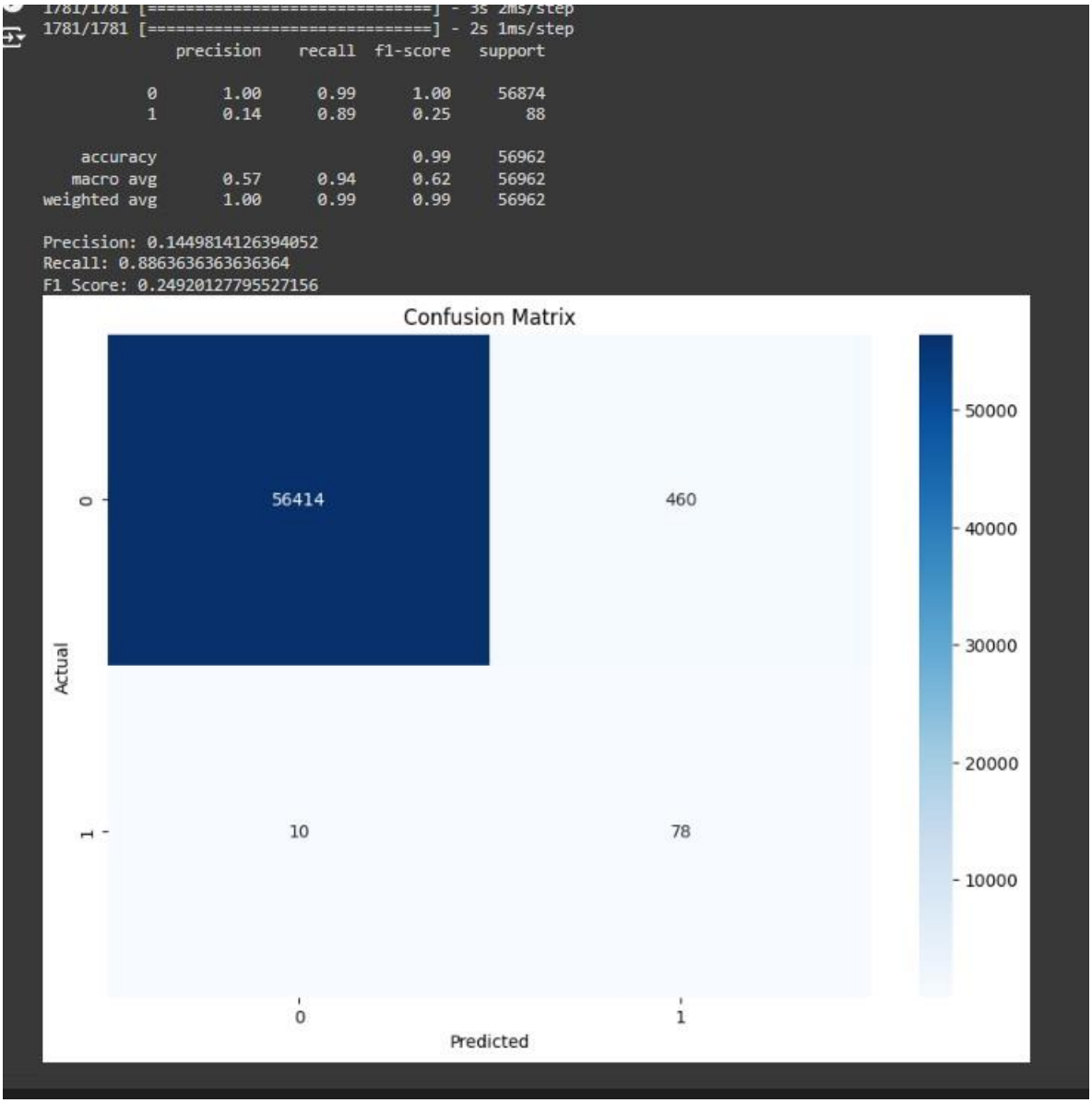
مسئله: دقت معیار مناسبی برای مجموعه داده های نامتعادل نیست زیرا می تواند گمراه کننده باشد. به عنوان مثال، اگر ۹۹ درصد تراکنشها مشروع باشند، مدلی که همه تراکنشها را مشروع پیشبینی می کند، ۹۹ درصد دقت را خواهد داشت.در اینجا معمولا از confusion matrix استفاده میکنند.

معیارهای جایگزین:

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

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

# Calculate metrics
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = classification_report(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print( accuracy)
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')

# Plot the confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

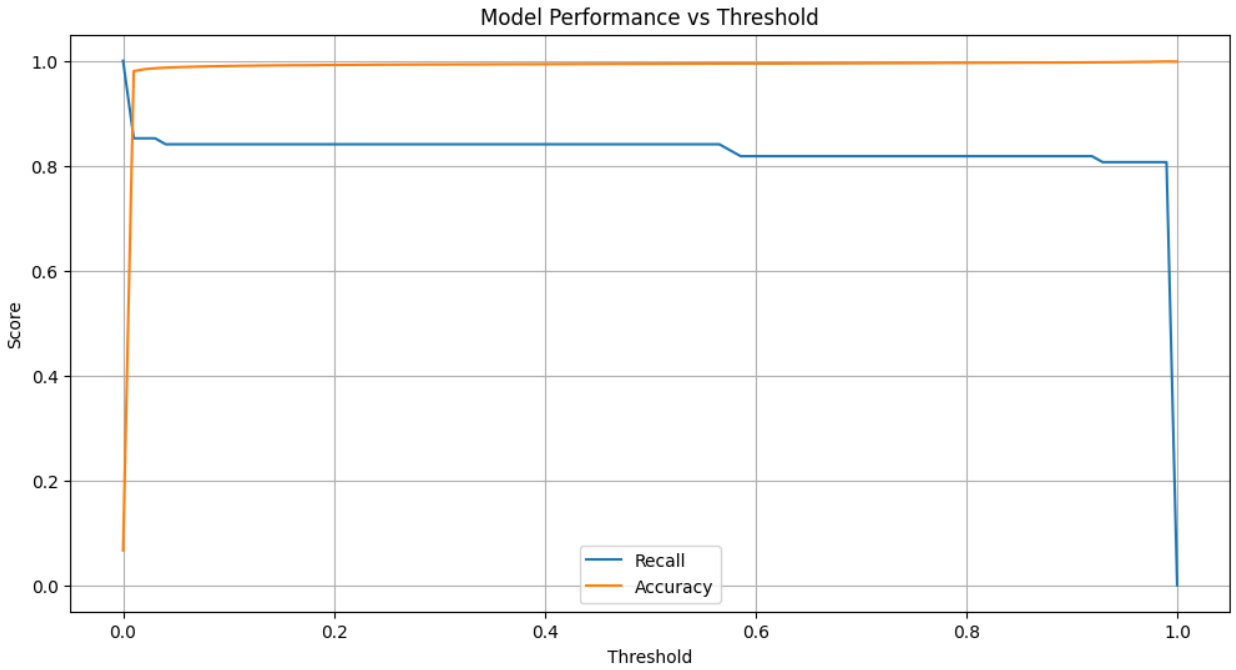
مرحله بعدی این است که داده های تست که denoise شده اند را وارد مدل classifier خود کنیم و آنرا ارزیابی کنیم. سپس classification report و confusion matrix را میکشیم.



خب همانطور که مشاهده می کنید مدل ما recall خوبی دارد. بین داده های کلاس ۱ توانسته ۰.۸۹ درست پیشبینی کند.

۲/۵. ه

آستانه برای پیش بینی (threshold) :



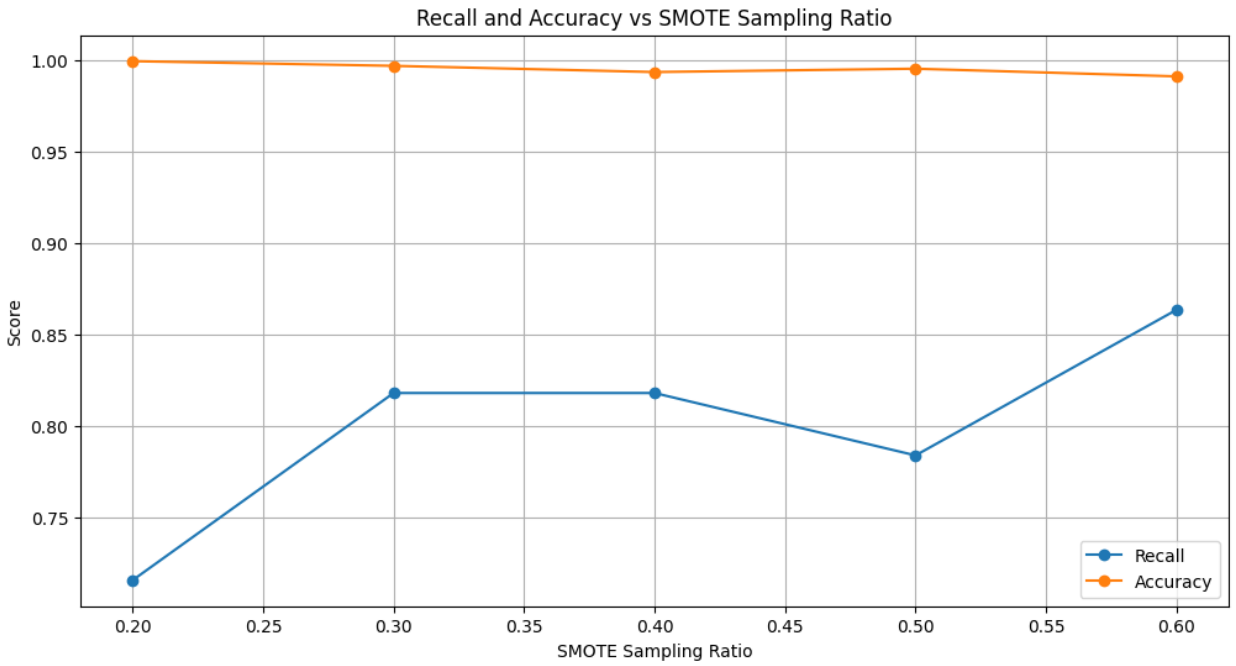
همینطور که میبینید نمودار تقریبا با نمودار مقاله یکی است .همینطور که میبینید در threshold = .5 که مقدار بخش های قبلی میباشد recall = .89 و accuracy = .99 شده اند.در آخر نیز مدل recall آن برابر ۰ میشود و دقت برابر ۱۰۰ میشود که این یعنی مدل نتوانسته هیچ کدام از داده های کلاس تقلب را اندازه گیری کند. قاعدتا واضح است که اگر آستانه را کم تر کنیم مدل بیشتر کلاس ها را کلاس داده های تقلب در نظر میگیرد زیرا داده های بیشتری و جود دارند که از آستانه بیشتر باشند پس به کلاس ۱ تعلق میگیرند و در این صورت داده های سالم را درست تشخیص نمی دهد برای همین accuracy آن کم است.

آستانه برای oversampling:

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

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

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

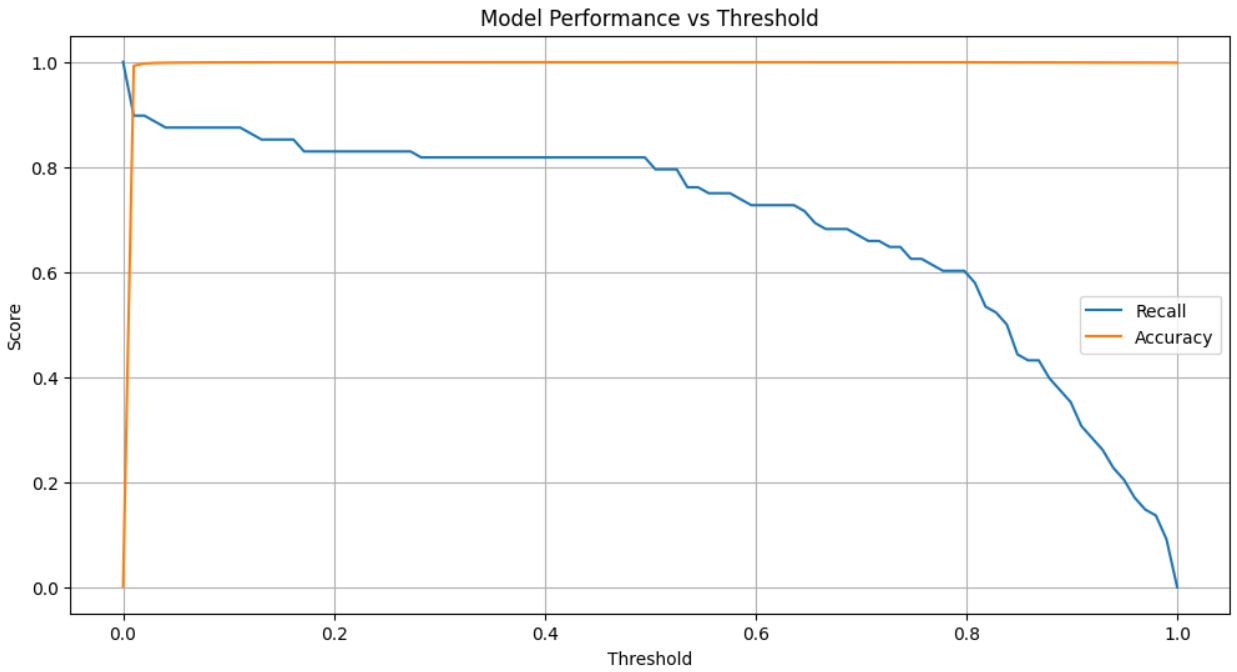
نتیجه:



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

۲/۶. و

در اینجا از DEA و oversampling استفاده نمیکنیم یعنی در واقع تمام داده ها را به classifier خود می دهیم که بدین شکل میشود.



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

در حالت معمول : چون معمولا threshold برابر ۰.۵ است میتوان از روی نمودار خواند که accuracy = 100 و recall = 0.78 می باشد.

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

همانطور که در بخش قبل دیدیم نمودار ما تا قبل از آستانه تقریبا ۰.۹ توانسته بالای ۸۰ درصد recall داشته باشد

و accuracy آن نیز بالا است ولی در اینجا بعد از تقریبا آستانه ۰.۴۵ این مقدار کم تر از ۰.۸ شده است ولی accuracy آن نیز همیشه (به جز آستانه های اول) تقریبا ۱۰۰ است. زیرا درصد داده های کلاس ۱ خیلی کم می باشد و مدل نتوانسته این کلاس را به خوبی یاد بگیرد و کلاس ۰ را به خوبی یاد گرفته است.