



تمرین سری سوم درس یادگیری ماشین

استاد:

دکتر مهدی علیاری شوره‌دلی

دانشجو:

نازنین بندرایان

۴۰۲۱۳۸۷۶

بهار ۱۴۰۳

فهرست مطالب

صفحة

عنوان

٦ سوال اول ١
٦ ١-١- بخش الف
١٥ ٢-١- بخش ب
٢٤ ٣-١- بخش ج
٣٤ ٤-١- بخش د
٥٠ ٢- سوال سوم
٥٠ ٢-١- بخش الف
٥١ ٢-٢- بخش ب
٥٢ ٢-٣- بخش ج
٥٨ ٢-٤- بخش د
٥٩ ٥-٢- بخش ه
٦١ ٦-٢- بخش و

فهرست شکل‌ها

صفحه	عنوان
۶	شکل ۱-۱ داده‌های نخستین دیتافرم – نتایج دستور <code>df.head()</code>
۶	شکل ۱-۲ نتایج دستور <code>df.info()</code>
۷	شکل ۱-۳ بررسی عدم وجود داده پوچ
۷	شکل ۱-۴ ابعاد دیتاست و تعداد نمونه داده‌های موجود در هر کلاس
۷	شکل ۱-۵ تعداد نمونه داده‌های موجود در هر کلاس
۸	شکل ۱-۶ نمودار هیستوگرام
۸	شکل ۱-۷ مقدار میانگین و واریانس هر ویژگی
۹	شکل ۱-۸ میزان همبستگی بین ویژگیها
۹	شکل ۱-۹ نمودار میزان همبستگی داده‌ها
۱۰	شکل ۱-۱۰ رسم ویژگیها به صورت دو تایی
۱۰	شکل ۱-۱۱ نمایش نحوه توزیع داده‌ها
۱۲	شکل ۱-۱۲ مجموعه داده Swiss Roll. حفظ فاصله کوچک با t-SNE (خط توپر) در مقایسه با بیشینه‌سازی واریانس.
۱۲	شکل ۱-۱۳ سنجش مشابهت دوتایی‌ها در فضای ابعاد بالا
۱۳	شکل ۱-۱۴ توزیع نرمال و توزیع تی-استیویدنت
۱۴	شکل ۱-۱۵ ابعاد داده‌های ورودی و خروجی آموزش و آزمایش
۱۵	شکل ۱-۱۶ نمایش نمونه‌های دیتاست توسط t-SNE
۱۶	شکل ۱-۱۷ نمایش PCA برای دو بعد
۱۷	شکل ۱-۱۸ نمودار مربوط به variance explained
۱۸	شکل ۱-۱۹ نمایش نمونه‌های دیتاست توسط PCA
۱۸	شکل ۱-۲۰ کاهش بعد از کاهش بعد با استفاده از LDA
۱۸	شکل ۱-۲۱ نمونه‌های کاهش بعد یافته شده توسط LDA
۱۹	شکل ۱-۲۲ نمایش نمونه‌های دیتاست بعد از کاهش بعد توسط LDA
۲۰	شکل ۱-۲۳ مدل آموزش دیده
۲۰	شکل ۱-۲۴ مشخصات ماشین آموزش دیده شده
۲۱	شکل ۱-۲۵ ارزیابی عملکرد ماشین
۲۱	شکل ۱-۲۶ ماتریس درهمیریختگی
۲۲	شکل ۱-۲۷ خطوط جداکننده داده‌های آموزش
۲۳	شکل ۱-۲۸ خطوط جداکننده داده‌های آزمون

۲۴	شکل ۱-۲۹ خطوط تصمیم داده‌های آموزش
۲۴	شکل ۱-۳۰ خطوط تصمیم داده آزمون
۲۷	شکل ۱-۳۱ نمایش میزان دقت مدل با درجات ۱۰-۱
۲۸	شکل ۱-۳۲ خطوط جداکننده و بردار پشتیبان داده‌های آموزش
۲۹	شکل ۱-۳۳ خطوط جداکننده و بردار پشتیبان داده‌های آزمایش
۳۰	شکل ۱-۳۴ ماتریس درهمریختگی
۳۱	شکل ۱-۳۵ محل ذخیره‌سازی gifها
۳۲	شکل ۱-۳۶ خطوط تصمیم داده‌های آموزش
۳۳	شکل ۱-۳۷ خطوط تصمیم داده‌های آموزش
۳۴	شکل ۱-۳۸ محل ذخیره‌سازی gifها
۳۴	شکل ۱-۳۹ دقت الگوریتم با افزایش درجه
۳۵	شکل ۱-۴۰ ابعاد داده‌های آموزش و آزمایش
۴۲	شکل ۱-۴۱ نمایش میزان دقت مدل با درجات ۷-۱
۴۲	شکل ۱-۴۲ دقت الگوریتم با افزایش درجه
۴۳	شکل ۱-۴۳ مرزهای تصمیم داده‌های آموزش
۴۴	شکل ۱-۴۴ مرزهای تصمیم داده‌های آزمون
۴۴	شکل ۱-۴۵ ماتریس درهمریختگی
۴۵	شکل ۱-۴۶ محل ذخیره‌سازی gifها
۴۵	شکل ۱-۴۷ نمایش میزان دقت مدل با درجات ۱۰-۱
۴۶	شکل ۱-۴۸ دقت الگوریتم با افزایش درجه
۴۷	شکل ۱-۴۹ مرزهای تصمیم داده‌های آموزش
۴۸	شکل ۱-۵۰ مرزهای تصمیم داده‌های آزمون
۴۹	شکل ۱-۵۱ ماتریس درهمریختگی
۴۹	شکل ۱-۵۲ محل ذخیره‌سازی gifها
۵۰	شکل ۲-۱ ساختار autoencoder
۵۱	شکل ۲-۲ مراحل عملکرد شبکه
۵۳	شکل ۲-۳ ۵ ردیف ابتدایی دیتاست
۵۳	شکل ۲-۴ نمایش تعداد داده‌های هر کلاس
۵۴	شکل ۲-۵ تعداد داده‌های هر کلاس پس از SMOT
۵۵	شکل ۲-۶ مدل Autoencoder
۵۶	شکل ۲-۷ نمودار هزینه شبکه autoencoder
۵۷	شکل ۲-۸ مدل classifier
۵۸	شکل ۲-۹ نمودار هزینه شبکه classifier

شکل ۲-۱۰ ماتریس درهمیریختگی داده‌های آزمون	۵۸
شکل ۲-۱۱ گزارش از نحوه عملکرد سیستم	۵۸
شکل ۲-۱۲ تاثیر نسبت‌های مختلف برای استفاده از SMOTE در عملکرد سیستم	۶۰
شکل ۲-۱۳ تاثیر نسبت‌های مختلف برای استفاده از SMOTE در عملکرد سیستم	۶۰
شکل ۲-۱۴ نمودار تابع هزینه مربوط به classifier	۶۲
شکل ۲-۱۵ گزارش عملکرد سیستم	۶۲
شکل ۲-۱۶ ماتریس درهمیریختگی	۶۳

<https://github.com/nazaninbondarian/MachineLearning2024/tree/main/Chapter%203>

۱- سوال اول

۱-۱- بخش اول

این مجموعه داده شامل ۳ نوع مختلف گل زنبق (Virginica و Versicolour و Setosa) طول گلبرگ و کاسبرگ است که در یک numpy.ndarray ۱۵۰x۴ ذخیره شده است.

سطرها نمونه و ستون‌ها عبارتند از: طول کاسبرگ، عرض کاسبرگ، طول گلبرگ و عرض گلبرگ.
در ابتدا ما کتابخانه‌های مورد نیاز را می‌افزاییم.

```
# Import necessary libraries
from sklearn import datasets
import pandas as pd
import numpy as np
```

سپس دیتاست مورد نظر را از سایت سایکیتلرن فراخوانی می‌کنیم. ویژگی‌های ورودی و خروجی مورد نظر را دسته‌بندی می‌کنیم و به صورت دیتافرم ذخیره می‌کنیم.

```
# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target
feature_names = iris.feature_names
```

```
# Convert to DataFrame for easier manipulation
df = pd.DataFrame(X, columns=feature_names)
df['species'] = y
```

سپس با استفاده از دستور زیر سرستون‌ها و پنج داده‌ی نخست دیتاست را نشان می‌دهیم.

```
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
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

شکل ۱-۱ داده‌های نخستین دیتافرم - نتایج دستور df.head()

با استفاده از دستور زیر اطلاعات در رابطه با دیتاست همچون عنوان‌های هر ستون و نوع داده‌های هر ستون را نمایش می‌دهیم.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   sepal length (cm) 150 non-null    float64
 1   sepal width (cm)  150 non-null    float64
 2   petal length (cm) 150 non-null    float64
 3   petal width (cm)  150 non-null    float64
 4   species          150 non-null    int64  
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

شکل ۱-۲ نتایج دستور df.info()

سپس بررسی صورت می‌گیرد که آیا داده null وجود دارد یا خیر. از آنجایی که یافت نشد نیاز به حذف ستونی نمی‌باشد.

```
print(Normal.isnull().sum())
```

```
    sepal length (cm)      0  
    sepal width (cm)      0  
    petal length (cm)     0  
    petal width (cm)      0  
    species                0  
    dtype: int64
```

شکل ۱-۳ بررسی عدم وجود داده پوچ

با استفاده از دستورات زیر ابعاد دیتاست و تعداد نمونه داده‌های موجود در هر کلاس را نمایش می‌دهیم.

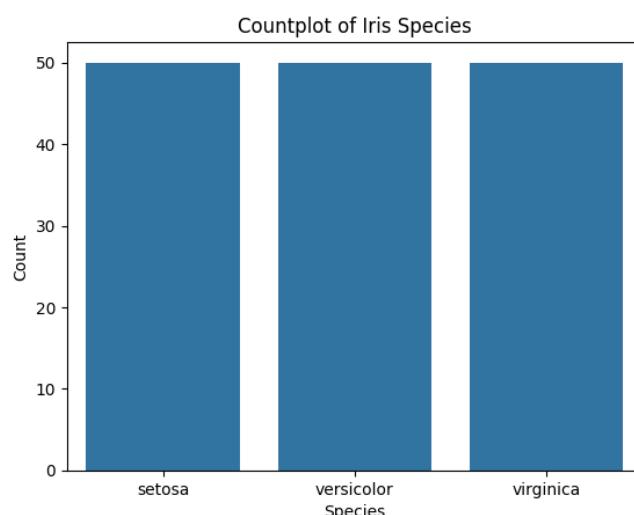
```
print("Dimensions:", df.shape)  
print("Sample size per species:\n", df['species'].value_counts())
```

```
Dimensions: (150, 5)  
Sample size per species:  
species  
0      50  
1      50  
2      50  
Name: count, dtype: int64
```

شکل ۱-۴ ابعاد دیتاست و تعداد نمونه داده‌های موجود در هر کلاس

مقادیر داده‌های موجود در هر کلاس را توسط دستور زیر به نمایش در می‌آوریم.

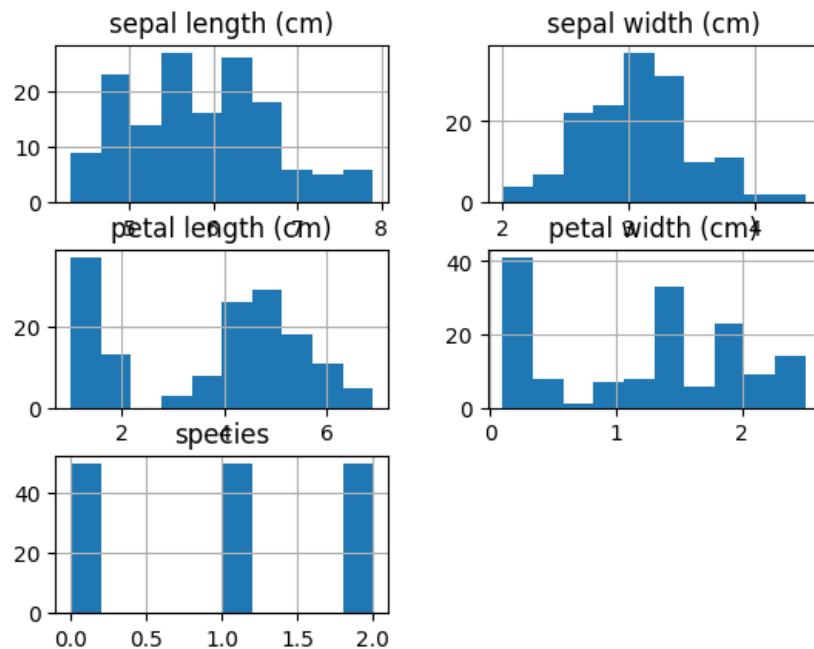
```
import seaborn as sns  
import matplotlib.pyplot as plt  
  
df_copy = df.copy(deep=True)  
df_copy['species'] = df_copy['species'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})  
  
sns.countplot(x='species', data=iris_df)  
plt.title('Countplot of Iris Species')  
plt.xlabel('Species')  
plt.ylabel('Count')  
plt.show()
```



شکل ۱-۵ تعداد نمونه داده‌های موجود در هر کلاس

نمودار هیستوگرام مربوط به داده‌ها را به صورت زیر رسم می‌نماییم.

df.hist()



شکل ۱-۶ نمودار هیستوگرام

با استفاده از دستورات زیر میزان میانگشن واریانس هر ویژگی را محاسبه می‌کنیم و مقدار آن را نمایش می‌دهیم.

mean = df.mean()

variance = df.var()

```
print("Mean:\n", mean)
print("Variance:\n", variance)
```

```
Mean:
    sepal length (cm)      5.843333
    sepal width (cm)       3.057333
    petal length (cm)      3.758000
    petal width (cm)       1.199333
    species                 1.000000
    dtype: float64
Variance:
    sepal length (cm)      0.685694
    sepal width (cm)       0.189979
    petal length (cm)      3.116278
    petal width (cm)       0.581006
    species                 0.671141
    dtype: float64
```

شکل ۱-۷ مقدار میانگین و واریانس هر ویژگی

با استفاده از دستور زیر میزان همبستگی ویژگی‌ها را به صورت متنی نمایش می‌دهیم.

correlation = df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']].corr()

```
print("Correlation:\n", correlation)
```

```

Correlation:
      sepal length (cm)  sepal width (cm)  petal length (cm) \
sepal length (cm)          1.000000       -0.117570        0.871754
sepal width (cm)         -0.117570        1.000000       -0.428440
petal length (cm)          0.871754       -0.428440        1.000000
petal width (cm)           0.817941       -0.366126        0.962865

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

```

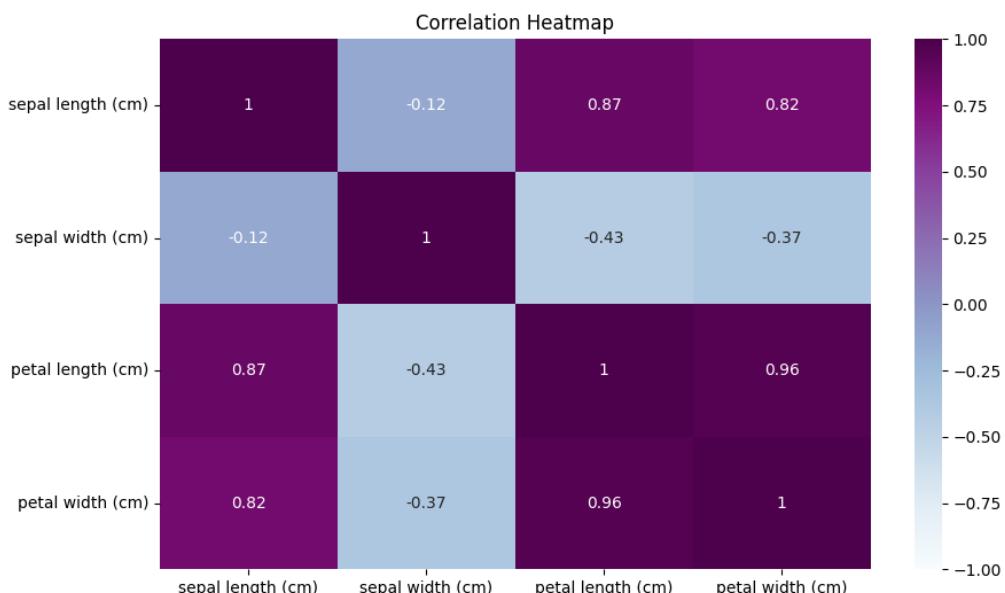
شکل ۱-۸ میزان همبستگی بین ویژگی‌ها

برای نمایش میزان همبستگی به صورت تصویر از دستورات زیر استفاده شده است. همانطور که می‌دانیم هرچقدر میزان همبستگی به عدد ۱ نزدیک باشد، به این معنی است که دو ویژگی به یکدیگر وابسته خطی‌تر می‌باشد و اطلاعات جدیدی را به ما ارائه نمی‌دهد.

```

corr=df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']].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()

```



شکل ۱-۹ نمودار میزان همبستگی داده‌ها

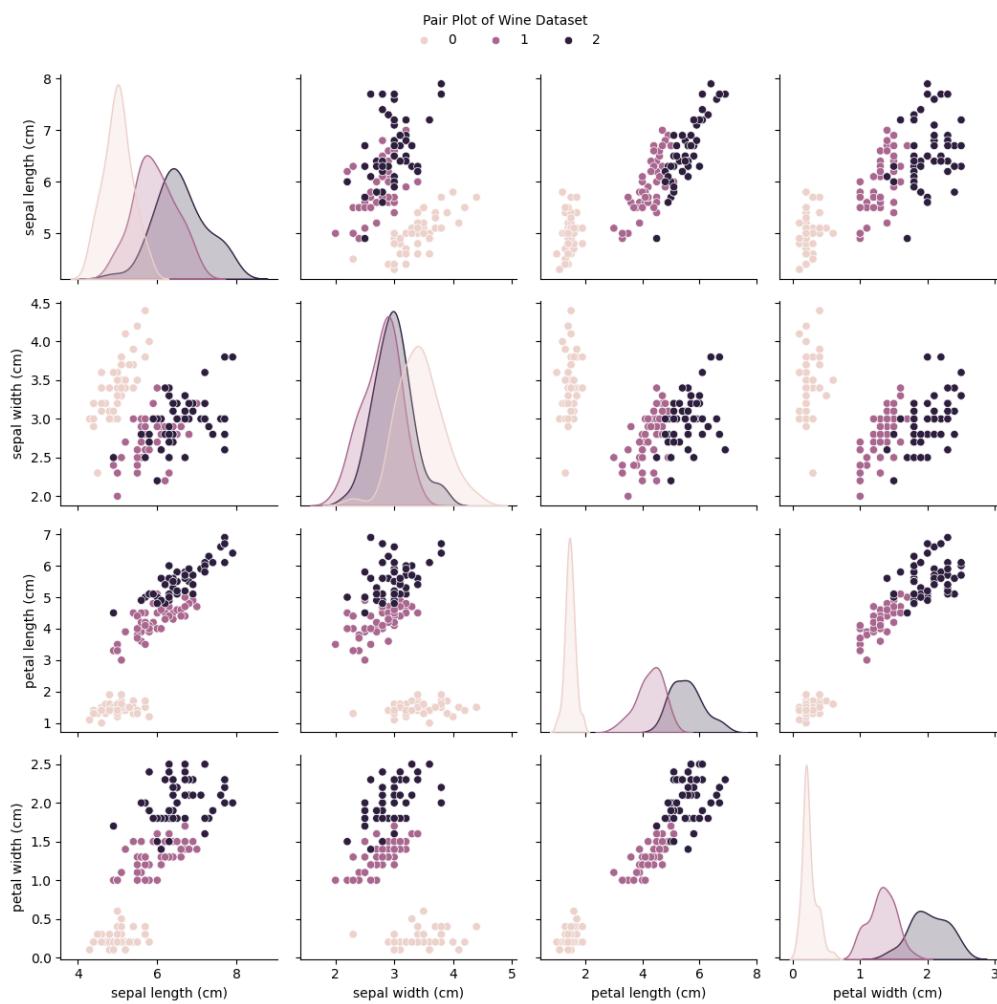
با در نظر گرفتن ویژگی‌ها به صورت دو به دو داده‌ها را رسم می‌کنیم. در قطر اصلی توزیع نمونه‌ها رسم شده است.

```

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

```

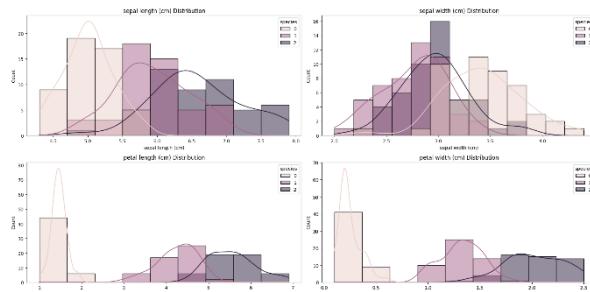
```
plt.tight_layout()
plt.show()
```



شکل ۱-۱۰ رسم ویژگی‌ها به صورت دو تایی

با استفاده از دستورات زیر نحوه توزیع براساس هر ویژگی را نشان می‌دهیم.

```
plt.figure(figsize=(18, 9))
for i, feature in enumerate(feature_names):
    plt.subplot(2, 2, i + 1)
    sns.histplot(data=df, x=feature, hue='species', kde=True)
    plt.title(f'{feature} Distribution')
plt.tight_layout()
plt.show()
```



شکل ۱-۱۱ نمایش نحوه توزیع داده‌ها

دیدی که ما از این بررسی‌ها می‌گیریم این می‌باشد که نتایج ترکیب هر یک از ویژگی‌ها با یکدیگر متفاوت می‌باشد. به این معنی که کاهش بعد و استخراج ویژگی می‌تواند همبستگی بین داده‌ها را از بین ببریم. یعنی در واقع توزیع داده‌ها را ببینیم نوعی حس همبستگی را به ما می‌دهد. یعنی دو ویژگی برای ما اطلاعات یکسانی دارد و با کاهش بعد می‌توانیم ویژگی بهینه را استخراج کنیم و مانع از overfitting شبکه ما گردد.

ما با دیتاست‌هایی سر و کار داریک که دارای ویژگی‌های زیادی می‌باشند که کاهش ابعاد برای آسان و بهینه کردن روش‌های مورد استفاده می‌گردد. کاهش بعد به دو منظور صورت می‌گردد: نخست برای نمایش داده‌ها صورت می‌گردد، دومی برای آسان کردن تحلیل فرآیند هوش مصنوعی می‌گردد. اصولاً بر دو پایه می‌باشد: انتخاب ویژگی و استخراج ویژگی.

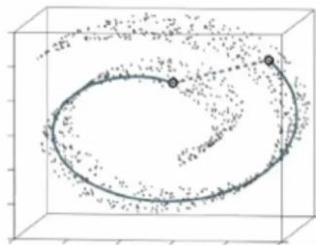
سه دسته کلی استخراج ویژگی را می‌توان چنین بیان کرد: t-SNE که برای نمایش داده‌ها مورد استفاده قرار می‌گیرد؛ PCA که هم برای نمایش داده و هم کاهش ابعاد مورد استفاده قرار می‌گیرد؛ LDA که صرفاً برای تحلیل مورد استفاده قرار می‌گیرد و برای نمایش کاربردی ندارد.

(t-SNE) یک الگوریتم کاهش ابعاد غیرخطی است که برای اکتشاف و بصری‌سازی داده‌های با ابعاد بالا استفاده می‌شود. این الگوریتم توسط لارنس ون در ماتن Laurens van der Maaten) و جفری هینتون Geoffrey Hinton توسعه داده شده است و به طور خاص برای کاهش ابعاد داده‌هایی که ساختارهای پیچیده‌ای دارند بسیار مفید است.

کاربرانی که با تحلیل مولفه اساسی (PCA-Principal Components Analysis) آشنایی دارند ممکن است چنین پرسشی را طرح کنند که بین این الگوریتم و t-SNE چه تفاوتی هست. اولین موردی که می‌توان به آن اشاره کرد این است که PCA در سال ۱۹۹۳ ساخته شد، در حالیکه t-SNE در سال ۲۰۰۸ ظهرور پیدا کرد. از سال ۱۹۹۳ به بعد چیزهای زیادی در دنیای علم داده تغییر کرده که مهمترین آن‌ها توان محاسباتی (و ابزارهای محاسباتی) و اندازه داده‌ها محسوب می‌شود.

دومین مساله این است که PCA یک روش کاهش ابعاد خطی است که در تلاش برای بیشینه کردن واریانس و حفظ فاصله‌های زیاد دوتایی‌ها از یکدیگر است. به عبارت دیگر، چیزهایی که با یکدیگر متفاوت هستند به صورت دور از هم به پایان می‌رسند. این امر می‌تواند منجر به بصری‌سازی ضعیف به ویژه هنگام کار با ساختارهای غیرخطی خمینه می‌شود. ساختار خمینه می‌تواند به صورت یک شکل جغرافیایی مثلاً استوانه، کره، منحنی و دیگر موارد باشد.

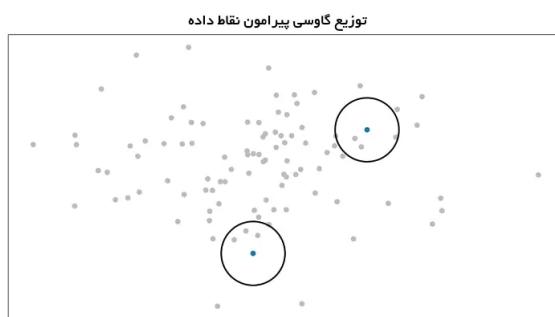
نظر به اینکه PCA به حفظ فاصله‌های دوتایی‌های بزرگ برای بیشینه کردن واریانس می‌پردازد، t-SNE-ها حفظ فاصله‌های کوچک دوتایی‌ها یا شباهت محلی از PCA متمایز می‌شود. لورنزو تفاوت PCA و t-SNE را با استفاده از مجموعه داده Swiss Roll به خوبی در شکل زیر نشان داده. می‌توان مشاهده کرد که به دلیل غیرخطی بودن این مجموعه داده (خمینه) و حفظ فواصل بزرگ، PCA ساختار داده‌ها را به اشتباه حفظ می‌کند.



شکل ۱-۱۲ مجموعه داده Swiss Roll. حفظ فاصله کوچک با t-SNE (خط توپر) در مقایسه با PCA بیشینه‌سازی واریانس.

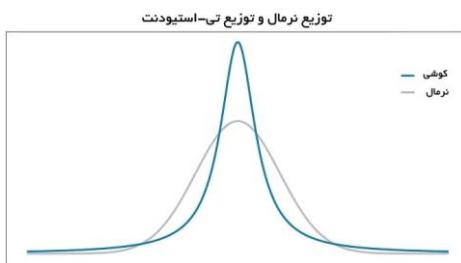
عملکرد الگوریتم t-SNE چنین تشریح می‌شود که این الگوریتم یک سنجه مشابهت را بین نمونه‌ها در داده‌های ابعاد بالا و در فضای ابعاد کم محاسبه و سپس برای بهینه‌سازی این دو سنجه مشابهت با استفاده از یکتابع هزینه تلاش می‌کند. آنچه بیان شد را می‌توان به سه گام اولیه که در زیر بیان شده‌اند شکست.

۱. در گام اول، مشابهت بین نقاط در فضای ابعاد بالا اندازه‌گیری می‌شود. برای درک بهتر موضوع، می‌توان دسته‌ای از نقاط داده پراکنده شده در یک فضای دو بعدی را مانند آنچه در شکل زیر آمده، در نظر گرفت. برای هر نقطه داده (x_i) توزیع گاووسی حول محور آن نقطه توسط کاربر متمرکز می‌شود. سپس، چگالی همه نقاط (x_j) تحت آن توزیع گاووسی محاسبه می‌شود. پس از آن، بازبینجارسازی (renormalize) برای همه نقاط داده انجام می‌شود. این امر یک مجموعه از احتمالات (P_{ij}) برای کلیه نقاط داده را به دست می‌دهد. این احتمالات متناسب با مشابهت‌ها هستند. این در واقع بدان معناست که اگر نقطه داده x_1 و x_2 مقادیر یکسانی تحت دایره گاووسی داشته باشند، نسبت‌ها و مشابهت‌های آن‌ها مساوی و بنابراین مشابهت‌های محلی در ساختار فضای ابعاد بالا فراهم است. توزیع یا دایره گاووسی با استفاده از آنچه سرگشتگی (perplexity) نامیده می‌شود، قابل دستکاری کردن محسوب می‌شود و واریانس توزیع (اندازه دایره) و اساساً تعداد نزدیک‌ترین همسایه‌ها را تحت تاثیر قرار می‌دهد. دامنه نرمال برای سرگشتگی بین ۵ و ۵۰ است.



شکل ۱-۱۳ سنجش مشابهت دوتاوی‌ها در فضای ابعاد بالا

۲. گام ۲ مشابه گام ۱ است، اما به جای توزیع گاووسی، توزیع تی-استودینت (Student's t-distribution) با یک درجه آزادی مورد استفاده قرار می‌گیرد که با عنوان توزیع کوشی (Cauchy distribution) نیز شناخته می‌شود (به عبارت دیگر، هنگامی که درجه آزادی در توزیع تی-استودینت برابر با یک باشد به توزیع کوشی تبدیل می‌شود). این کار دومین مجموعه از احتمالات (Q_{ij}) را در فضای ابعاد پایین به دست می‌دهد. چنانچه از تصویر زیر مشهود است، توزیع تی-استودینت دارای دم سنگین‌تری نسبت به توزیع نرمال است. دم سنگین امکان مدل‌سازی بهتر فواصل طولانی‌تر از هم را فراهم می‌کند.



شکل ۱-۱۴ توزيع نرمال و توزيع تي-استيودنت

۳. گام آخر اين است که اين مجموعه از احتمالات از فضای ابعاد پايانی (Q_{ij}) آن‌هايی که در فضای ابعاد بالاي (P_{ij}) قرار دارند را به بهترین شكل ممکن منعکس کنند. خواسته آن است که ساختار هر دو نقشه مشابه باشد. تفاوت بين توزيع‌های احتمال فضای دو بعدی با استفاده از معيار واگرایي کولبک-لبلر (Kullback-Liebler divergence | KL) قابل محاسبه است. در اين مطلب به مبحث KL به طور جزئي پرداخته نخواهد شد و تنها به بيان اين نكته که يك رو يکرد نامتقارن است که به طور موثر مقادير p_{ij} و q_{ij} را مقایسه می‌کند اكتفا خواهد شد. در نهايتي، از گراديان نزولي برای کاهشتابع هزينه KL استفاده می‌شود.

برای جمع‌بندی می‌توان بيان نمود که از اين تبديل برای يافتن شباهت الگوي موجود بین داده‌ها مورد استفاده قرار می‌گيرد و نمونه‌ها مشابه را نزديك، نمونه‌های غيرمشابه را دور نگه دارد. برای يافتن فاصله بین دو داده از رابطه گووسین کرنل استفاده می‌کنيم که رابطه آن به صورت زير است. در مخرج آن واريانس وجود دارد که باعث از بين رفتن پراكندگي و ديد يكسان به داده‌ها با مقیاس‌های گوناگون می‌گردد.

$$K(x, x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right)$$

سپس با استفاده از روابط زير احتمالات شباهت دو نمونه را به دست می‌آورد و احتمالات يكسان بودن ويزگي دو داده در فضای ويزگي را محاسبه می‌کند. به اين معنى که اگر دو نمونه بهم شباهت داشته باشند، در فضای ويزگي نيز به يكديگر شباهت دارند.

$$p_{j|i} = \frac{\exp\left(-\frac{\|x_i - x'_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_i - x'_k\|^2}{2\sigma_i^2}\right)}$$

$$P = (p_{ij})_{i,j=1}^n, p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

$$Q = (q_{ij})_{i,j=1}^n, q_{ij} = \frac{(1+|y_i - y_j|^2)^{-1}}{\sum_{k \neq i} (1+|y_k - y_l|^2)^{-1}}$$

حال برای مشابه کردن اين دو احتمال از روابط زير استفاده شده است. از يك معيار واگرایي استفاده کرده است و ميزان آن را بهينه می‌کند. در نتيجه برای هر يك x درون نمونه واقعی يك y محاسبه می‌گردد و به اين گونه مدل t-SNE خود را آموزش می‌دهيم.

$$KL(P|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

$$\frac{\partial KL(P|Q)}{\partial y_i} = 4 \sum_{j=1}^n (p_{ij} - q_{ij}) (y_i - y_j) (1 + |y_i - y_j|^2)^{-1}$$

کاهش ابعاد: t-SNE داده‌های با ابعاد بالا را به ابعاد پايانی تر (معمولًا ۲ یا ۳ بعد) کاهش می‌دهد.

حفظ همسایگی: هدف اصلی t-SNE حفظ همسایگی نسبی داده‌ها است، به این معنی که در فضای اصلی به یکدیگر نزدیک هستند، در فضای کاهش یافته نیز نزدیک به هم باقی می‌مانند.

توزیع تصادفی: t-SNE از توزیع‌های تصادفی (احتمالاتی) برای نگاشت نقاط استفاده می‌کند که به حفظ ساختار محلی داده‌ها کمک می‌کند.

اکنون که روش کار t-SNE شرح داده شد، مختصری به موارد استفاده از آن پرداخته می‌شود.

اکتشاف داده‌ها: t-SNE برای بصری‌سازی و اکتشاف داده‌های با ابعاد بالا بسیار مفید است، به خصوص در داده‌هایی که به طور طبیعی خوشه‌بندی شده‌اند.

پیش‌پردازش داده: می‌تواند به عنوان یک مرحله پیش‌پردازش قبل از اعمال الگوریتم‌های دیگر یادگیری ماشین استفاده شود.

t-SNE به دلیل قابلیت حفظ همسایگی و توزیع ساختارهای محلی، به خصوص برای داده‌های پیچیده با ابعاد بالا بسیار کاربردی است و در بسیاری از پروژه‌های تحلیل داده و یادگیری ماشین به کار می‌رود.

در ابتدا با استفاده از کتابخانه درون سایکیتلرن داده‌ها را به دو بخش آموزش و آزمایش (۲۰٪ از کل نمونه‌ها) تقسیم می‌کنیم و ابعاد هر کدام را نشان می‌دهیم.

```
# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=76)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
((120, 4), (120,), (30, 4), (30,))
```

شكل ۱-۱۵ ابعاد داده‌های ورودی و خروجی آموزش و آزمایش

سپس برای ایجاد مدل t-SNE کتابخانه مورد نظر را فراخوانی می‌کنیم و تعداد بعد آن را دو در نظر می‌گیریم.

```
from sklearn.manifold import TSNE
```

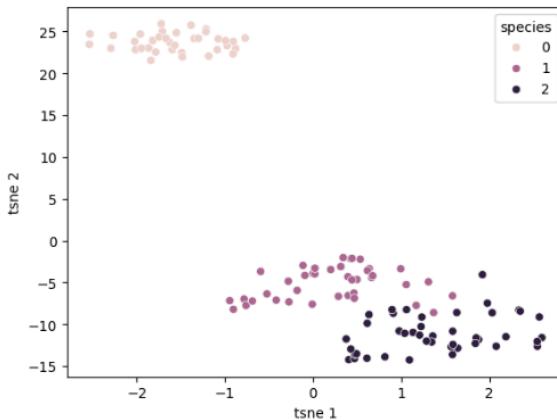
```
model = TSNE(n_components=2, random_state=76)
tsneData = model.fit_transform(X_train)
```

سپس با استفاده از داده‌های آموزش پیاده‌سازی می‌کنیم و داده‌های t-SNE را براساس خروجی نوع گل زنبق رسم می‌کنیم. همانطور که از نتایج پیداست کلاس ۰ به راحتی قابل جداسازی ولی کلاس ۱ و ۲ دارای پراکندگی داخل یکدیگر می‌باشد؛ در نتیجه کاهش ابعاد می‌تواند به ما کمک کند.

```
y_train1=pd.DataFrame(y_train, columns=['species'])
```

```
tsneDataFrame = pd.DataFrame(data = tsneData
, columns = ['tsne 1', 'tsne 2'])
tsneDataFrame.reset_index(drop=True, inplace=True)
y_train1.reset_index(drop=True, inplace=True)
y_train_df = pd.DataFrame(y_train1)
final_tsne_df = tsneDataFrame
final_tsne_df['species'] = y_train_df
ax = sns.scatterplot(x=final_tsne_df.iloc[:,0], y=final_tsne_df.iloc[:,1],
hue ='species',
data=final_tsne_df,
legend=True)
```

plt.show()



شکل ۱-۱۶ نمایش نمونه‌های دیتاست توسط t-SNE

با توجه به اطلاعات عددی و آماری به دست آمده از میانگین، واریانس و همبستگی ویژگی‌ها، و همچنین با استفاده از تصویری‌سازی t-SNE، می‌توان تحلیل کرد که کاهش ابعاد می‌تواند برای این دیتاست مفید باشد.
میانگین و واریانس: میانگین و واریانس نشان می‌دهد که داده‌ها دارای پراکندگی مناسبی هستند و ویژگی‌ها به خوبی توزیع شده‌اند.

همبستگی: ماتریس همبستگی نشان می‌دهد که برخی از ویژگی‌ها ممکن است به طور خطی با یکدیگر همبستگی داشته باشند که می‌تواند نشان‌دهنده وابستگی‌های خطی در داده‌ها باشد.

تصویری‌سازی t-SNE: نتایج t-SNE به خوبی نشان می‌دهد که داده‌ها در فضای دو بعدی به خوبی از هم جدا شده‌اند که بیانگر این است که کاهش ابعاد می‌تواند برای این دیتاست مناسب باشد و باعث فهم بهتر داده‌ها و کاهش پیچیدگی محاسباتی شود.

۱-۲- بخش ب

در این بخش هر دو الگوریتم PCA و LDA بررسی می‌شود. با توجه به اینکه داده‌های ما دارای برچسب می‌باشد باید از الگوریتم LDA که یک الگوریتم با ناظر است استفاده گردد.

از الگوریتم PCA برای یافتن یکسری مولفه‌های اصلی را برای ما می‌یابد؛ که در واقع یکسری access می‌باشند که در فضای ویژگی بر روی هم عمود می‌باشند. این مولفه‌ها در راستای بیشترین واریانس‌ها می‌باشد، به این معنی که این access‌ها در راستای بیشترین واریانس ممکن و بر روی هم عمود می‌باشند انتخاب می‌گردد. از نظر PCA ویژگی‌هایی مهم هستند که دارای بیشترین واریانس ممکن می‌باشد و بیشترین اطلاعات را به ما می‌دهد. در واقع ویژگی دوم ویژگی است که بر ویژگی اول عمود و همچنین در راستای بیشترین واریانس ممکن باشد؛ ویژگی سوم ویژگی است که بر دو ویژگی دیگر عمود است و در راستای بیشترین واریانس ممکن باشد. این تکرار تا زمان رسیدن به ابعاد اصلی ویژگی‌ها ادامه پیدا می‌کند.

می‌توان بیان کرد که این یک تبدیل خطی است که با یک چرخش و یک scale به دست می‌آید. در واقع می‌توان گفت ویژگی که دارای بیشترین واریانس می‌باشد دارای اطلاعات زیادی می‌باشد. دیتاهایی که دارای واریانس کمی می‌باشد، در واقع برای تک تک نمونه‌ها به مانند یک مقدار dc می‌باشد و تاثیری نخواهد داشت. می‌توان این الگوریتم را در چهار مرحله پیاده سازی نمود:

۱. استانداردسازی: با استفاده از رابطه زیر میانگین داده‌ها را صفر و واریانس را یک می‌کنیم.

$$Z = \frac{x - \mu}{\sigma}$$

۲. ماتریس کواریانس: ماتریس کواریانس داده‌های استاندارد را به دست می‌آوریم.

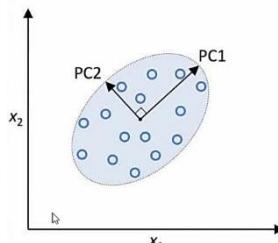
$$\text{cov}(x_1, x_2) = \frac{\sum_{i=1}^n (x_{1i} - \bar{x}_1)(x_{2i} - \bar{x}_2)}{n-1}$$

۳. مقادیر ویژه و بردار ویژه: مقادیر ویژه و بردار ویژه ماتریس کواریانس گام قبلی را محاسبه می‌کنیم.

$$AX - \lambda X = 0 \rightarrow (A - \lambda I)X = 0 \rightarrow |A - \lambda I| = 0$$

۴. مرتب‌سازی: در آخر نیز مقادیر ویژه و بردار ویژه را مرتب می‌کنیم. بردار ویژه که دارای مقدار ویژه بیشتری می‌باشد، ویژگی مهم می‌باشد و در راستای بیشترین واریانس قرار دارد. بردار ویژه دوم که دارای بیشترین واریانس می‌باشد ویژگی دوم است که بر ویژگی اول عمود و همچنین در راستای بیشترین واریانس ممکن می‌باشد. این گام را تکرار می‌کنیم. با توجه به اینکه می‌خواهیم ابعاد را چه میزان کاهش دهیم، به آن تعداد بردار ویژه را در نظر می‌گیریم.

در شکل زیر یک نمایش از PCA دو بعد را نشان می‌دهیم.



شکل ۱-۱۷ نمایش PCA برای دو بعد

با توجه به توضیحات ذکر شده در نتیجه با فراخوانی کتابخانه‌های مورد نیاز داده‌ها را استاندارد می‌کنیم.

from sklearn.preprocessing import StandardScaler

#scaling and centering the data

sc = StandardScaler()

X_train_scaled = sc.fit_transform(X_train)

X_test_scaled = sc.transform(X_test)

حال باید مقدار ماتریس کواریانس را حساب کنیم. از آنجایی که داده‌ها استاندار شده‌اند و دارای میانگین صفر و واریانس یک می‌باشند، می‌توانیم داده‌ها را در هم ضرب کنیم. سپس مقادیر ویژه را به دست می‌آوریم و مقادیر آن را به صورت نزولی مرتب می‌کنیم. در نهایت می‌توانیم براساس مقادیر مرتب شده تعداد ویژگی مدنظر را در نظر بگیریم. یک مفهوم مهم به نام variance explained وجود دارد. این مفهوم بیان می‌کند که این مولفه‌ی اصلی چه میزان از اطلاعات کل دیتاست را شامل می‌گردد. رابطه‌ی آن به صورت زیر می‌باشد.

$$\text{variance explained} = \frac{\lambda_i}{\sum \lambda_j}$$

با توجه به توضیحات بالا کد زیر مورد استفاده قرار می‌گیرید. همانطور که از نتایج مشخص است با در نظر گرفتن دو مولفه‌ی اصلی می‌توانیم به اطلاعات کافی دست بیابیم.

calculate covariance matrix, eigenvalues and eigenvectors

cov_mat = np.cov(X_train_scaled.T)

eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)

```

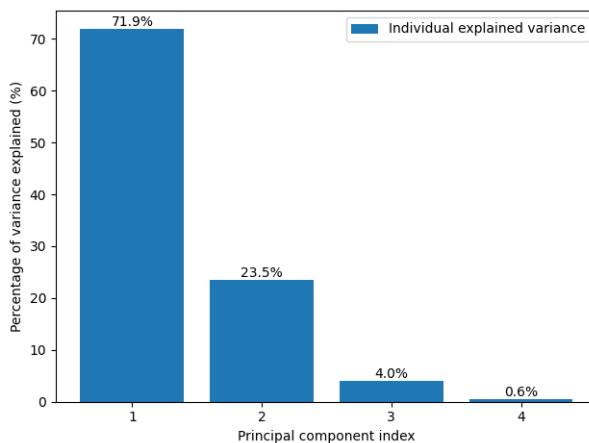
exp_var = []
# Sort the eigenvalues in descending order eigen_vals = np.sort(eigen_vals) [::-1]
for i in eigen_vals:
    var = (i / np.sum (eigen_vals)) * 100
    exp_var.append(var)

bar = plt.bar(range(1, 5), exp_var, align='center',
              label='Individual explained variance')

# Adding data labels to the top of bars
for i, bar in enumerate(bar):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height(), f'{exp_var[i]:.1f}%',
             ha='center', va='bottom')

plt.ylabel('Percentage of variance explained (%)')
plt.xlabel('Principal component index')
plt.xticks(ticks=list (range(1, 5)))
plt.legend (loc='best')
plt.tight_layout()

```



شکل ۱-۱۸ نمودار مربوط به variance explained

حال با در نظر گرفتن دو مولفه اصلی PCA را ایجاد می‌کنیم. در ابتدا کتابخانه مورد نظر را فراخوانی می‌کنیم و یک شی از آن را ایجاد می‌کنیم. با استفاده از داده‌های آموزش fit_transform صورت می‌گیرد و با استفاده از داده آزمون transform صورت می‌گیرد. و در نهایت نتایج را نمایش می‌دهیم. در مقایسه با t-SNE می‌توانیم بگوییم داده‌ها بهم منسجم‌تر شده‌اند و مرزی قابل رسم برای آنها تقریباً موجود می‌باشد.

from sklearn.decomposition import PCA

```

pca = PCA (n_components=2)
principalComponents = pca.fit_transform(X_train_scaled)
pca_test = pca.transform(X_test_scaled)
pca_df = pd.DataFrame (data = principalComponents
                       , columns = ['principal component 1', 'principal component 2'])

pca_df.reset_index(drop=True, inplace=True)
y_train2=pd.DataFrame(y_train, columns=['species'])
y_train2.reset_index (drop=True, inplace=True)
y_train2 = pd.DataFrame(y_train2)

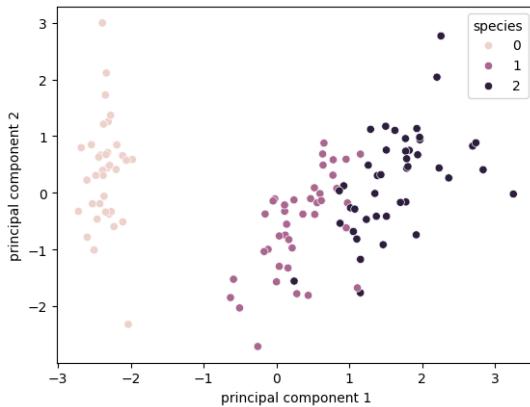
```

```

final_pca_df = pca_df
final_pca_df['species'] = y_train2

ax = sns.scatterplot(x = final_pca_df.iloc[:,0], y = final_pca_df.iloc[:,1],
                     hue = 'species',
                     data=final_pca_df,
                     legend=True)
plt.show()

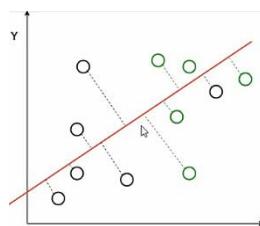
```



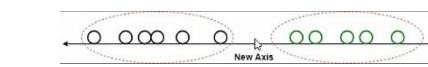
شکل ۱-۱۹ نمایش نمونه‌های دیتاست توسط PCA

برای جمع بندی می‌توان بیان نمود که PCA برای دیتاست با تعداد بسیار يالا مناسب می‌باشد. ویژگی‌های محاسبه شده به دلیل اینکه بهینه می‌باشند، از هم مستقل می‌باشند که این موضوع از overfitting جلوگیری می‌کند. تأکید بر روی مولفه‌های که بیشترین واریانس را دارد سبب کاهش نویز می‌گردد. ضعف این الگوریتم در عدم تفسیرپذیری است. نحوه داده‌ها را به صورت خطی فرض می‌کند. و بسیار حساس می‌باشد؛ به همین منظور استانداردسازی صورت می‌گیرد. مورد دیگر تاثیرپذیری از داده‌ها outlayer است؛ وقتی میزان داده‌های outlayer زیاد است باید در حذف این داده‌ها صورت گیرد سپس از PCA استفاده گردد. این الگوریتم در حالی که ابعاد داده‌ها زیاد، شبه خطی، برای نمایش بصری و زمانی که داده‌ها ارتباط خطی دارند، کاربرد دارد.

الگوریتم LDA برخلاف دو الگوریتم قبلی که در آنها برچسب مهم نبود، برای کلاس‌بندی استفاده می‌گردد و به صورت با ناظر می‌باشد. این الگوریتم ویژگی‌هایی را به دست می‌آور که واریانس‌ها کم و میانگین آنها زیاد باشد و بر این اساس محورهای خود را به ما معرفی می‌کند.



شکل ۱-۲۰ کاهش بعد از کاهش بعد با استفاده از LDA



شکل ۱-۲۱ نمونه‌های کاهش بعد یافته شده توسط LDA

روابط مورد استفاده در زیر نمایش داده شده است. این روابط با فرض تصویر کردن بر روی آن خط میانگین و واریانس را محاسبه می‌کند. با بهینه کردن به آن access مورد نظر در هر تعداد بعد مورد نظر دست می‌یابیم.

$$\widetilde{\mu_1} = \frac{1}{n_1} \sum_{x_i \in c_1}^{n_1} v^T x_i = v^T \mu_1$$

$$\widetilde{\mu_2} = v^T \mu_2$$

$$\widetilde{s_1^2} = \sum_{y_i \in c_1} (y_i - \mu_1)^2$$

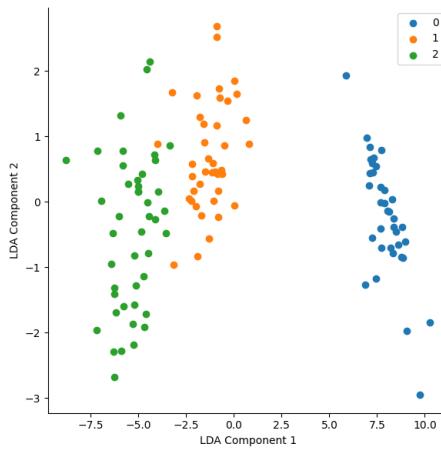
$$\widetilde{s_2^2} = \sum_{y_i \in c_1} (y_i - \mu_2)^2$$

$$J(v) = \frac{|\widetilde{\mu_1} - \widetilde{\mu_2}|}{\sqrt{\widetilde{s_1^2} + \widetilde{s_2^2}}} = \frac{v^T s_l v}{v^T s_w v}$$

کار LDA یافتن هر ابرصفحه‌ای است که در آن واریانس کلاس‌ها کم و میانگین زیاد می‌باشد را برای ما باید. برای پیاده سازی ما در ابتدا کتابخانه مدنظر را فراخوانی می‌کنیم و یک شی از آن ایجاد می‌کنیم. در نهایت داده‌های کاهش بعد یافته را رسم می‌کنیم.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

```
lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)
tmp_Df = pd.DataFrame(X_train_lda, columns=['LDA Component 1', 'LDA Component 2'])
tmp_Df['Class'] = y_train
sns.FacetGrid(tmp_Df, hue ="Class",
              height=6).map(plt.scatter,
                            'LDA Component 1',
                            'LDA Component 2')
plt.legend (loc='upper right')
```



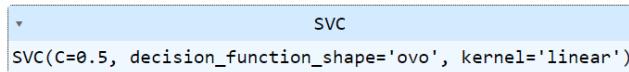
شکل ۱-۲۲ نمایش نمونه‌های دیتاست بعد از کاهش بُعد توسط LDA

این الگوریتم به خاطر دیدی که از کلاس بندی دارد، بهتر داده‌ها را تفکیک کرده است. از مزیتهای این روش می‌توان به افزایش قابلیت ماقزیم سازی برای جدا کردن کلاس‌ها اشاره نمود. این روش برای کلاس‌بندی کاربرد دارد. از معایب این روش می‌توان به حساسیت به داده‌های outlier است. در این روش فرض می‌گردد که توزیع داده‌ها در همه‌ی کلاس‌ها نرمال می‌باشد. داده‌های آموزش برای اینکه جداسازی را ماقزیم کنیم، لازم می‌باشد.

از این روش برای کلاس‌بندی، حفظ اطلاعات، زمانی که می‌دانیم توزیع داده‌ها نرمال می‌باشد استفاده می‌گردد. این الگوریتم برای داده‌های باناظر کاربرد دارد. در ادامه از داده‌های مربوط به کاهش بعد یافته LDA استفاده می‌کنیم.

برای ایجاد یک ماشین بردار پشتیبان (SVM) در ابتدا توسط کتابخانه فراخوانی شده SVC که برای کلاس‌بندی کاربرد دارد، یک شی ایجاد می‌کنیم. هایپر پارامترهای آن را تعیین می‌کنیم. C پارامتر regularization می‌باشد، نوع کرنل را خطی انتخاب می‌کنیم و نوعتابع تصمیم‌گیری را از نوع one-vs-one انتخاب می‌کنیم. با استفاده از متدهای fit مدل را آموزش می‌دهیم.

```
clf = SVC(kernel='linear', C=0.5, decision_function_shape='ovo')
clf.fit(X_train_lda, y_train)
```



```
SVC
SVC(C=0.5, decision_function_shape='ovo', kernel='linear')
```

شکل ۱-۲۳ مدل آموزش دیده

با استفاده از دستورات زیر اندیس‌ها، تعداد بردارهای پشتیبان داخل هر کلاس و بردارهای پشتیبان را نمایش می‌دهیم.

```
print("Support vectors index:", clf.support_)
print("Number of support vectors:", clf.n_support_)
print("Support vectors:", clf.support_vectors_)

Support vectors index: [ 7 15 34 53 95 98 113 1 40 75 81 96]
Number of support vectors: [1 6 5]
Support vectors: [[ 5.85860892  1.93592944]
 [-4.01795696  0.88102549]
 [-2.21995638  0.01360671]
 [-2.35468514  0.0470736]
 [-3.25183186  1.66730049]
 [-3.1439927 -0.96625126]
 [ 0.80042796  0.87981569]
 [-3.57505658 -0.47313884]
 [-3.97885069  0.15117968]
 [-3.34808536  0.86333038]
 [-4.42044683  2.14104586]
 [-3.65993004 -0.13940437]]
```

شکل ۱-۲۴ مشخصات ماشین آموزش دیده شده

با استفاده از دستورات زیر داده‌های آزمون را پیش‌بینی می‌کنیم مربوط به کدام کلاس می‌باشد.

```
y_pred = clf.predict(X_test_lda)
```

با استفاده از دستورات زیر میزان دقیقیت عملکرد ماشین آموزش دیده شده را نمایش می‌دهیم. همانطور که در ماتریس درهم‌ریختگی نشان داده شده است، دو داده مربوط به کلاس ۱ به اشتباهی مربوط به کلاس ۲ تشخیص داده شده‌اند.

```
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
```

```
# Evaluate the classifier
```

```
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
# Visualize the results
```

```
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```

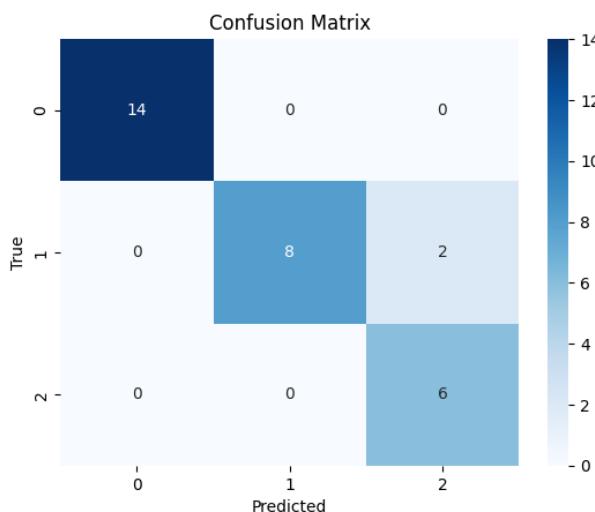
Accuracy: 0.9333333333333333
Classification Report:
precision    recall   f1-score   support
          0       1.00     1.00     1.00      14
          1       1.00     0.80     0.89      10
          2       0.75     1.00     0.86       6

   accuracy                           0.93      30
macro avg                           0.92      30
weighted avg                          0.95      30

Confusion Matrix:
[[14  0  0]
 [ 0  8  2]
 [ 0  0  6]]

```

شکل ۱-۲۵ ارزیابی عملکرد ماشین



شکل ۱-۲۶ ماتریس درهم ریختگی

با استفاده ازتابع زیر ناحیه‌های جداساز به همراه ناحیه حاشیه رسم شده است. در ابتدا یکسری نقاط در محدوده ویژگی اول و دوم ایجاد می‌کنیم سپس توسط دستور meshgrid از آرایه‌های یک بعدی، آرایه‌های دو بعدی بسازیم، که تمام جایگشت‌های مربوطه را در نظر بگیریم. سپس با استفاده از متدهای predict مقدار مربوط به ناحیه را شناسایی می‌کنیم. سپس توسط دستور contour خطوط را رسم می‌کنیم. توسط دستور coef_ وزن‌ها و intercept_ مقدار بایاس را محاسبه می‌کنیم و با استفاده از آن عرض از مبداء و شیب خط جداساز را محاسبه می‌کنیم. داده‌های SV را نیز توسط دستور support_vectors_ نمایش می‌دهیم.

```

def plot_decision_boundaries(X, y, model, title):
    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)

    colors = ['r', 'g', 'b']
    for i, color in zip(range(3), colors):
        idx = np.where(y == i)

```

```

plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i], edgecolors='k')
w = model.coef_[i]
b = model.intercept_[i]
# At the decision boundary, w0*x0 + w1*x1 + b = 0
# => x1 = -w0/w1 * x0 - b/w1
x0 = np.linspace(x_min, x_max, 200)
decision_boundary = (-(w[0]/w[1])* x0) - (b/w[1])

margin = 1/w[1]
#margin = 1/np.sqrt(np.sum(w**2))
gutter_up = decision_boundary + margin
gutter_down = decision_boundary - margin

plt.plot(x0, decision_boundary, '-', c=colors[i], linewidth=2)
plt.plot(x0, gutter_up, '--', c=colors[i], linewidth=2)
plt.plot(x0, gutter_down, '--', c=colors[i], linewidth=2)

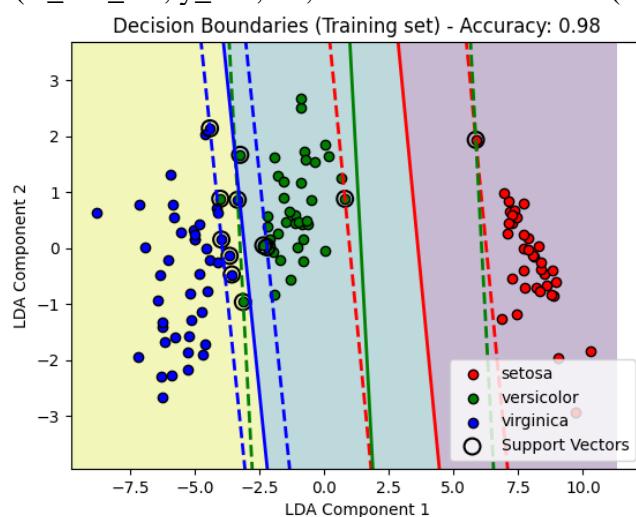
sv = model.support_vectors_
plt.scatter(sv[:, 0], sv[:, 1], facecolors='none', edgecolors='k', s=100, linewidths=1.5,
label='Support Vectors')

y_pred = model.predict(X)
accuracy = accuracy_score(y, y_pred)

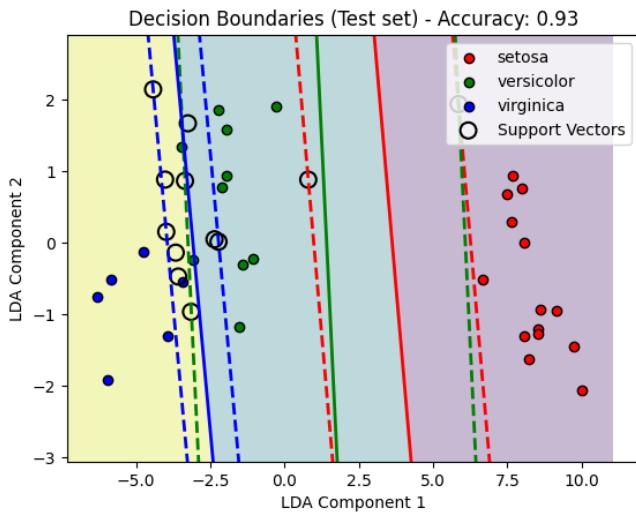
plt.ylim([y_min,y_max])
plt.title(f'{title} - Accuracy: {accuracy:.2f}')
plt.xlabel('LDA Component 1')
plt.ylabel('LDA Component 2')
plt.legend(loc='best')
plt.show()

```

plot_decision_boundaries(X_train_lda, y_train, clf, "Decision Boundaries (Training set)")
plot_decision_boundaries(X_test_lda, y_test, clf, "Decision Boundaries (Test set)")



شکل ۱-۲۷ خطوط جداکننده داده‌های آموزش



شکل ۱-۲۸ ۱- خطوط جدا کننده داده های آزمون

با استفاده از تابع زیر خطوط تصمیم را رسم می کنیم.

```
def visualize_multiclass_classification(X_train, y_train1, class_labels, trainset, model):
```

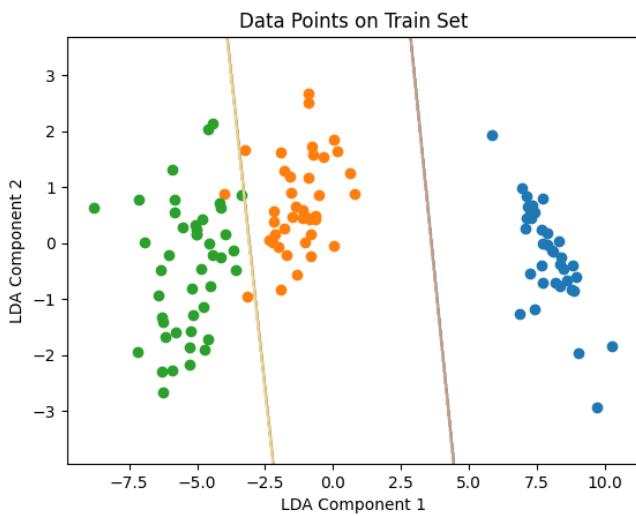
```
for i, target_name in enumerate(class_labels):
    plt.scatter(X_train[y_train1 == i, 0], X_train[y_train1 == i, 1], label=target_name)
```

```
h = .02 # step size in the mesh
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
k = np.arange(x_min, x_max, h)
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
x_test = np.c_[xx.ravel(), yy.ravel()]
Z = model.predict(x_test)
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
```

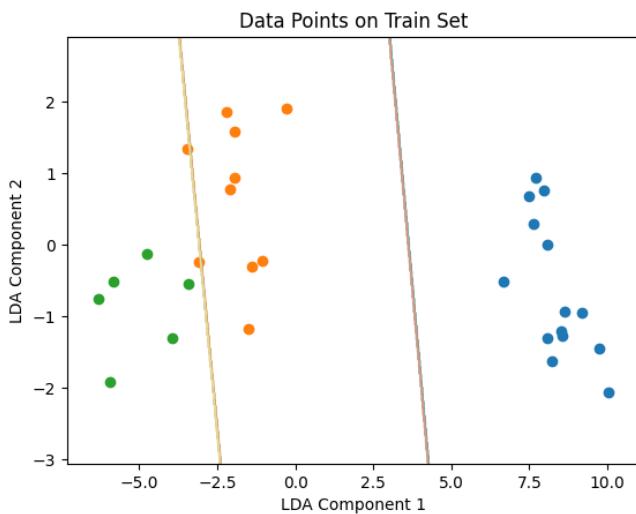
```
if trainset:
    plt.title('Data Points on Train Set')
else:
    plt.title('Data Points on Test Set')
```

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

```
visualize_multiclass_classification(X_train_lda, y_train, iris.target, 'True', clf)
visualize_multiclass_classification(X_test_lda, y_test, iris.target, 'False', clf)
```



شکل ۱-۲۹ خطوط تصمیم داده‌های آموزش



شکل ۱-۳۰ خطوط تصمیم داده آزمون

۱-۳- بخش ج

کرنل‌ها در واقع برای زمانی کاربرد دارد که داده‌ها در صفحه قابل جداسازی نباشد؛ رفتن به فضای با ابعاد بالا توسط کرنل امید به جداسازی می‌باشد. در این بخش با استفاده از کرنل چندجمله‌ای از درجه ۱ تا ۱۰ با استفاده از کتابخانه سایکیتلرن بررسی می‌کنیم. برای ذخیره تصاویر به صورت gif در ابتدا ماثول مورد نظر را بارگذاری می‌کنیم.

```
!pip install numpy matplotlib scikit-learn imageio
```

سپس یک پوشه برای ذخیره سازی تصاویر ایجاد می‌کنیم.

```
import os
```

```
# Define the folder name
folder_name = "SupportVector"
```

```
# Check if the folder already exists, and create it if not
if not os.path.exists(folder_name):
    os.makedirs(folder_name)
```

```
# Commented out IPython magic to ensure Python compatibility.  
%cd SupportVector
```

با استفاده از تابع زیر خطاهای جداساز و بردارهای پشتیبان را رسم می‌کنیم.

```
def plot_decision_boundary(X, y, model, degrees, trainset):  
    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)  
  
    colors = ['r', 'g', 'b']  
    for i, color in zip(range(3), colors):  
        idx = np.where(y == i)  
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i], edgecolors='k')  
  
    sv = model.support_vectors_  
    plt.scatter(sv[:, 0], sv[:, 1], facecolors='none', edgecolors='k', s=100, linewidths=1.5,  
               label='Support Vectors')  
    if trainset:  
        plt.title(f'Data Points on Train Set-SVM with Polynomial Kernel Degree {degrees}')  
    else:  
        plt.title(f'Data Points on Test Set-SVM with Polynomial Kernel Degree {degrees}')  
  
    plt.xlabel('LDA Component 1')  
    plt.ylabel('LDA Component 2')  
    plt.legend(loc='best')
```

سپس یک آرایه خالی برای ذخیره تصاویر ایجاد می‌کنیم. سپس داخل حلقه for ماشین بردار پشتیبان غیرخطی با درجات ۱ تا ۱۰ ایجاد می‌کنیم. میزان دقت عملکرد ماشین‌ها را نمایش می‌دهیم و در نهایت خطوط جداگانه به همراه بردارهای پشتیبان را نمایش می‌دهیم.

```
from sklearn.metrics import precision_score, recall_score, f1_score  
import imageio
```

```
degrees = range(1, 11)  
results = []  
images_train = []  
images_test = []  
images_conf = []  
# Train SVM with polynomial kernels of degree 1 to 10  
for degree in degrees:  
    clf1 = SVC(kernel='poly', degree=degree, random_state=76, decision_function_shape='ovo')  
    clf1.fit(X_train_lda, y_train)  
    y_pred = clf1.predict(X_test_lda)  
  
    accuracy = accuracy_score(y_test, y_pred)  
    report = classification_report(y_test, y_pred)
```

```

results.append((degree, accuracy, report))
print(f"Degree: {degree}\nAccuracy: {accuracy}\n{report}\n")

plt.figure()
plot_decision_boundary(X_train_lda, y_train, clf1, degree, True)

filename1 = f'svm_degree_{degree}_train_support_vector.png'
plt.savefig(filename1)
images_train.append(imageio.imread(filename1))
plt.close()

plt.figure()
plot_decision_boundary(X_test_lda, y_test, clf1, degree, False)

filename2 = f'svm_degree_{degree}_test_support_vector.png'
plt.savefig(filename2)
images_test.append(imageio.imread(filename2))
plt.close()

# Visualize the results
plt.figure()
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title(f'Confusion Matrix {degree}')
filename3 = f'svm_degree_{degree}_Confusion_Matrix.png'
plt.savefig(filename3)
images_conf.append(imageio.imread(filename3))
plt.close()

# Path to save the GIF
gif_path_train = './svm_poly_degrees_train_support_vector.gif'
imageio.mimsave(gif_path_train, images_train, duration=10)
# Display the path to the GIF
print(f"GI saved as {gif_path_train}")

# Path to save the GIF
gif_path_test = './svm_poly_degrees_test_support_vector.gif'
imageio.mimsave(gif_path_test, images_test, duration=10)
# Display the path to the GIF
print(f"GI saved as {gif_path_test}")

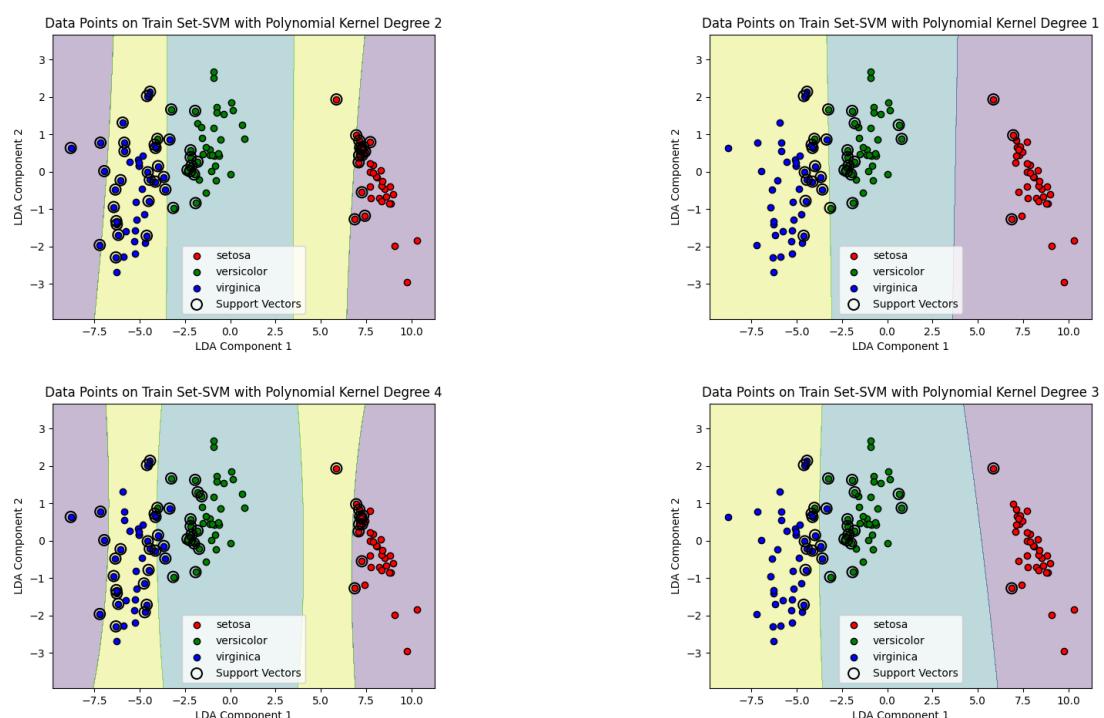
# Path to save the GIF
gif_path_conf = './svm_poly_degrees_Confusion_Matrix.gif'
imageio.mimsave(gif_path_conf, images_conf, duration=10)
# Display the path to the GIF
print(f"GI saved as {gif_path_conf}")

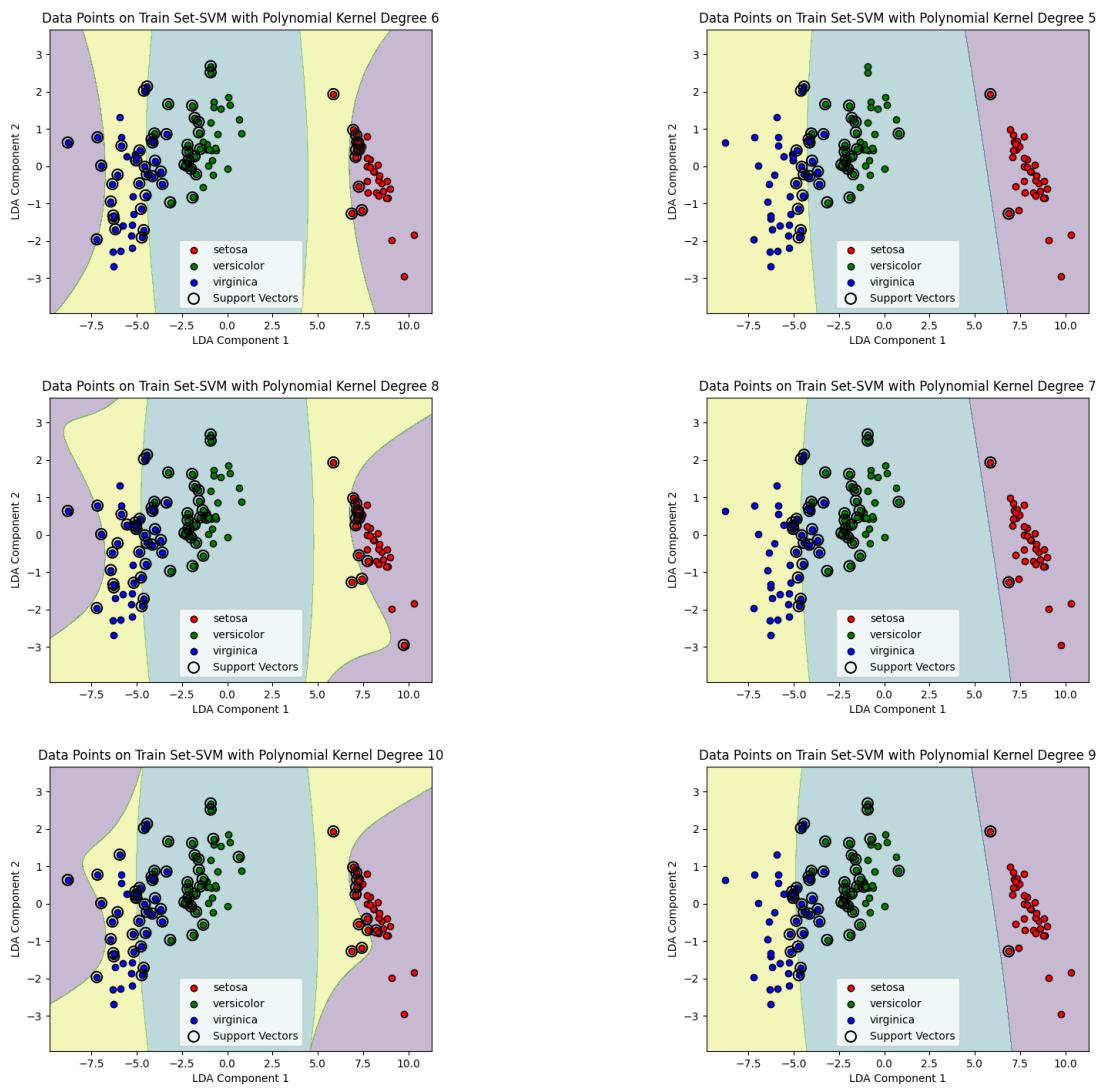
```

میزان دقیق عملکرد این روش در زیر نشان داده شده است. خطوط جداگانه به همراه بردارهای پشتیبان مربوط به داده‌های آموزش و آزمون را در زیر نمایش می‌دهیم

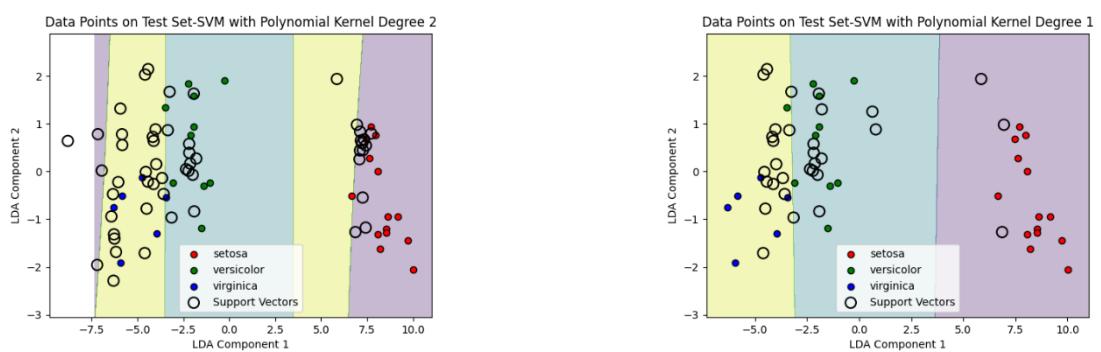
Degree: 2	Degree: 1
Accuracy: 0.9333333333333333	Accuracy: 0.9666666666666667
precision recall f1-score support	precision recall f1-score support
0 1.00 0.93 0.96 14	0 1.00 1.00 1.00 14
1 0.91 1.00 0.95 10	1 1.00 0.90 0.95 10
2 0.83 0.83 0.83 6	2 0.86 1.00 0.92 6
accuracy	accuracy
macro avg	macro avg
weighted avg	weighted avg
Degree: 4	Degree: 3
Accuracy: 0.9	Accuracy: 0.9666666666666667
precision recall f1-score support	precision recall f1-score support
0 1.00 0.93 0.96 14	0 1.00 1.00 1.00 14
1 0.83 1.00 0.91 10	1 0.91 1.00 0.95 10
2 0.80 0.67 0.73 6	2 1.00 0.83 0.91 6
accuracy	accuracy
macro avg	macro avg
weighted avg	weighted avg
Degree: 6	Degree: 5
Accuracy: 0.9	Accuracy: 0.9333333333333333
precision recall f1-score support	precision recall f1-score support
0 1.00 0.93 0.96 14	0 1.00 1.00 1.00 14
1 0.83 1.00 0.91 10	1 0.83 1.00 0.91 10
2 0.80 0.67 0.73 6	2 1.00 0.67 0.80 6
accuracy	accuracy
macro avg	macro avg
weighted avg	weighted avg
Degree: 8	Degree: 7
Accuracy: 0.8666666666666667	Accuracy: 0.9333333333333333
precision recall f1-score support	precision recall f1-score support
0 1.00 0.93 0.96 14	0 1.00 1.00 1.00 14
1 0.77 1.00 0.87 10	1 0.83 1.00 0.91 10
2 0.75 0.50 0.60 6	2 1.00 0.67 0.80 6
accuracy	accuracy
macro avg	macro avg
weighted avg	weighted avg
Degree: 10	Degree: 9
Accuracy: 0.8666666666666667	Accuracy: 0.9
precision recall f1-score support	precision recall f1-score support
0 1.00 0.93 0.96 14	0 1.00 1.00 1.00 14
1 0.77 1.00 0.87 10	1 0.77 1.00 0.87 10
2 0.75 0.50 0.60 6	2 1.00 0.50 0.67 6
accuracy	accuracy
macro avg	macro avg
weighted avg	weighted avg

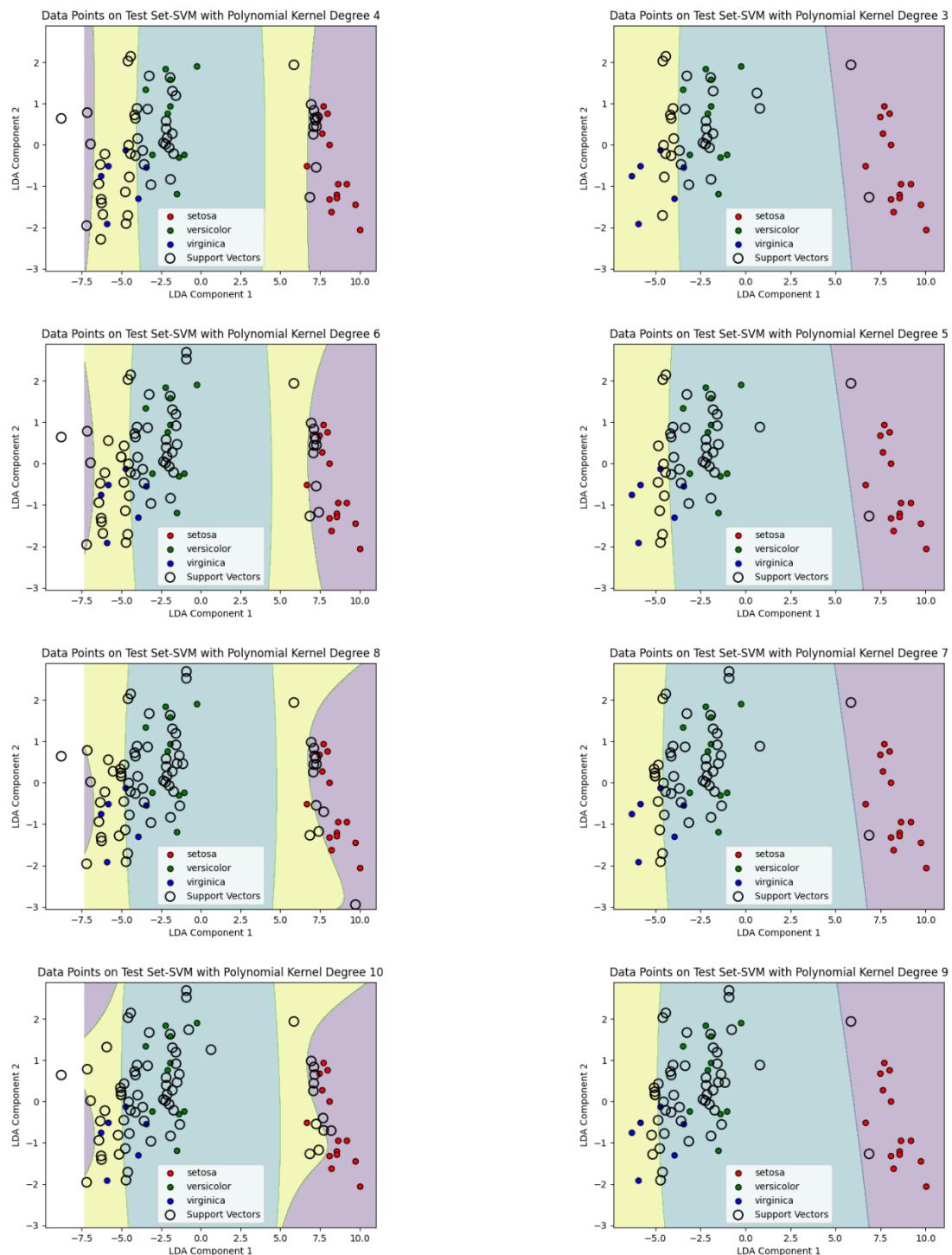
شکل ۱-۳۱ نمایش میزان دقیق مدل با درجهات ۱-۱۰





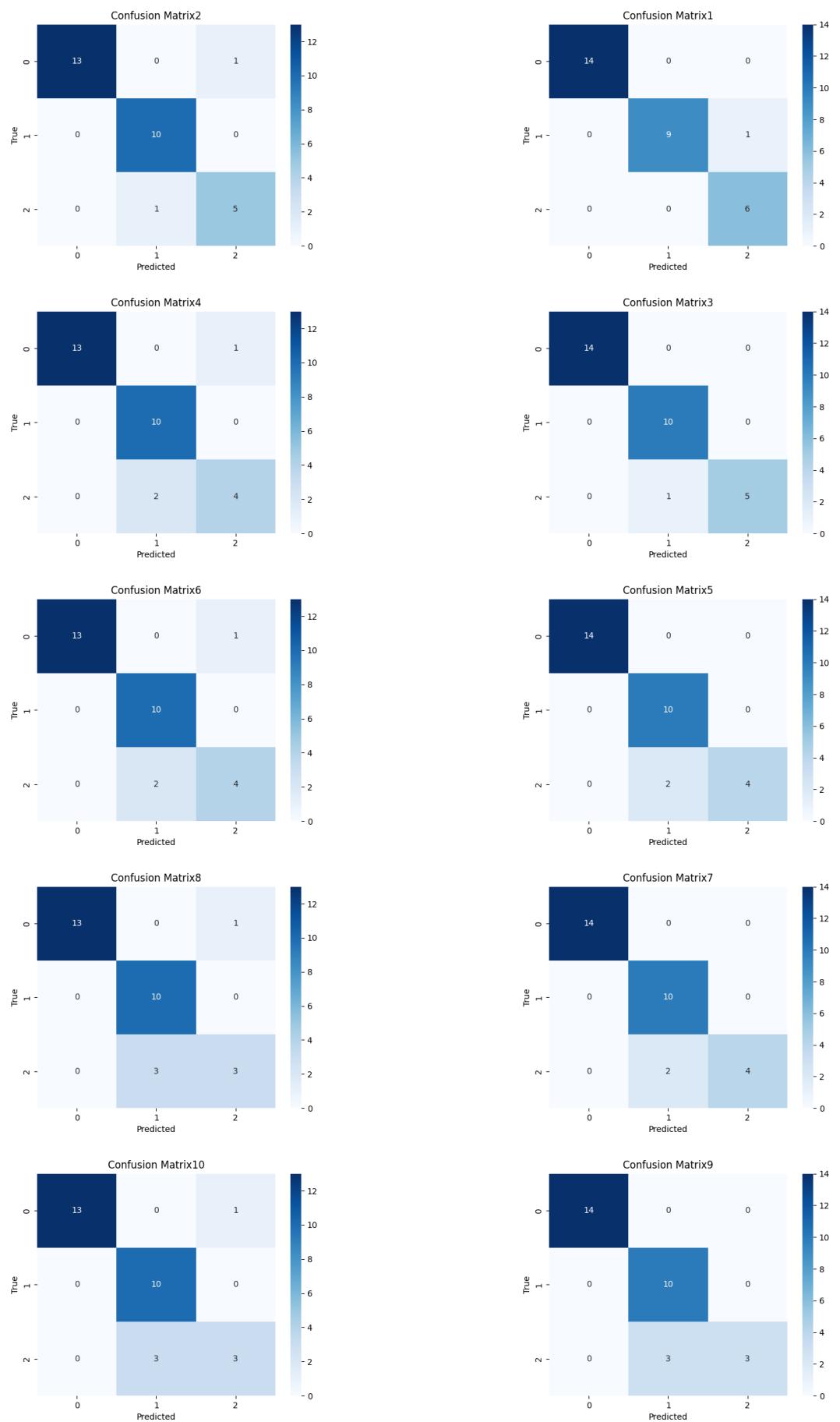
شکل ۱-۳۲ خطوط جداکننده و بردار پشتیبان داده‌های آموزش





شکل ۱-۳۳ خطوط جداکننده و بردار پشتیبان داده‌های آزمایش

در زیر ماتریس درهم‌ریختگی را نمایش می‌دهیم. همانطور که مشاهده می‌گردد مدل در مرتب بالاتر دچار مشکل می‌گردد و میزان دقیقت عملکرد آن کاهش می‌یابد.



شکل ۱-۳۴ ماتریس درهم ریختگی

محل ذخیره‌سازی gif‌های مربوطه در زیر نماش داده شده است. برای دسترسی راحت داخل درایو به آدرس زیر نیز قرار داده شده است.

```
GIF saved as ./svm_poly_degrees_train_support_vector.gif
GIF saved as ./svm_poly_degrees_test_support_vector.gif
GIF saved as ./svm_poly_degrees_Confusion_Matrix.gif
```

شکل ۱-۳۵ محل ذخیره‌سازی gif‌ها

https://drive.google.com/file/d/1tZ8yxbZbuTuNTNLYeaO4R0juK2CZEsQF/view?usp=drive_link
https://drive.google.com/file/d/1ESuHO_M6EPFfrTmTpAYzz7nSCwOjybQA/view?usp=drive_link

https://drive.google.com/file/d/1MSJex2Nr9zs8Y47OskLrgqXLh_MEAgoc/view?usp=drive_link با استفاده از تابع زیر خطوط تصمیم‌گیری رسم شده‌اند و در یک پوشه جداگانه ذخیره شده‌اند. خطوط تصمیم‌گیری مربوط به داده‌های آموزش و آزمون در زیر رسم شده است.

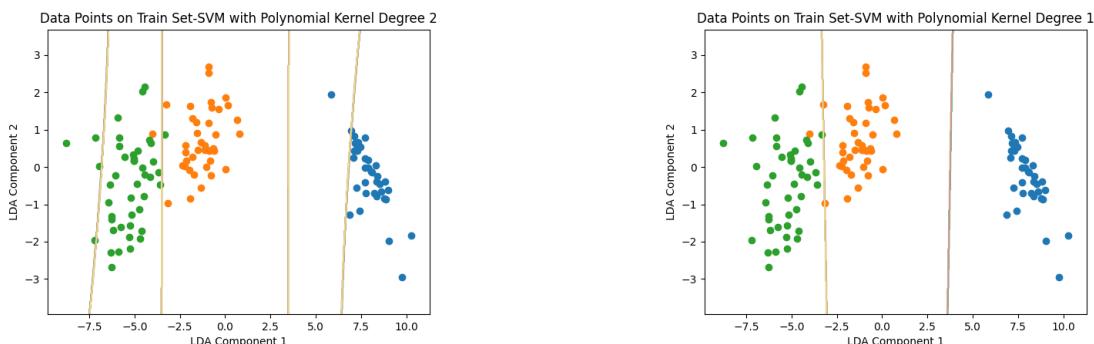
```
def visualize_multiclass(X_train, y_train1, class_labels, trainset, model, degrees):
```

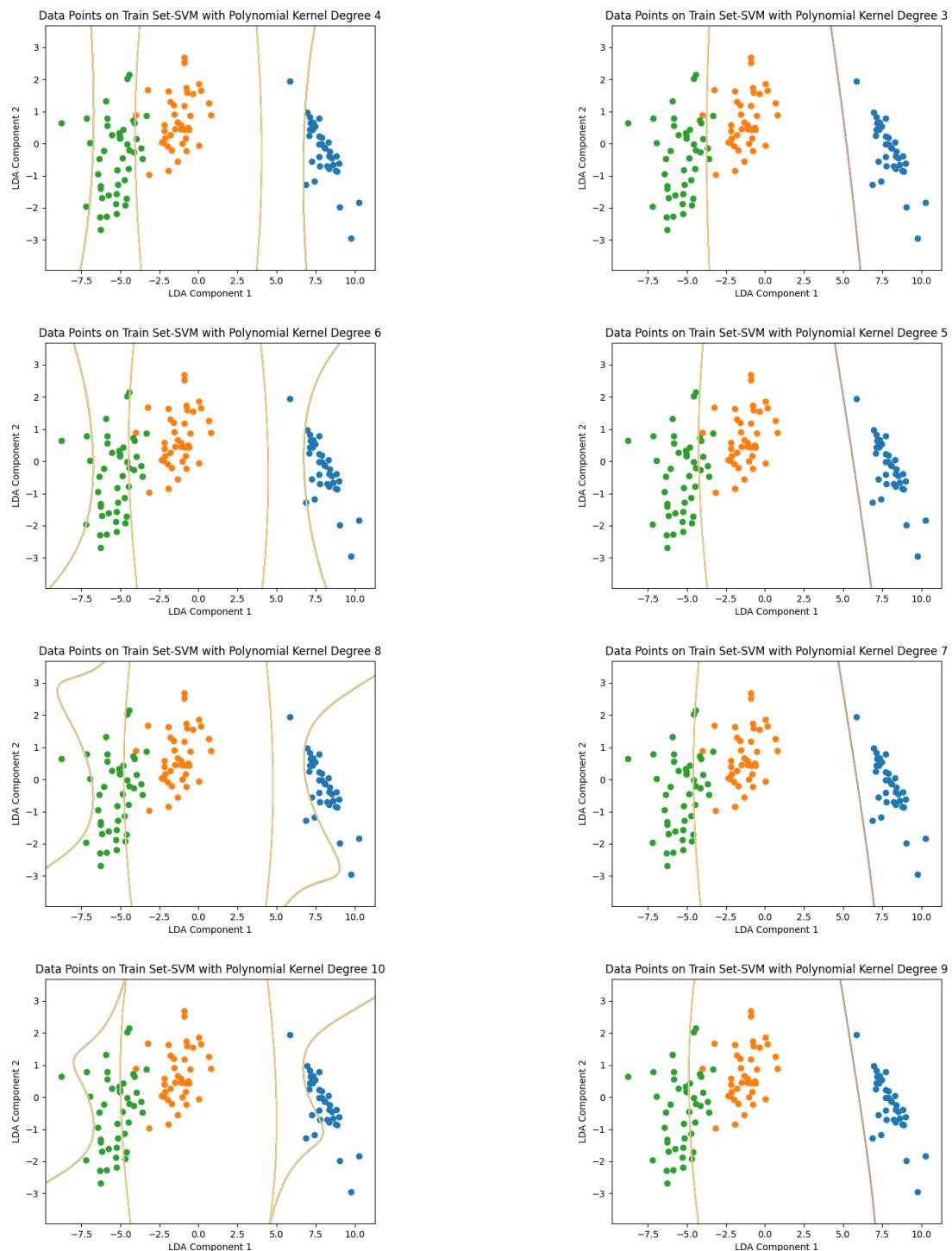
```
for i, target_name in enumerate(class_labels):
    plt.scatter(X_train[y_train1 == i, 0], X_train[y_train1 == i, 1], label=target_name)

h = .02 # step size in the mesh
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
k = np.arange(x_min, x_max, h)
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
x_test = np.c_[xx.ravel(), yy.ravel()]
Z = model.predict(x_test)
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

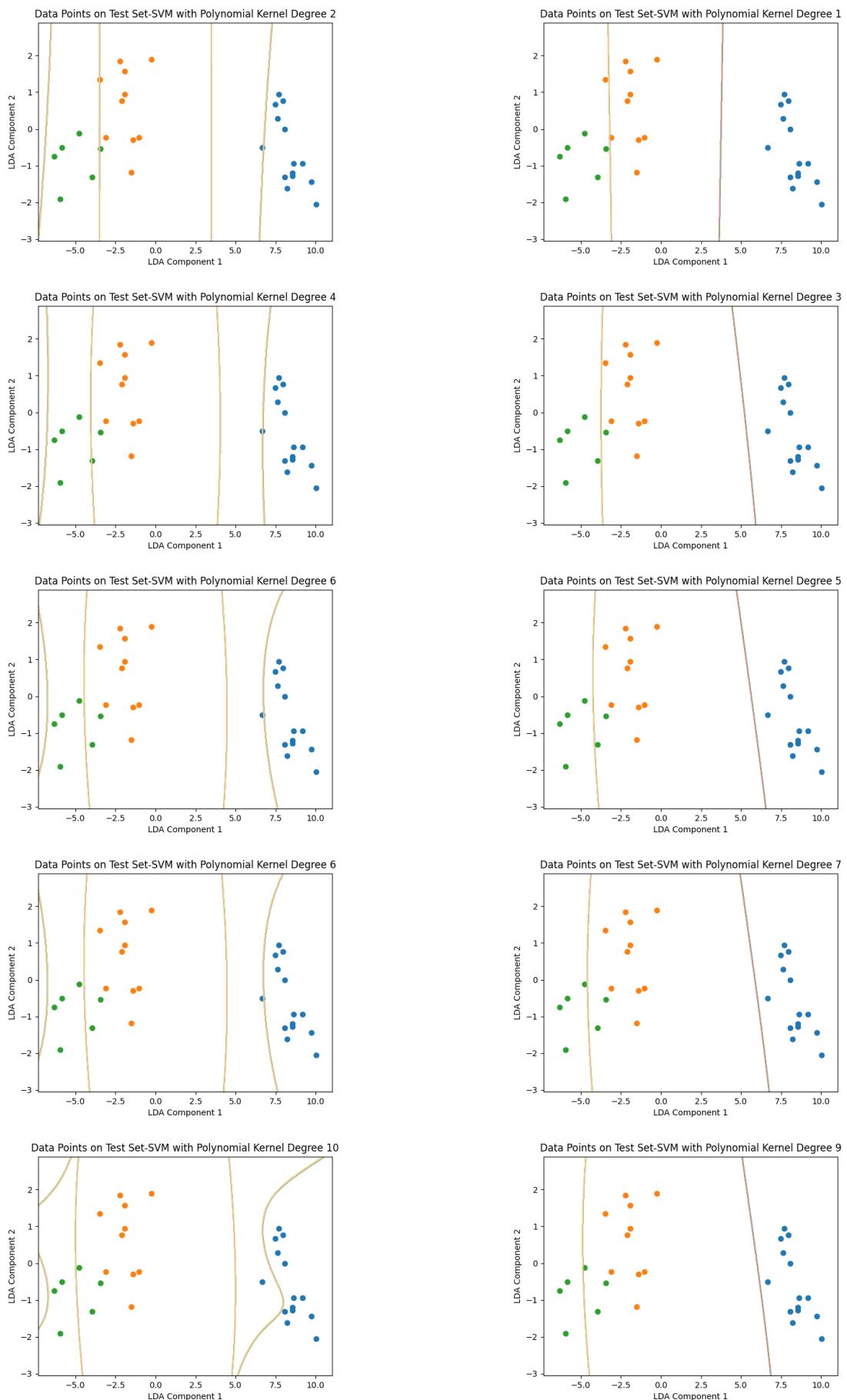
if trainset:
    plt.title(f'Data Points on Train Set-SVM with Polynomial Kernel Degree {degrees}')
else:
    plt.title(f'Data Points on Test Set-SVM with Polynomial Kernel Degree {degrees}')
```

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





شکل ۱-۳۶ خطوط تصمیم داده‌های آموزش



شکل ۱-۳۷ خطوط تصمیم داده‌های آموزش

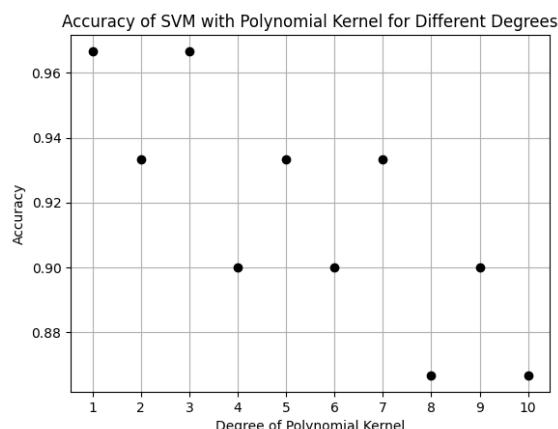
محل ذخیره‌سازی gif‌های مربوطه در زیر نماش داده شده است. برای دسترسی راحت داخل درایو به آدرس زیر نیز قرار داده شده است.

```
GIF saved as ./svm_poly_degrees_train.gif
GIF saved as ./svm_poly_degrees_test.gif
```

شکل ۱-۳۸ محل ذخیره‌سازی gif‌ها

https://drive.google.com/file/d/1iIoUt6cnkrdVFZZgB18vIbgeJFTiH6Md/view?usp=drive_link
https://drive.google.com/file/d/1uW4FXI0g1IFZmpGS-T0ckru-8I4RoWMx/view?usp=drive_link

میزان دقیق عملکرد ماشین در دجات مختلف در شکل زیر نمایش داده شده است.



شکل ۱-۳۹ دقیقیت الگوریتم با افزایش درجه

۱-۴- بخش ۵

در این بخش در ابتدا کتابخانه‌های مورد نظر را بارگذاری می‌کنیم. برای ایجاد تصاویر به صورت gif ماثول مورد نیاز را نصب می‌کنیم. سپس مجدداً دیتاست iris را بارگذاری و داده‌های ورودی و خروجی را تعیین می‌کنیم. سپس به صورت دیتا فرم تبدیل می‌کنیم. داده‌های آموزش و آزمون را جدا سازی می‌کنیم.

```
# Import necessary libraries
from sklearn import datasets
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import cvxopt
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
import seaborn as sns
import os
import imageio
!pip install numpy matplotlib scikit-learn imageio
# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```

feature_names = iris.feature_names

# Convert to DataFrame for easier manipulation
df = pd.DataFrame(X, columns=feature_names)
df['species'] = y

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=76)
X_train.shape, y_train.shape, X_test.shape, y_test.shape
((120, 4), (120,), (30, 4), (30,))

```

شکل ۱-۴۰ ابعاد داده‌های آموزش و آزمایش

سپس با استفاده از الگوریتم LDA کاهش ابعاد به دو بعد را انجام می‌دهیم.

```

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

```

سپس هر یک از انواع کرنل‌های خطی، چندجمله‌ای، گوسی و سیگموئید را تعریف می‌کنیم.

```

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

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

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

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

```

در ادامه یک تابعی نوشته‌یم که محاسبه‌ی هر یک از کرنل‌ها را به ازای نقاط داده‌ها امکان‌پذیر می‌کند. برای حل مسئله کواردتیک از cvxopt در حالت dual استفاده شده است. برای بهینه کردن تابع کواردتیک نیاز به محاسبه‌ی p , q , A , b , G و h می‌باشد؛ که A , b , G و h قیود نامساوی و مساوی مسئله لAGRANZ می‌باشند. می‌دانیم که مینیمم مسئله‌ی primal برابر با ماکریمم مقدار مسئله‌ی dual می‌باشد. چون مسئله‌ی ما به صورت convex و کواردتیک می‌باشد، می‌دانیم این دو مقدار با یکدیگر برابر می‌باشند. از آنجایی که primal زمانبر و سخت می‌باشد، از حالت dual استفاده می‌گردد. پس از بهینه‌سازی در گام بعدی support vector را محاسبه می‌کنیم. برای حالت مقادیر وزن‌ها و بایاس را محاسبه می‌کند. به راحتی به کمک علامت آن‌ها می‌توان تعیین کنیم. برای مسئله‌ی غیرخطی باید کرنل را با استفاده از توابع تعریف شده محاسبه کنیم. سپس مقادیر پیش‌بینی را با کمک علامت آن‌ها محاسبه می‌گردد (تابع Fit و Predict در این بخش تعریف شده‌اند).

```

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

# Support vectors have non-zero Lagrange multipliers
ind = np.arange(len(a))[sv]
a = a[sv]
sv_x = X[sv]
sv_y = y[sv]
numbers_of_sv = len(sv_y)

# Bias (For linear it is the intercept):
bias = 0
if len(a) > 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)
else:
    print("No support vectors found")
    bias = 0

```

```

# 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

```

برای حالت چند کلاسه باید این الگوریتم را به تعداد کلاس‌ها تکرار می‌کنیم. ما در اینجا از استفاده کرده‌ایم. در این حالت باید یک کلاس در نظر بگیریم از بقیه جدا کنیم، به این ترتیب برای هر کلاس انجام دهیم. به این منظور ماتریسم مقدار در نظر گرفته شده است.

```
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

```

```

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)

```

برای نمایش خطوط تصمیم‌گیری از تابع زیر استفاده شده است. در ابتدا یکسری نقاط در محدوده ویژگی اول و دوم ایجاد می‌کنیم سپس توسط دستور meshgrid از آرایه‌های یک بعدی، آرایه‌های دو بعدی بسازیم، که تمام جایگشت‌های مربوطه را در نظر بگیریم. سپس با استفاده از تابع multiclass_svm برای نقاط ایجاد شده در این صفحه، کلاس‌بندی (یک نقطه تصمیم) صورت می‌گیرد. توسط دستور contour خطوط تصمیم رسم می‌گردند.

```

def visualize_multiclass_classification(X_train, y_train1, kernel_type, trainset, classifiers,
class_labels, w_stack, b_stack,kernel_and_params,degrees):
    plt.figure(figsize=(8, 6))
    (_,_poly_params, RBF_params, sigmoid_params,C) = kernel_and_params
    # Plotting data points for each class
    for i, target_name in enumerate(class_labels):
        plt.scatter(X_train[y_train1 == i, 0], X_train[y_train1 == i, 1], label=target_name)

    if kernel_type == 'linear':
        h = .02 # step size in the mesh
        x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
        y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
        k=np.arange(x_min, x_max, h)
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
        x_test = np.c_[xx.ravel(), yy.ravel()]
        model=multiclass_svm(x_train,x_test, y_train, C,kernel_type, poly_params, RBF_params,
sigmoid_params)
        pred,_,_,_=model
        Z = pred.reshape(xx.shape)
        plt.contour(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

    else:
        h = .02 # step size in the mesh
        x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1

```

```

y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
k=np.arange(x_min, x_max, h)
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
x_test = np.c_[xx.ravel(), yy.ravel()]
model=multiclass_svm(x_train,x_test, y_train, C,kernel_type, poly_params, RBF_params,
sigmoid_params)
pred,_,_,_=model

Z = pred.reshape(xx.shape)
plt.contour(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

if trainset:
    plt.title(f'Data Points on Train Set {degrees}')
else:
    plt.title(f'Data Points on Test Set {degrees}')

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

```

با استفاده از دستورات زیر یک تابع برای رسم ماتریس درهم‌ریختگی را ایجاد می‌کنیم. مقادیر recall .precision و f1-score نیز محاسبه می‌گردند.

```

def calculate_metrics_and_plot(y_true, y_pred, degrees, labels=None):
    # Calculate confusion matrix
    cm = confusion_matrix(y_true, y_pred, labels=labels)

    # Calculate precision, recall, and F1-score
    precision = precision_score(y_true, y_pred, average=None, labels=labels)
    recall = recall_score(y_true, y_pred, average=None, labels=labels)
    f1 = f1_score(y_true, y_pred, average=None, labels=labels)

    # Calculate accuracy
    accuracy = np.sum(np.diag(cm)) / np.sum(cm)

    # Print precision, recall, F1-score, and accuracy
    for i in range(len(labels)):
        print(f"Class {labels[i]} - Precision: {precision[i]:.4f}, Recall: {recall[i]:.4f}, F1-score: {f1[i]:.4f}")

    print(f"Accuracy: {accuracy:.4f}")

    # Set custom color map and font size
    sns.set(font_scale=1.2)
    sns.set_style("whitegrid")

    # Plot the confusion matrix as a heatmap
    plt.figure(figsize=(8, 6))
    heatmap = sns.heatmap(cm, annot=True, fmt='d', cmap='BuGn', xticklabels=labels,
                          yticklabels=labels, annot_kws={"size": 14})

```

```

# Set the title font
heatmap.set_title(f'Confusion Matrix {degrees}', fontdict={'fontsize': 16, 'family': 'serif'})

plt.xlabel('Predicted')
plt.ylabel('True')

return accuracy

```

حال برای ذخیره سازی تصاویر ایجاد شده یک پوشه توسط دستور زیر ایجاد می‌کنیم و به داخل آن می‌رویم.

```

# Define the folder name
folder_name = "SupportVector"

```

Check if the folder already exists, and create it if not
if not os.path.exists(folder_name):
 os.makedirs(folder_name)

Commented out IPython magic to ensure Python compatibility.
%cd SupportVector

حال برای بررسی اثرات کرنل چندجمله‌ای از درجه ۱ تا ۱۰، در ابتدا برای ذخیره سازی تصاویر آرایه‌های خالی ایجاد شده است. سپس درون حلقه for ابتدا یک شی از از مدل را ایجاد می‌کنیم. سپس توسطتابع visualize_multiclass_classification مرزهای تصمیم مربوط به داده‌های آموزش و آزمون، توسطتابع calculate_metrics_and_plot ماتریس درهم‌ریختگی را رسم می‌کنیم.

```

degrees = range(1, 11)
acc = []
images_train = []
images_test = []
images_conf = []
# Train SVM with polynomial kernels of degree 1 to 10
for degree in degrees:
    #*****here is the callable function *****
    # Split the data into training and test sets
    x_train = X_train_lda
    x_test = X_test_lda

    model=multiclass_svm(x_train,x_test, y_train, degree,'polynomial', poly_params=(1, degree),
RBF_params=0.5, sigmoid_params=(1, 0.01))
    pred, kernel_and_params,w_catch, b_catch, classifiers=model

    #*****here is the callable function *****
    class_0 = 0
    class_1 = 1
    class_2 = 2

    plt.figure()
    visualize_multiclass_classification(X_train_lda, y_train, kernel_and_params[0], True, classifiers,
iris.target_names[class_0:class_2+2], w_catch, b_catch, kernel_and_params, degree)
    filename1 = f'svm_degree_{degree}_train_support_vector.png'
    plt.savefig(filename1)

```

```

images_train.append(imageio.imread(filename1))
plt.close()

plt.figure()
visualize_multiclass_classification(X_test_lda, y_test, kernel_and_params[0], False, classifiers,
iris.target_names[class_0:class_2+2], w_catch, b_catch, kernel_and_params, degree)
filename2 = f'svm_degree_{degree}_test_support_vector.png'
plt.savefig(filename2)
images_test.append(imageio.imread(filename2))
plt.close()

print(iris.target_names[class_0:class_2+2])

# Evaluate the model on the training set
if __name__ == "__main__":
    plt.figure()
    y_true = y_test
    y_pred = pred
    print(f"Degree: {degree}\n")
    acc.append(calculate_metrics_and_plot(y_true, y_pred, degree, labels=[0, 1, 2]))
    filename3 = f'svm_degree_{degree}_Confusion_Matrix.png'
    plt.savefig(filename3)
    images_conf.append(imageio.imread(filename3))
    plt.close()

# Path to save the GIF
gif_path_train = './svm_poly_degrees_train_support_vector.gif'
imageio.mimsave(gif_path_train, images_train, duration=10)
# Display the path to the GIF
print(f"GIF saved as {gif_path_train}")

# Path to save the GIF
gif_path_test = './svm_poly_degrees_test_support_vector.gif'
imageio.mimsave(gif_path_test, images_test, duration=10)
# Display the path to the GIF
print(f"GIF saved as {gif_path_test}")

```

در ادامه نتایج مربوط را نمایش می‌دهیم. از آنجایی که با تغییرات هایپر پارامترها نیز موفق به نمایش تا درجه ۱۰ نشده‌ایم، نتایج تا مرتبه ۷ را نمایش می‌دهیم.

```

Degree: 2
Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Class 1 - Precision: 0.9091, Recall: 1.0000, F1-score: 0.9524
Class 2 - Precision: 1.0000, Recall: 0.8333, F1-score: 0.9091
Accuracy: 0.9667

```

```

Degree: 1
Class 0 - Precision: 0.7778, Recall: 1.0000, F1-score: 0.8750
Class 1 - Precision: 1.0000, Recall: 0.6000, F1-score: 0.7500
Class 2 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Accuracy: 0.8667

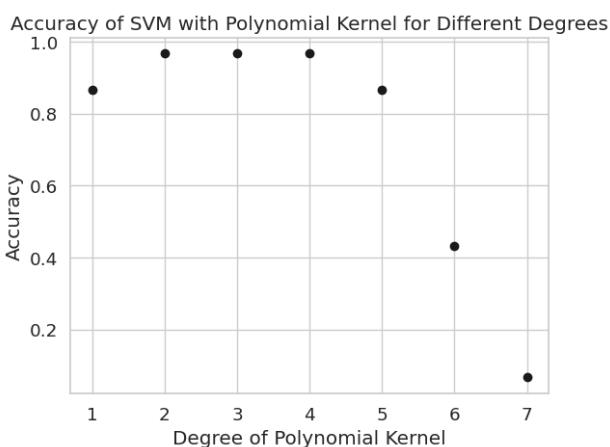
```

```

Degree: 4
Class 0 - Precision: 0.9333, Recall: 1.0000, F1-score: 0.9655
Class 1 - Precision: 1.0000, Recall: 0.9000, F1-score: 0.9474
Class 2 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000
Accuracy: 0.9667
Degree: 6
Class 0 - Precision: 0.0000, Recall: 0.0000, F1-score: 0.0000
Class 1 - Precision: 1.0000, Recall: 0.9000, F1-score: 0.9474
Class 2 - Precision: 0.2222, Recall: 0.6667, F1-score: 0.3333
Accuracy: 0.4333
Degree: 7
Class 0 - Precision: 0.0000, Recall: 0.0000, F1-score: 0.0000
Class 1 - Precision: 0.0000, Recall: 0.0000, F1-score: 0.0000
Class 2 - Precision: 0.1000, Recall: 0.3333, F1-score: 0.1538
Accuracy: 0.0667

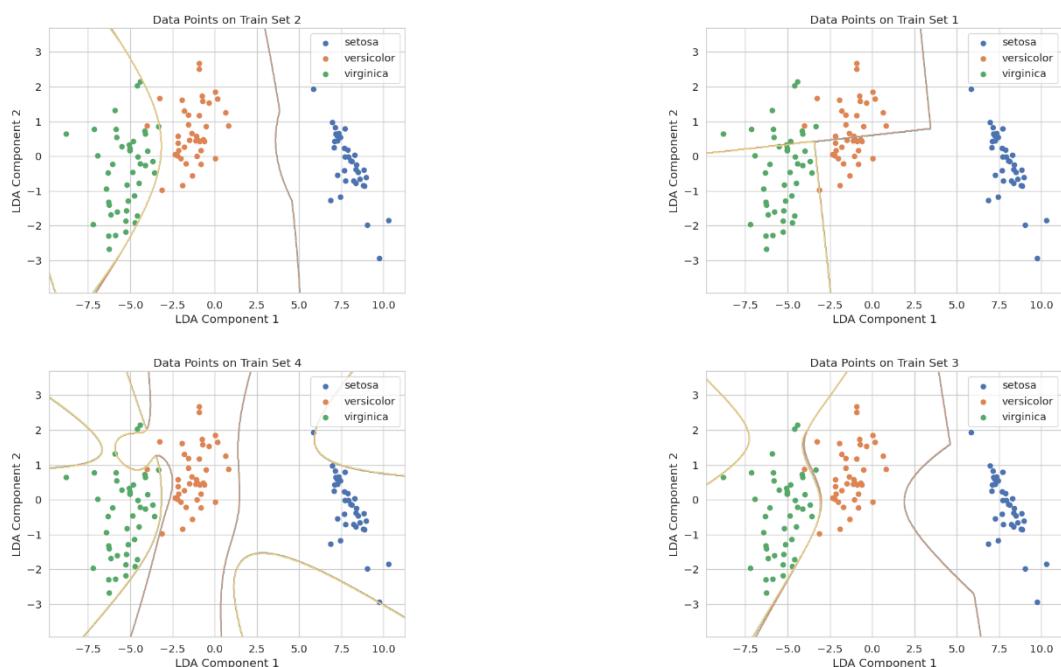
```

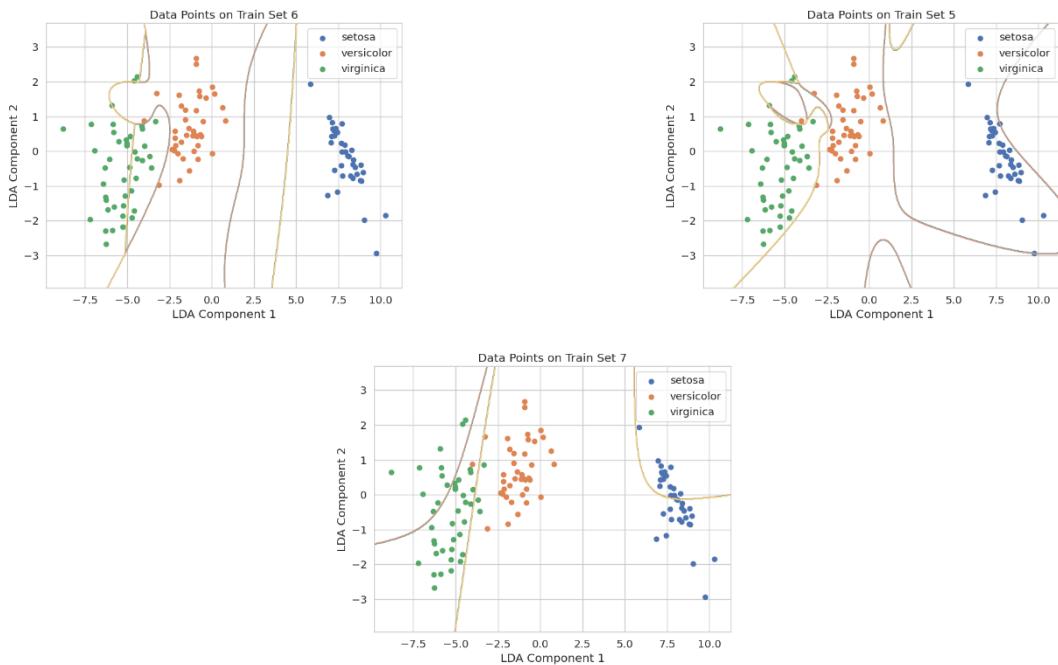
شکل ۱-۴۱ نمایش میزان دقت مدل با درجات ۱



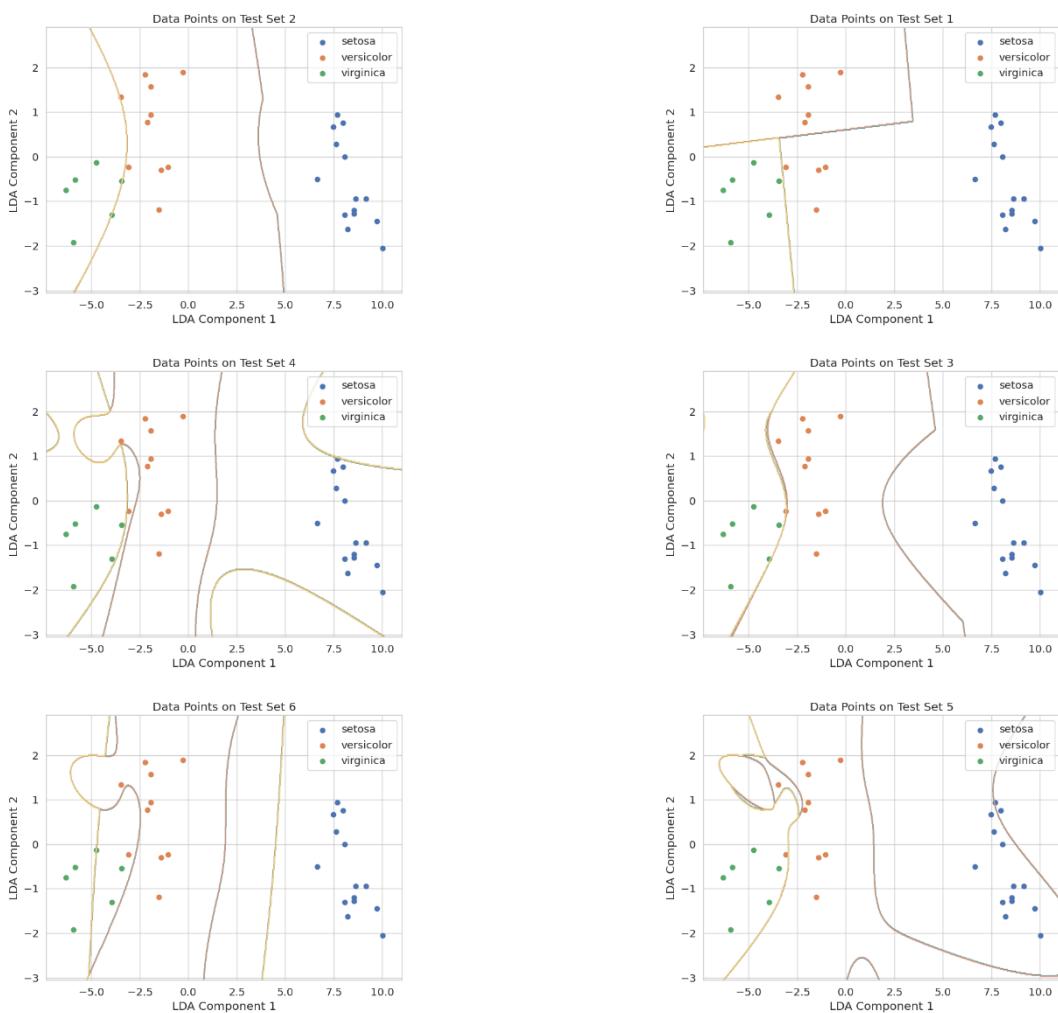
شکل ۱-۴۲ دقت الگوریتم با افزایش درجه

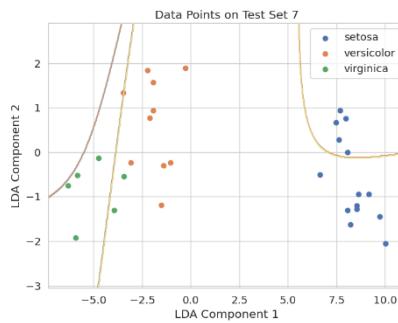
همانطور که از نتایج مشخص است تا درجه ۵ مدل دقت خوبی دارد؛ اما پس از آن مدل دچار مشکل می‌گردد. در زیر مرزهای تصمیم مربوط به داده‌های آموزش و آزمون، کاتریس درهم ریختگی را نمایش می‌دهیم.



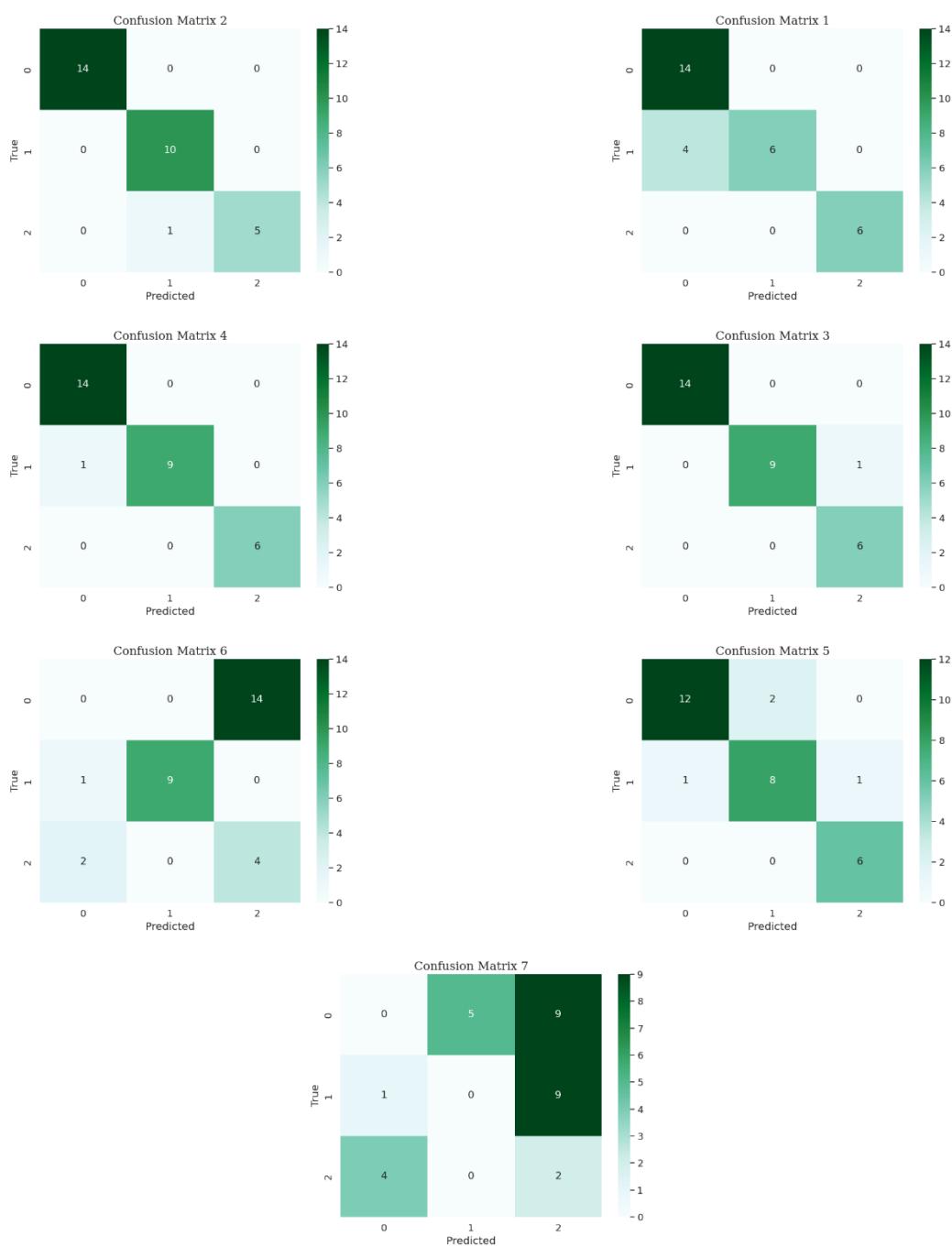


شکل ۱-۴۳ مراحلی تعمیم داده‌های آموزش





شکل ۱-۴۴ مرزهای تصمیم داده‌های آزمون



شکل ۱-۴۵ ماتریس درهم‌بینگی

همانطور که در ماتریس درهم ریختگی نشان داده شده است، با استفاده از کرنل چندجمله‌ای مرتبه ۶ و ۷ مدل به درستی عمل نکرده است.

محل ذخیره‌سازی gif‌های مربوطه در زیر نماش داده شده است. برای دسترسی راحت داخل درایو به آدرس زیر نیز قرار داده شده است.

```
GIF saved as ./svm_poly_degrees_train_support_vector_LDA.gif  
GIF saved as ./svm_poly_degrees_test_support_vector_LDA.gif  
GIF saved as ./svm_poly_degrees_Confusion_Matrix_LDA.gif
```

شکل ۱-۴۶ محل ذخیره‌سازی gif‌ها

https://drive.google.com/file/d/131rJFT0FhOo2gVjoKjKrsq78tXCTzH5q/view?usp=drive_link
https://drive.google.com/file/d/12U5x135Nn9E69Zzs5lMKAO9P3hR0ro5q/view?usp=drive_link
https://drive.google.com/file/d/1N0sIw9ak2X-oIOTqBOw-b5stoP4x1OFt/view?usp=drive_link

برای بررسی تا ۱۰ مرتبه کرنل چندجمله‌ای از الگوریتم کاهش ابعاد PCA می‌کنیم.

```
from sklearn.decomposition import PCA  
from sklearn.preprocessing import StandardScaler
```

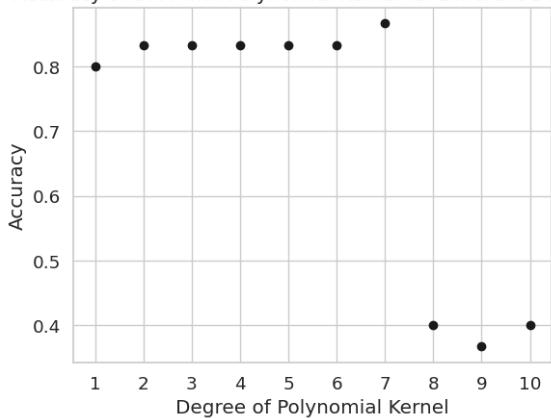
```
#scaling and centering the data  
sc = StandardScaler()  
X_train_scaled = sc.fit_transform(X_train)  
X_test_scaled = sc.transform(X_test)  
  
pca = PCA (n_components=2)  
X_train_pca = pca.fit_transform(X_train_scaled)  
X_test_pca = pca.transform(X_test_scaled)
```

در زیر نتایج مربوطه را نمایش می‌دهیم.

```
Degree: 2  
Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000  
Class 1 - Precision: 0.8571, Recall: 0.6000, F1-score: 0.7059  
Class 2 - Precision: 0.5556, Recall: 0.8333, F1-score: 0.6667  
Accuracy: 0.8333  
Degree: 4  
Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000  
Class 1 - Precision: 0.8571, Recall: 0.6000, F1-score: 0.7059  
Class 2 - Precision: 0.5556, Recall: 0.8333, F1-score: 0.6667  
Accuracy: 0.8333  
Degree: 6  
Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000  
Class 1 - Precision: 0.7778, Recall: 0.7000, F1-score: 0.7368  
Class 2 - Precision: 0.5714, Recall: 0.6667, F1-score: 0.6154  
Accuracy: 0.8333  
Degree: 8  
Class 0 - Precision: 1.0000, Recall: 0.0714, F1-score: 0.1333  
Class 1 - Precision: 0.7778, Recall: 0.7000, F1-score: 0.7368  
Class 2 - Precision: 0.2000, Recall: 0.8667, F1-score: 0.3077  
Accuracy: 0.4000  
Degree: 10  
Class 0 - Precision: 0.0000, Recall: 0.0000, F1-score: 0.0000  
Class 1 - Precision: 0.7273, Recall: 0.8000, F1-score: 0.7619  
Class 2 - Precision: 0.2105, Recall: 0.6667, F1-score: 0.3200  
Accuracy: 0.4000  
  
Degree: 1  
Class 0 - Precision: 0.7778, Recall: 1.0000, F1-score: 0.8750  
Class 1 - Precision: 1.0000, Recall: 0.5000, F1-score: 0.6667  
Class 2 - Precision: 0.7143, Recall: 0.8333, F1-score: 0.7692  
Accuracy: 0.8000  
Degree: 5  
Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000  
Class 1 - Precision: 0.8571, Recall: 0.6000, F1-score: 0.7059  
Class 2 - Precision: 0.5556, Recall: 0.8333, F1-score: 0.6667  
Accuracy: 0.8333  
Degree: 7  
Class 0 - Precision: 1.0000, Recall: 1.0000, F1-score: 1.0000  
Class 1 - Precision: 0.7778, Recall: 0.7000, F1-score: 0.7368  
Class 2 - Precision: 0.5714, Recall: 0.6667, F1-score: 0.6154  
Accuracy: 0.8333  
Degree: 9  
Class 0 - Precision: 0.9333, Recall: 1.0000, F1-score: 0.9655  
Class 1 - Precision: 1.0000, Recall: 0.7000, F1-score: 0.8235  
Class 2 - Precision: 0.6250, Recall: 0.8333, F1-score: 0.7143  
Accuracy: 0.8667  
Degree: 11  
Class 0 - Precision: 0.0000, Recall: 0.0000, F1-score: 0.0000  
Class 1 - Precision: 0.3043, Recall: 0.7000, F1-score: 0.4242  
Class 2 - Precision: 0.5714, Recall: 0.6667, F1-score: 0.6154  
Accuracy: 0.3667
```

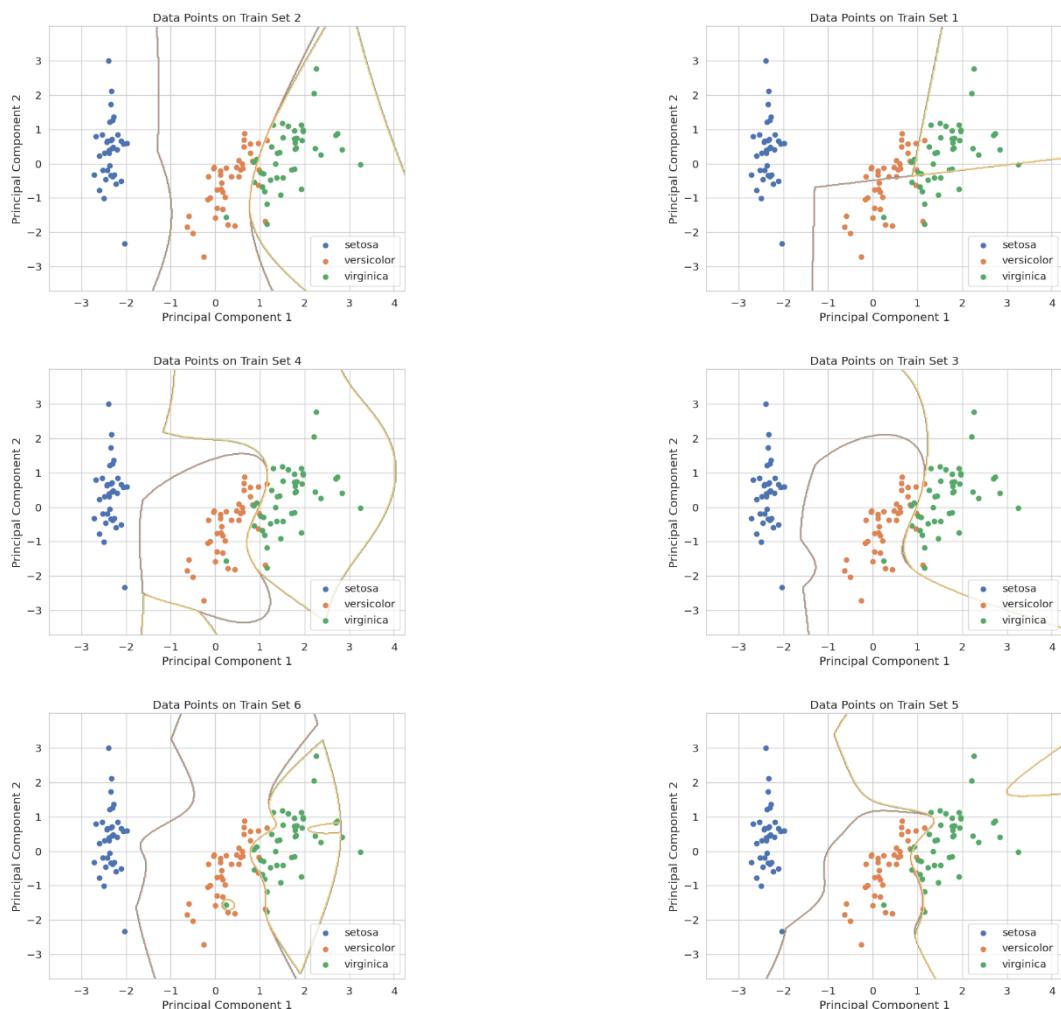
شکل ۱-۴۷ نمایش میزان دقیقت مدل با درجهات ۱-۱۰

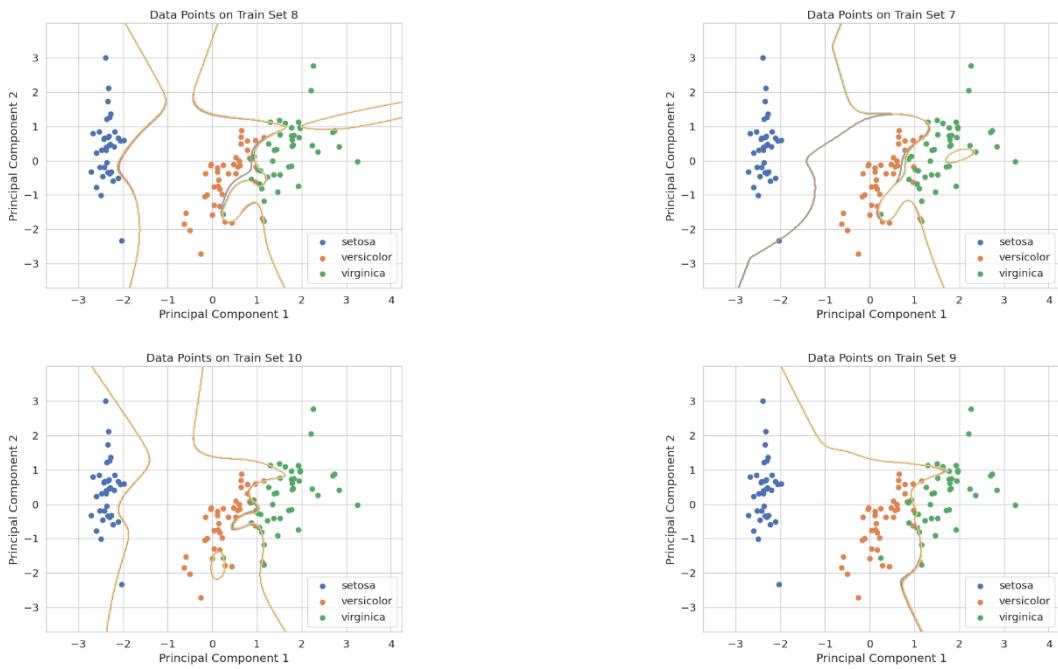
Accuracy of SVM with Polynomial Kernel for Different Degrees



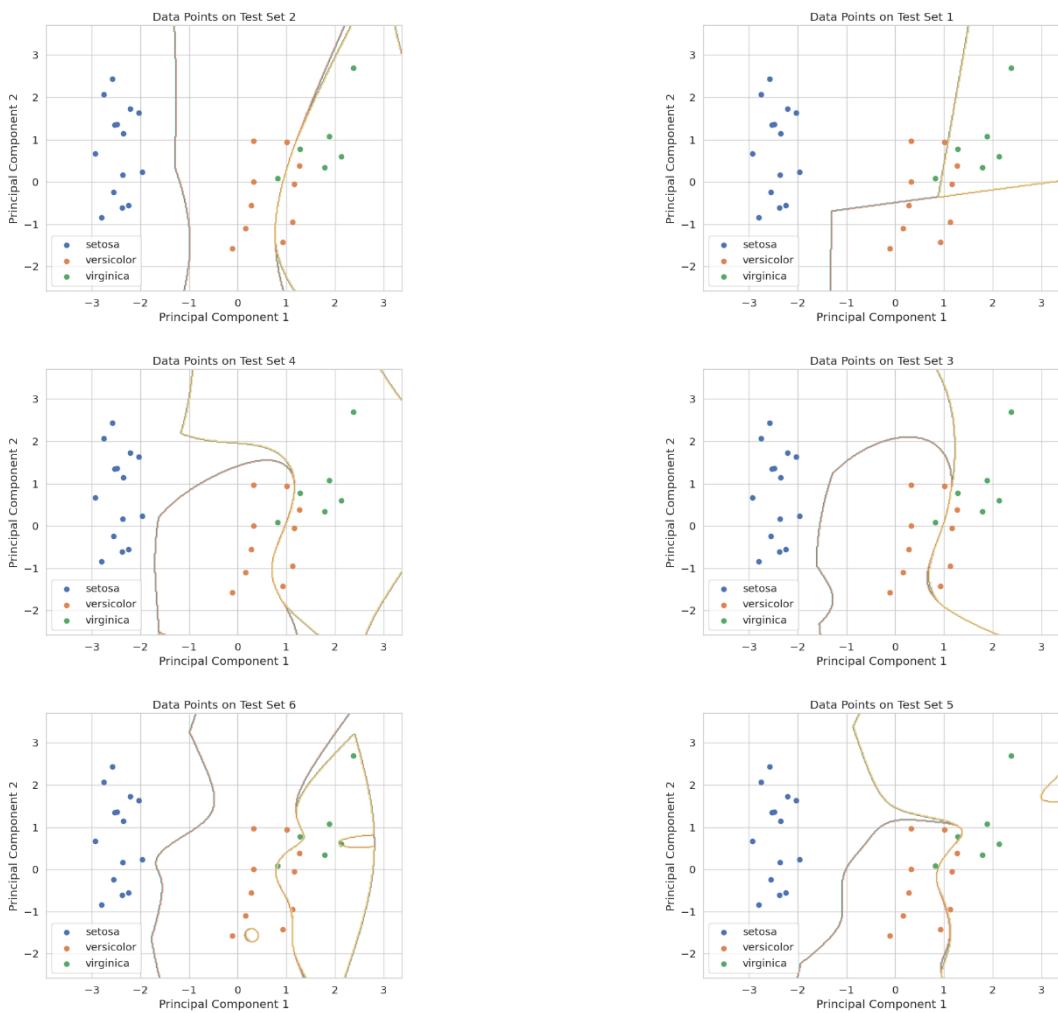
شکل ۱-۴۸ دقت الگوریتم با افزایش درجه

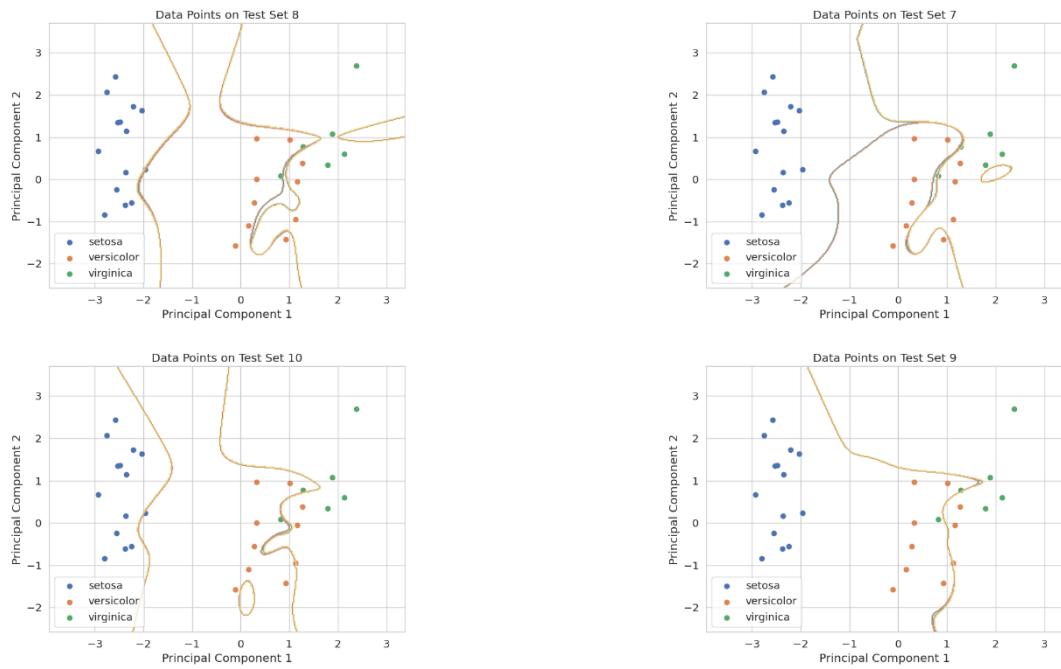
همانطور که از نتایج مشخص است تا درجه ۷ مدل دقیق خوبی دارد، اما پس از آن مدل دچار مشکل می‌گردد. در زیر مراحلی تصمیم مربوط به داده‌های آموزش و آزمون، کاتریس در هم ریختگی را نمایش می‌دهیم.



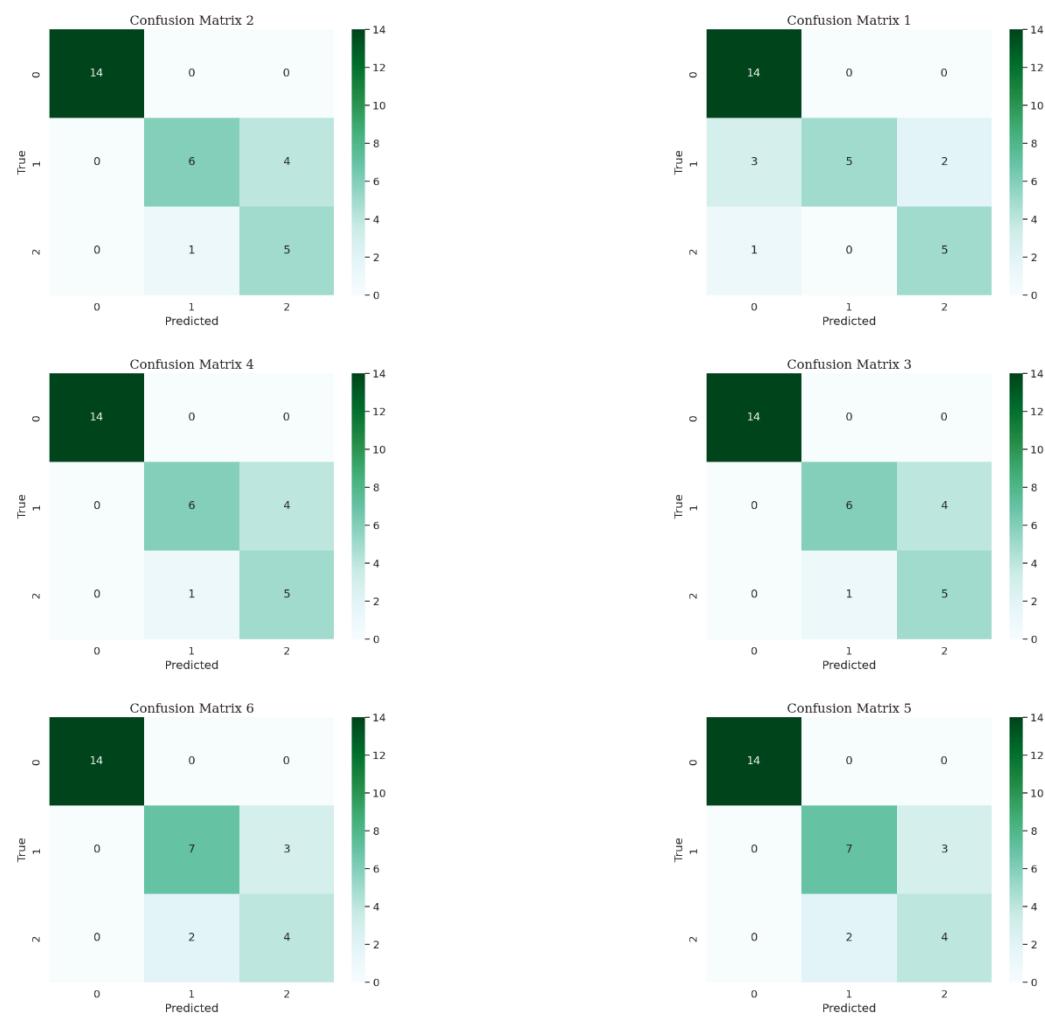


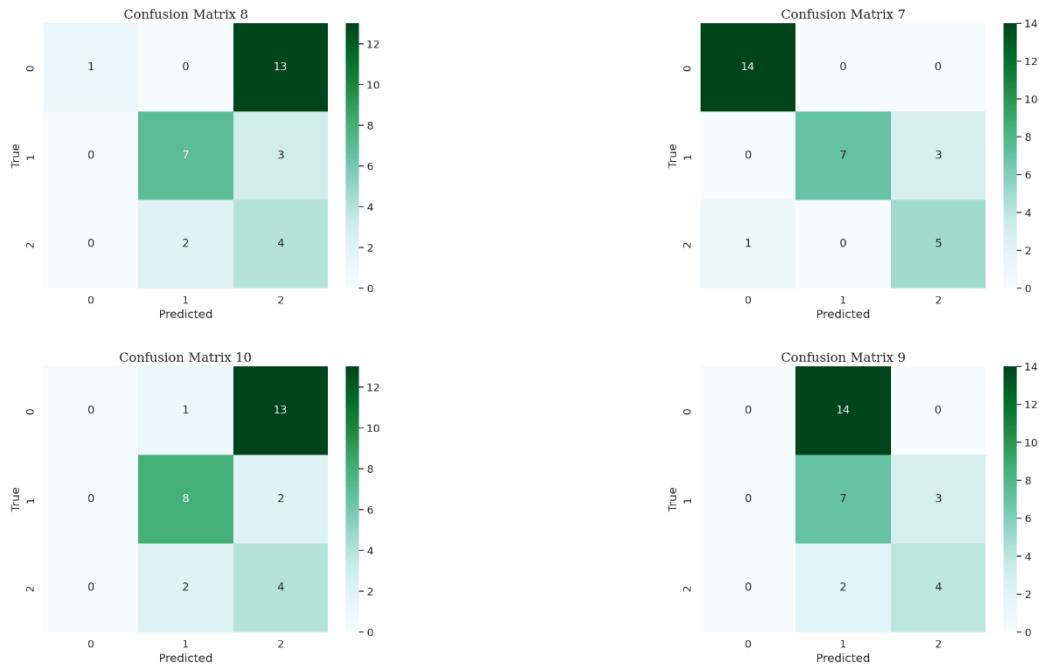
شکل ۱-۴۹ مزهای تصمیم داده‌های آموزش





شکل ۱-۵۰ مرزهای تصمیمی دادههای آزمون





شکل ۱-۵۱ ماتریس درهم ریختگی

همانطور که در ماتریس درهم ریختگی نشان داده شده است، با استفاده از کرنل چندجمله‌ای مرتبه ۸، ۹ و ۱۰ مدل به درستی عمل نکرده است.
 محل ذخیره‌سازی gif‌های مربوطه در زیر نماش داده شده است. برای دسترسی راحت داخل درایو به آدرس زیر نیز قرار داده شده است.

```
GIF saved as ./svm_poly_degrees_train_support_vector_PCA.gif
GIF saved as ./svm_poly_degrees_test_support_vector_PCA.gif
GIF saved as ./svm_poly_degrees_Confusion_Matrix_PCA.gif
```

شکل ۱-۵۲ محل ذخیره‌سازی gif‌ها

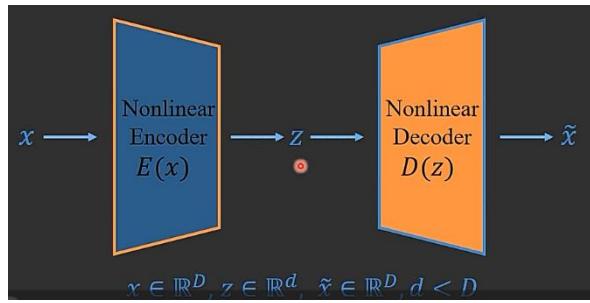
https://drive.google.com/file/d/1ictN3P50b1z8tzGu2ww2bFUyWtDx8_o/view?usp=drive_link
https://drive.google.com/file/d/16rdB1LkyKlyX6dM-ByzaY2NRiwPFIswa/view?usp=drive_link
https://drive.google.com/file/d/1EmaimghASFhtsrH3D6R8YpJ4oGsIJKXa/view?usp=drive_link

۲- سوال سوم

۲-۱- بخش الف

Encoderها برای کاهش ابعاد به صورت غیرخطی کاربرد دارد. این موضوع سبب کاهش بعد ابعاد بیشتر می‌گردد، در این حالت باعث حفظ بیشتر اطلاعات می‌گردد؛ این موضوع به دلیل پیچیدگی‌های ذاتی نگاشت غیرخطی این موضوع امکان‌پذیر است. این مدل به راحتی قابلیت آموزش ندارد، زیرا ما اطلاعی از نحوه ویژگی کاهش بعد یافته نداریم. علاقه‌مند هستیم که رمزگذار غیرخطی اطلاعات ورودی را در درون خروجی فشرده کند و اگر encoder مدل خوبی باشد باید خروجی تمام اطلاعات ورودی را به صورت فشرده درون خود داشته باشد.

Decoder مدل‌هایی هستند که امکان بازسایی ورودی براساس خروجی فشرده شده را دارند. اگر encoder به خوبی آموزش دیده باشد این انتظار را می‌توان از decoder داشت. به اسن ترکیب autoencoder گفته می‌شود که در شکل زیر نمایش داده شده است.



شکل ۲-۱ ساختار

برای آموزش دیدن می‌تواند تابع هزینه زیر را مینیمم کند.

$$J = \sum_{n=1}^N (\tilde{x}_n - x_n)^2$$

پس از آموزش می‌توانیم از decoder صرف نظر کنیم.

به طور خلاصه می‌توانیم چالش‌های موجود در این مقاله را به صورت موردی زیر بیان کنیم.

عدم توازن داده‌ها: تعداد تراکنش‌های تقلبی (کلاس اقلیت) بسیار کمتر از تراکنش‌های عادی (کلاس اکثیریت) است، که باعث می‌شود مدل‌های یادگیری ماشینی نتوانند به خوبی از داده‌های تقلبی یاد بگیرند. مدل‌های بیشتر تمایل به شناسایی کلاس اکثیریت داشته باشند، که در نتیجه دقت تشخیص معاملات تقلبی کاهش می‌باید.

تغییر الگوهای تقلب: تقلب‌ها دائمًا در حال تغییر و تکامل هستند، بنابراین مدل‌ها باید بتوانند به سرعت خود را با این تغییرات تطبیق دهند.

دقت و قابلیت اعتماد: مدل‌های تشخیص تقلب باید دارای دقت بالا و نرخ پایینی از تشخیص‌های نادرست باشند تا هزینه‌های مالی و زمانی ناشی از اشتباهات کاهش یابد.

برای رفع این مشکلات روش‌های زیر در این مقاله پیشنهاد شده است.

استفاده از Autoencoder Neural Network: این مدل‌ها با یادگیری الگوهای معمول در داده‌ها، به شناسایی تراکنش‌های غیرمعمول و تقلبی کمک می‌کنند. این رویکرد به طور خاص برای کاهش مشکل عدم توازن داده‌ها مناسب است زیرا بر روی داده‌های عادی تمرکز می‌کند و تقلب‌ها را به عنوان استثنای تشخیص می‌دهد.

استفاده از **Denoising Autoencoder**: این تکنیک به تقویت دقت مدل کمک می‌کند، زیرا باعث می‌شود مدل نسبت به نویز و داده‌های غیرعادی مقاوم‌تر شود و توانایی بیشتری در شناسایی تقلب‌های جدید داشته باشد.

استفاده از فرآیند **Oversampling**: با افزایش تعداد نمونه‌های تقلبی از طریق تکنیک‌های oversampling، مدل‌ها می‌توانند بهتر الگوهای تقلبی را تشخیص دهند و دقت خود را افزایش دهند.

۲-۲- بخش ب

به طور خلاصه می‌توان چنین بیان نمود که:

Autoencoder Neural Network

- این شبکه شامل یک لایه ورودی، چند لایه پنهان، و یک لایه خروجی است.
- لایه‌های پنهان وظیفه دارند که ویژگی‌های مهم را از داده‌ها استخراج کنند و سپس این ویژگی‌ها را فشرده کنند.
- شبکه پس از فشرده‌سازی، داده‌ها را بازسازی می‌کند تا شباهت بین ورودی و خروجی را به حداقل برساند و هر گونه داده غیرعادی (تراکنش‌های تقلبی) را شناسایی کند.

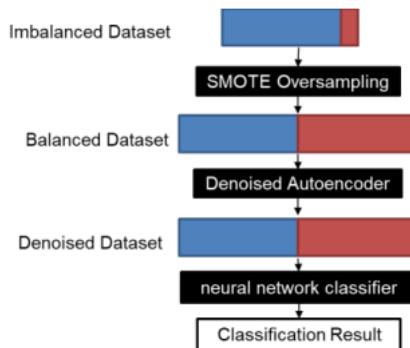
Denoising Autoencoder (DAE)

- این شبکه به نوعی از Autoencoder است که برای کاهش نویز و بازسازی دقیق‌تر داده‌ها طراحی شده است.
- DAE با افروzen نویز به داده‌های ورودی و سپس تلاش برای بازسازی داده‌های بدون نویز آموزش داده می‌شود، که باعث افزایش دقت مدل در شناسایی تراکنش‌های تقلبی می‌شود.

Oversampling Process

- برای مقابله با عدم توازن داده‌ها، فرآیند Oversampling برای افزایش تعداد نمونه‌های تقلبی استفاده می‌شود، که مدل را قادر می‌سازد تا الگوهای تقلبی را بهتر یاد بگیرد.

در شکل زیر نحوه عملکرد شبکه نشان داده شده است.



شکل ۲-۲ مراحل عملکرد شبکه

طبق مقاله، مدل شبکه عصبی مورد استفاده یک شبکه عصبی خودکار رمزگذار نویزدار (Denoising Autoencoder) است. این مدل شبکه عصبی خودکار رمزگذار برای یادگیری داده‌های نویزی به کار می‌رود. شبکه عصبی خودکار رمزگذار نویزدار شامل دو بخش اصلی است:

رمزگذار (Encode): این بخش وظیفه فشردهسازی دادهها را بر عهده دارد. به عبارت دیگر، دادههای ورودی را به یک نمایش فشرده با بعد کمتر از ورودی تبدیل می‌کند.

رمزگشا (Decoder): این بخش وظیفه بازسازی دادهها از نمایش فشرده را بر عهده دارد. به عبارت دیگر، نمایش فشرده را به دادههای اصلی تبدیل می‌کند.

فرآیند عملکرد شبکه عصبی خودکار رمزگذار نویزدار را می‌توان به صورت زیر بیان نمود.
ورود دادههای نویزی: دادههای ورودی به شبکه عصبی رمزگذار وارد می‌شوند. این دادهها ممکن است شامل نویز باشند.

فشردهسازی دادهها: رمزگذار دادهها را فشرده کرده و یک نمایش فشرده از آنها را تولید می‌کند. این نمایش فشرده حاوی ویژگی‌های کلیدی دادهها است.

بازسازی دادهها: نمایش فشرده به شبکه عصبی رمزگشا وارد می‌شود. رمزگشا تلاش می‌کند تا از این نمایش فشرده، دادههای اصلی را بازسازی کند.

در فرآیند آموزش، شبکه عصبی یاد می‌گیرد که چگونه دادههای ورودی را به طور دقیق فشرده و بازسازی کند. این کار باعث می‌شود که شبکه عصبی ویژگی‌های کلیدی دادهها را یاد بگیرد و بتواند آنها را از دادههای ناقص یا نویزی تشخیص دهد.

مزایای استفاده از شبکه عصبی خودکار رمزگذار نویزدار را می‌توان چنین بیان نمود:
افزایش دقت طبقه‌بندی: این مدل با یادگیری ویژگی‌های اصلی دادهها، می‌تواند دقت طبقه‌بندی را به ویژه در دادههای نامتعادل افزایش دهد.

کاهش نویز: با استفاده از فرآیند رمزگذاری و بازسازی، دادههای نویزی تصحیح می‌شوند که منجر به بهبود کیفیت دادههای ورودی برای مراحل بعدی می‌شود.

نمایش فشرده: ایجاد نمایش‌های فشرده از دادهها به کاهش ابعاد و افزایش کارایی مدل کمک می‌کند.
ساختار شبکه عصبی خودکار رمزگذار نویزدار شامل لایه‌های مختلفی است که در زیر توضیح داده شده است:
لایه ورودی: دریافت دادههای ورودی نویزی.

لایه‌های رمزگذار: چندین لایه متوالی که دادهها را به نمایش‌های فشرده تبدیل می‌کنند.

لایه‌های رمزگشا: چندین لایه متوالی که نمایش‌های فشرده را بازسازی می‌کنند.

لایه خروجی: تولید دادههای بازسازی شده که باید مشابه دادههای ورودی اصلی باشند.

با توجه به توضیحات فوق، این مدل به دلیل توانایی بالا در تشخیص و تصحیح نویز و همچنین افزایش دقت طبقه‌بندی، به عنوان یک روش مؤثر در پردازش دادههای نامتعادل مورد استفاده قرار می‌گیرد.

۲- بخش ج

در ابتدا فایل دیتاست مورد نظر را دانلود می‌کنیم. سپس یک فایل را برای ذخیرهسازی دیتاست ایجاد می‌کنیم. از روش سوم ذکر شده در فیلم‌ها، با استفاده از دستور زیر فایل دیتاست را در گوگل کولب قرار می‌دهیم. ID مربوط به اشتراک فایل قرار گرفته در گوگل درایو را در اینجا قرار می‌دهیم.

```
import os
```

```
# Define the folder name  
folder_name = "Data"
```

```

# Check if the folder already exists, and create it if not
if not os.path.exists(folder_name):
    os.makedirs(folder_name)

# Commented out IPython magic to ensure Python compatibility.
%cd Data
!pip install --upgrade --no-cache-dir gdown
!gdown 1xzfOpzTwMv73HPmV3QbpuplYVptiG0cK

```

فایل فشرده را با استفاده از دستور زیر از حالت فشرده خارج می‌کنیم.
`!unzip '/content/Data/archive.zip' -d '/content/Data/archive'`
 کتابخانه‌های مورد نیاز را می‌افزاییم.

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

```

سپس دیتاست آپلود شده را بارگذاری می‌کنیم، آدرس آن را با کلیک راست بر روی آن و انتخاب `- Copy path` می‌بابیم. ۵ ردیف ابتدایی آن را توسط دستور زیر نمایش می‌دهیم.

```

df = pd.read_csv('/content/Data/archive/creditcard.csv')
df.head()

```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.9666272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

شکل ۲-۳ ۵ ردیف ابتدایی دیتاست

با استفاده از دستورات زیر داده‌های ورودی و خروجی را مشخص می‌کنیم. داده‌ی زمان را در نظر نمی‌گیریم.

```
X = df.drop(['Time', 'Class'], axis=1).values
```

```
y = df['Class'].values
```

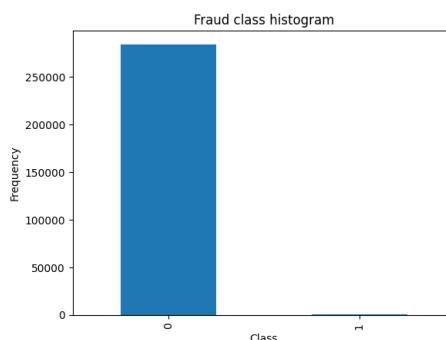
سپس با استفاده از دستورات زیر تعداد داده‌های مربوط به هر کلاس را به صورت تصویر نمایش می‌دهیم. همانطور که از شکل زیر مشخص است داده‌ها به صورت نامتوازن می‌باشند.

```
count_classes = pd.value_counts(df['Class'], sort = False)
```

```

count_classes.plot (kind='bar')
plt.title ("Fraud class histogram")
plt.xlabel ("Class")
plt.ylabel ("Frequency")

```



شکل ۲-۴ نمایش تعداد داده‌های هر کلاس

نمونه‌های مصنوعی از کلاس اقلیت استفاده می‌کند تا تعادل بیشتری بین کلاس‌ها ایجاد شود. این تکنیک به جای تکرار ساده نمونه‌های موجود (که می‌تواند به overfitting منجر شود)، نمونه‌های جدیدی را با استفاده از روش‌های خطی بین نمونه‌های موجود در کلاس اقلیت ایجاد می‌کند.

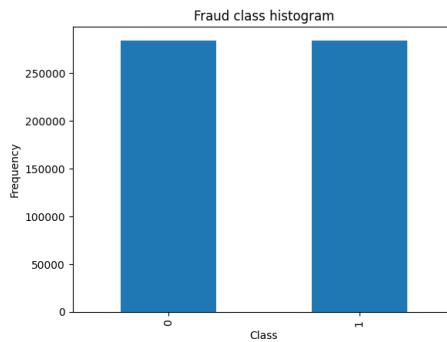
```
from imblearn.over_sampling import SMOTE
```

```
smote = SMOTE(sampling_strategy='auto', random_state=76)
X_resampled, y_resampled = smote.fit_resample(X, y)

df_resampled = pd.DataFrame(X_resampled, columns=df.columns[1:-1])
df_resampled['Class'] = y_resampled
```

```
count_classes = pd.value_counts(df_resampled['Class'], sort=False)
```

```
count_classes.plot(kind='bar')
plt.title("Fraud class histogram")
plt.xlabel("Class")
plt.ylabel("Frequency")
```



شکل ۲-۵ تعداد داده‌های هر کلاس پش از SMOT

سپس داده‌ها را به دو بخش آموزش و آزمایش تقسیم می‌کنیم. با استفاده از تابع fit مربوط به کلاس داده‌های آموزش را استاندارد می‌کنیم. داده‌های آزمون نیز توسط متod ransform استاندارد می‌گردند.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=76, stratify=y_resampled)
```

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

با کمک دستور زیر نویز گوسی (نرمال) با میانگین صفر را به داده‌ها اضافه می‌کنیم.

```
noise_factor = 0.1
```

```
X_train_noisy = X_train_scaled + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train_scaled.shape)
```

```
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
```

```

X_test_noisy = X_test_scaled + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=X_test_scaled.shape)
X_test_noisy = np.clip(X_test_noisy, 0., 1.)

```

در ادامه آموزش و ارزیابی یک شبکه Denoising Autoencoder و شبکه classifier صورت گرفته است. ساختار کلی Autoencoder را در زیر می‌توان مشاهده نمود که دارای ۷ لایه می‌باشد.

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

شکل ۲-۶ مدل Autoencoder

یک callback است که برای ذخیره مدل بر اساس شرایط مشخصی استفاده می‌شود. monitor='val_loss' متغیری که بر اساس آن مدل ذخیره می‌شود. در اینجا بر اساس از دست دادن کمترین مقدار تابع هزینه بر روی داده‌های اعتبارسنجی (val_loss)، مدل ذخیره می‌شود. اگر save_best_only=True باشد، فقط بهترین مدل (کمترین val_loss) ذخیره می‌شود. verbose=1 برای نمایش پیام‌هایی که نشان می‌دهد آیا مدل ذخیره شده است یا خیر استفاده می‌گردد. یک EarlyStopping callback است که برای متوقف کردن آموزش مدل بر اساس شرایط مشخصی استفاده می‌شود.

monitor='val_loss' متغیری که بر اساس آن آموزش متوقف می‌شود. در اینجا براساس val_loss انجام می‌شود. patience=10 تعداد epoch‌هایی که بعد از آن اگر بهبودی در val_loss حاصل نشود، آموزش متوقف می‌شود. verbose=1 برای نمایش پیام‌هایی که نشان می‌دهد آیا مدل ذخیره شده است یا خیر استفاده می‌گردد. این دو callback به طور همزمان برای آموزش مدل خودرنگار استفاده می‌شوند تا بهترین مدل براساس val_loss را ذخیره کنند و در صورت لزوم آموزش را متوقف کنند تا از بیش‌برازش جلوگیری شود. برای autoencoder تابع هزینه Mean Squared Error (MSE) برای آموزش بر روی داده‌های نویززده (X_train_noisy) و خروجی متناظر آن‌ها (X_train_scaled) تعریف شده است؛ که برای اندازه‌گیری تفاوت بین ورودی و خروجی استفاده می‌شود. همچنین الگوریتم بهینه‌سازی Adam برای بهینه‌سازی شبکه استفاده شده است.

```

import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import confusion_matrix, classification_report, r2_score,
precision_score, recall_score, accuracy_score

input_dim = X_train_noisy.shape[1]

```

```

encoding_dim = 10

input_layer = Input(shape=(input_dim,))
encoder = Dense(22, activation="relu")(input_layer)
encoder = Dense(15, activation="relu")(encoder)
encoder = Dense(encoding_dim, activation="relu")(encoder)

decoder = Dense(15, activation="relu")(encoder)
decoder = Dense(22, activation="relu")(decoder)
decoder = Dense(input_dim, activation="sigmoid")(decoder)

autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

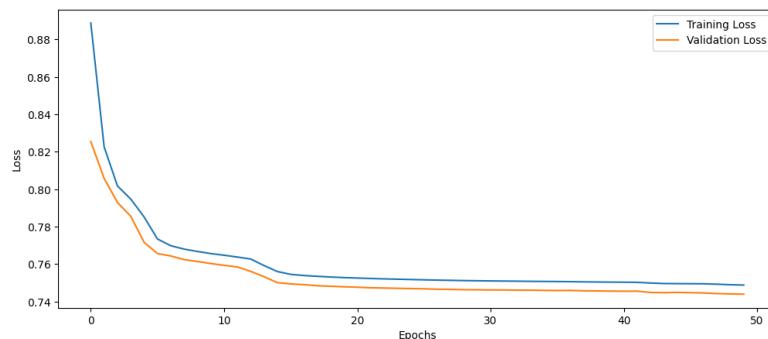
model_checkpoint      = ModelCheckpoint('best_autoencoder.h5',      monitor='val_loss',
save_best_only=True, verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1)
history_ae = autoencoder.fit(X_train_noisy, X_train_scaled,
epochs=50,
batch_size=500,
shuffle=True,
validation_split=0.2,
callbacks=[model_checkpoint, early_stopping])

autoencoder.load_weights('best_autoencoder.h5')

X_train_denoised = autoencoder.predict(X_train_noisy)
X_test_denoised = autoencoder.predict(X_test_noisy)

plt.figure(figsize=(12, 5))
plt.plot(history_ae.history['loss'], label='Train Loss')
plt.plot(history_ae.history['val_loss'], label='Validation Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()

```



شکل ۲-۷ نمودار هزینه شبکه autoencoder

حال خروجی autoencoder که بدون نویز می‌باشد را به یک classifier با ساختار زیر داده می‌شود. این سک شبکه عصبی با عمق زیاد و اتصالات کامل است.

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

شكل ٢-٨ مدل

```

input_layer = Input(shape=(input_dim,))
classifier = Dense(22, activation="relu")(input_layer)
classifier = Dense(15, activation="relu")(classifier)
classifier = Dense(10, activation="relu")(classifier)
classifier = Dense(5, activation="relu")(classifier)
classifier = Dense(2, activation="softmax")(classifier)

classifier_model = Model(inputs=input_layer, outputs=classifier)
classifier_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

model_checkpoint_classifier = ModelCheckpoint('best_classifier.h5', monitor='val_loss',
save_best_only=True, verbose=1)

history_clf = classifier_model.fit(X_train_denoised, y_train,
epochs=50,
batch_size=500,
shuffle=True,
validation_split=0.2,
callbacks=[model_checkpoint_classifier])

classifier_model.load_weights('best_classifier.h5')

y_pred = classifier_model.predict(X_test_denoised)
y_pred_classes = np.argmax(y_pred, axis=1)

rscore_1 = r2_score(y_test, y_pred_classes)
print("R2 Score: ", rscore_1)

plt.figure(figsize=(12, 5))
plt.plot(history_clf.history['loss'], label='Train Loss')
plt.plot(history_clf.history['val_loss'], label='Validation Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()

cf_matrix = confusion_matrix(y_test, y_pred_classes)
cf_matrix_percent = cf_matrix.astype('float') / cf_matrix.sum(axis=1)[:, np.newaxis] * 100

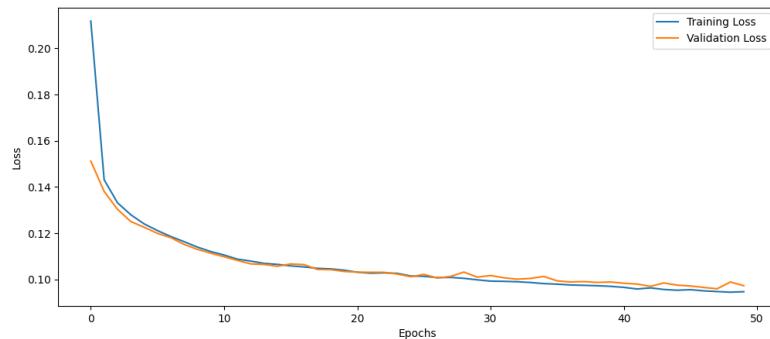
```

```

plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix_percent, annot=True, fmt='.2f', cmap='Blues', annot_kws={"size": 12})

class_report = classification_report(y_test, y_pred_classes)
print("\nClassification Report:\n", class_report)

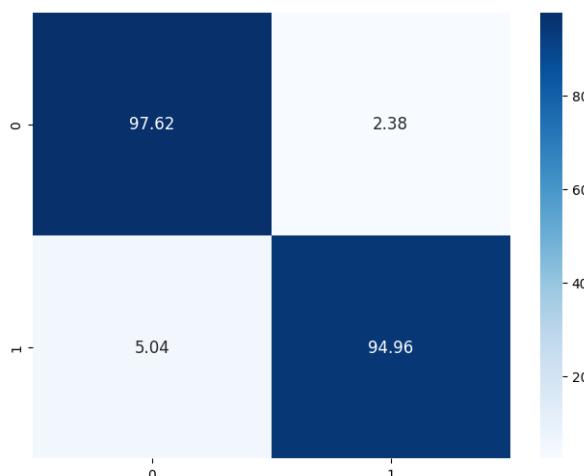
```



شکل ۲-۹ نمودار هزینه شبکه classifier

۲-۴- بخش ۵

در این بخش ماتریس درهم ریختگی و گزارشی از نحوهی عملکرد classifier را نمایش می‌دهیم.



شکل ۲-۱۰ ماتریس درهم ریختگی داده‌های آزمون

```

' Classification Report:
precision    recall   f1-score   support
          0       0.95      0.98      0.96     56863
          1       0.98      0.95      0.96     56863

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

Precision: 0.96
Recall: 0.96
F1 Score: 0.96
Accuracy: 0.96

```

شکل ۲-۱۱ گزارش از نحوهی عملکرد سیستم

مدل شبکه عصبی خودکار رمزگذار نویزدار با این ساختار و پارامترها، عملکرد خوبی در طبقه‌بندی معاملات تقلیلی و واقعی از یکدیگر دارد. Accuracy، Recall و F1 score برای هر دو کلاس به طور متوسط ۹۶٪ می‌باشند که نشان می‌دهد که مدل به خوبی با تعادل بین Recall و Accuracy عمل می‌کند و به خوبی معاملات تقلیلی را تشخیص می‌دهد.

این نتایج نشان می‌دهند که استفاده از شبکه عصبی خودکار رمزگذار نویزدار به همراه روش‌های بالانس کردن مانند SMOT برای پردازش داده‌های نامتوازن توانسته است بهبود قابل توجهی در عملکرد طبقه‌بندی داشته باشد. ارزیابی سیستم‌های کلاس‌بندی خطی باینری معمولاً با استفاده از معیارهایی انجام می‌شود که به طور کلی شامل موارد زیر است:

۱. دقت (Accuracy): نسبت تعداد داده‌هایی که به درستی تشخیص داده شده‌اند به کل داده‌های مورد ارزیابی است. این معیار می‌تواند به صورت فرمولی به صورت زیر نشان داده شود:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

۲. دقت تشخیص مثبت (Precision): نسبت داده‌هایی که به درستی تشخیص داده شده‌اند به تمام داده‌هایی که به عنوان مثبت تشخیص داده شده‌اند. فرمول دقت تشخیص مثبت به صورت زیر است:

$$Precision = \frac{TP}{TP+FP}$$

۳. حساسیت (Recall): نسبت داده‌هایی که به درستی تشخیص داده شده‌اند به کل داده‌هایی که در واقع مثبت هستند. فرمول حساسیت به صورت زیر است:

$$Recall = \frac{TP}{TP+FN}$$

۴. اندازه‌گیری F1 scoare (F1 score): میانگین هارمونیک دقت و حساسیت است و به صورت زیر محاسبه می‌شود:

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

این معیارها از جمله معیارهای ارزیابی رایج برای سیستم‌های کلاس‌بندی خطی باینری هستند. در مسائلی که توزیع برچسب‌ها نامتوازن است، استفاده از معیار Accuracy به تنها‌یی ممکن است تصمیم‌گیری اشتباہی ایجاد کند. از آنجا که توزیع برچسب‌ها نامتوازن است، معیار Accuracy ممکن است به خوبی عملکرد مدل را نمایش ندهد. به عنوان مثال، فرض کنید داده‌ها دارای ۹۰٪ برچسب کلاس ۰ و ۱۰٪ برچسب کلاس ۱ باشند. یک مدل ساده که همه‌ی نمونه‌ها را به عنوان کلاس ۰ پیش‌بینی کند، به دلیل توزیع نامتوازن داده‌ها، دارای دقت ۹۰٪ خواهد بود. اما این مدل به هیچ وجه یک مدل خوب برای تشخیص کلاس ۱ نیست.

برای ارزیابی بهتر عملکرد مدل‌ها در مواجهه با توزیع نامتوازن، معیارهای مانند Precision، Recall و F1 score مناسب هستند. این معیارها به طور جداگانه برای هر کلاس محاسبه می‌شوند و به ارزیابی دقیق‌تری از توانایی مدل در تشخیص هر کلاس کمک می‌کنند.

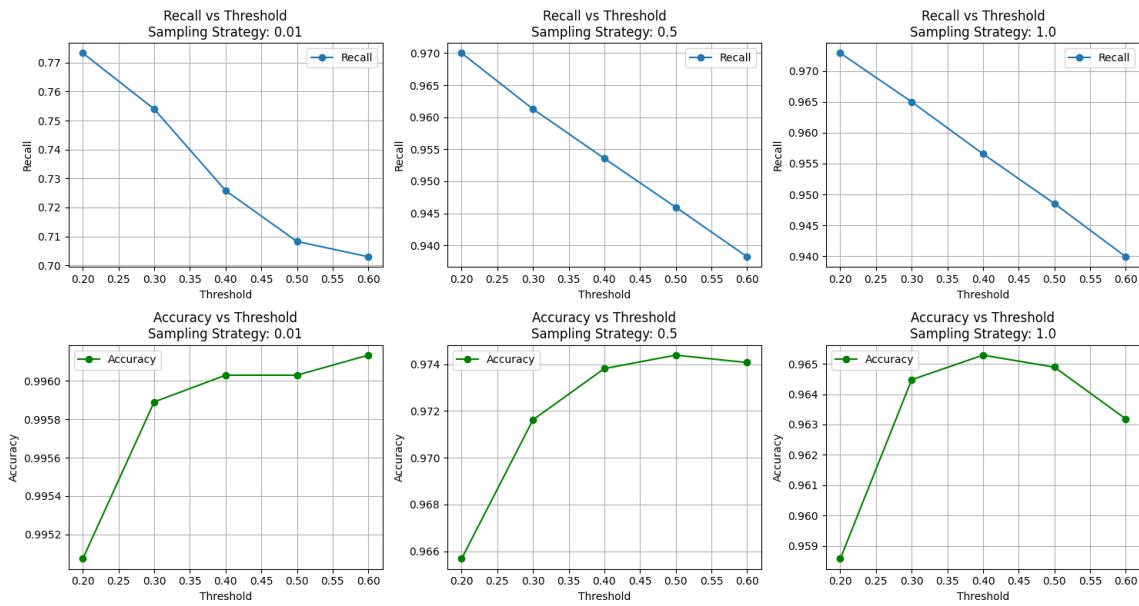
به طور کلی، در مواجهه با مسائل نامتوازن، بهتر است از معیارهای Precision، Recall، F1 score و همچنین ماتریس درهم‌ریختگی (Confusion Matrix) استفاده کنیم تا عملکرد مدل را به صورت جامع‌تر و دقیق‌تر ارزیابی کنیم. Recall و Accuracy به عنوان مکمل یکدیگر استفاده می‌گردند.

۵-۵- بخش ۵

این بخش برای ارزیابی تأثیر استفاده از روش‌های نمونه‌برداری برای حل مشکل ناتوازن بودن داده‌ها در مدل‌های آموزش دیده شده است. در اینجا، چندین روش نمونه‌برداری با نسبت‌های مختلف برای استفاده از SMOTE تعیین شده است. این نسبت‌ها به ترتیب نشان می‌دهند که چند برابر نمونه‌های اقلیتی نسبت به نمونه‌های اکثریت تولید شوند. در یک حلقه، برای هر یک از روش‌های نمونه‌برداری، داده‌ها را با استفاده از SMOTE نمونه‌برداری کرده و

سپس مدل classifier و شبکه autoencoder بر روی داده‌های نمونه‌برداری شده آموزش می‌دهد و نتایج را ارزیابی می‌کند.

sampling_strategies = [0.01, 0.5, 1.0]
thresholds = [0.2, 0.3, 0.4, 0.5, 0.6]



شکل ۲-۱۲ تأثیر نسبت‌های مختلف برای استفاده از SMOTE در عملکرد سیستم

با استفاده از دستورات زیر هر دو نمودار را با یکدیگر رسم می‌نماییم.

```
plt.figure(figsize=(15, 8))
```

```
for idx, strategy in enumerate(sampling_strategies):
```

```
    plt.subplot(2, len(sampling_strategies), idx + 1)
    plt.plot(thresholds, all_recall_scores[idx], marker='o', linestyle='-', label='Recall')
```

```
    plt.plot(thresholds, all_accuracy_scores[idx], marker='o', linestyle='-', label='Accuracy',
             color='g')
```

```
    plt.title(f'AccurRecall & Accuracyacy vs Threshold\nSampling Strategy: {strategy}')
```

```
    plt.xlabel('Threshold')
```

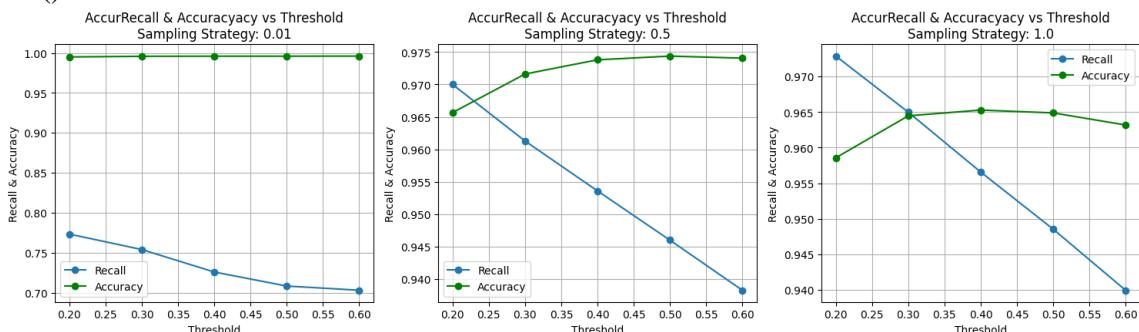
```
    plt.ylabel('Recall & Accuracy')
```

```
    plt.grid(True)
```

```
    plt.legend()
```

```
plt.tight_layout()
```

```
plt.show()
```



شکل ۲-۱۳ تأثیر نسبت‌های مختلف برای استفاده از SMOTE در عملکرد سیستم

استراتژی نمونهبرداری ۱۰۰ که نسبتاً کمترین تعداد نمونه را اضافه می‌کند، recall پایینی دارد. این نشان می‌دهد که مدل توانایی خوبی در شناسایی کلاس کم‌تعدادتر ندارد. برای این دیستاست خاص، استراتژی ۱۰۰ باعث کمبود recall شده است، در حالی که استراتژی‌های بالاتر از ۲۰ بهبود قابل توجهی در recall داشته‌اند. با افزایش استراتژی نمونهبرداری، recall بهبود می‌یابد. به عبارت دیگر، با افزایش تعداد نمونه‌ها، مدل قادر به بهتر شناسایی کلاس کم‌تعدادتر می‌شود.

همچنین با افزایش تعداد نمونه‌ها، accuracy نیز ممکن است کاهش یابد، اما در کل برای همه استراتژی‌ها accuracy بسیار بالاست و مدل به درستی توانسته است نمونه‌ها را شناسایی کند. بهترین استراتژی نمونه‌برداری بر اساس recall و accuracy، استراتژی ۱.۰ است که نسبت تعداد نمونه‌های هر دو کلاس را برابر می‌کند. در نتیجه، برای مسائلی که توزیع برچسب‌ها نامتوافق است، انتخاب استراتژی مناسب نمونه‌برداری می‌تواند بهبود مدل را افزایش دهد، به خصوص در مواردی که بازخوانی کلاس کم‌تعدادتر مهم است.

۶-۲ بخش و

برای آموزش یک مدل شبکه عصبی بر روی داده‌های ورودی X_{train_noisy} و برچسب‌های y_{train} استفاده می‌شود. مدل این‌جا به صورت یک شبکه عصبی کاملاً پیوسته (fully-connected neural network) تعریف شده است که دیگر از autoencoder پرای حذف نویز استفاده نشده است.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=76, stratify=y)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
noise_factor = 0.1
X_train_noisy = X_train_scaled + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=X_train_scaled.shape)
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
X_test_noisy = X_test_scaled + noise_factor * np.random.normal(loc=0.0, scale=1.0,
size=X_test_scaled.shape)
X_test_noisy = np.clip(X_test_noisy, 0., 1.)
```

```
input_dim = X_train.shape[1]
```

```
input_layer = Input(shape=(input_dim,))  
classifier = Dense(22, activation="relu")(input_layer)  
classifier = Dense(15, activation="relu")(classifier)  
classifier = Dense(10, activation="relu")(classifier)  
classifier = Dense(5, activation="relu")(classifier)  
classifier = Dense(2, activation="softmax")(classifier)
```

```
classifier_model = Model(inputs=input_layer, outputs=classifier)
classifier_model.compile(optimizer='adam',
                         loss='sparse_categorical_crossentropy',
                         metrics=['accuracy'])
```

```
model_checkpoint_classifier = ModelCheckpoint('best_classifier.h5', monitor='val_loss',  
save_best_only=True, verbose=1)
```

```

history_clf = classifier_model.fit(X_train_noisy, y_train,
                                    epochs=10,
                                    batch_size=256,
                                    shuffle=True,
                                    validation_split=0.2,
                                    callbacks=[model_checkpoint_classifier])

classifier_model.load_weights('best_classifier.h5')

y_pred = classifier_model.predict(X_test_noisy)
y_pred_classes = np.argmax(y_pred, axis=1)

rscore_1 = r2_score(y_test, y_pred_classes)
print("R2 Score: ", rscore_1)

plt.figure(figsize=(12, 5))
plt.plot(history_clf.history['loss'], label='Train Loss')
plt.plot(history_clf.history['val_loss'], label='Validation Loss')
plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.show()

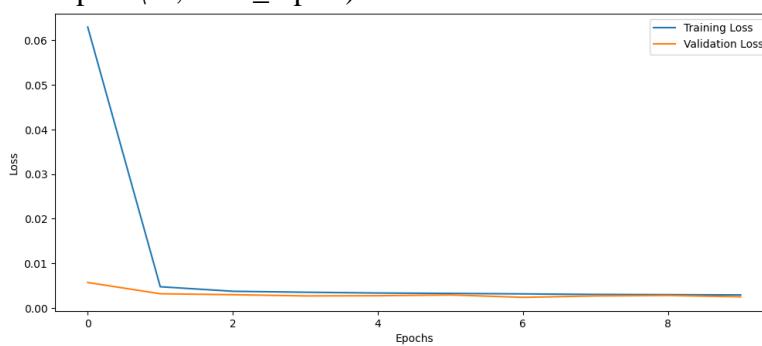
cf_matrix = confusion_matrix(y_test, y_pred_classes)

cf_matrix_percent = cf_matrix.astype('float') / cf_matrix.sum(axis=1)[:, np.newaxis] * 100

plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix_percent, annot=True, fmt='.2f', cmap='Blues', annot_kws={"size": 12})

class_report = classification_report(y_test, y_pred_classes)
print("\nClassification Report:\n", class_report)

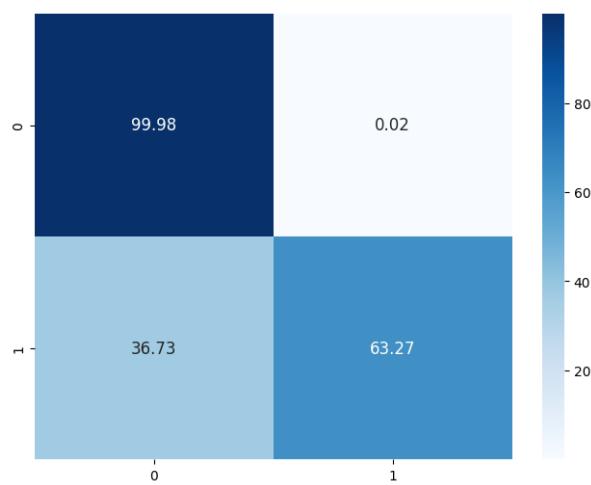
```



شكل ٢-١٤ نمودار تابع هزینه مربوط به classifier

Classification Report:				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	56864
1	0.86	0.63	0.73	98
accuracy			1.00	56962
macro avg	0.93	0.82	0.86	56962
weighted avg	1.00	1.00	1.00	56962

شكل ٢-١٥ گزارش عملکرد سیستم



شکل ۲-۱۶ ماتریس درهم ریختگی

همانطور که از نتایج مشخص است عدم حذف نویز و متوازن نبودن دیتا ها موجب بد شدن نتایج در مقایسه با قسمت های قبل می شود. عدم توازن می تواند به overfitting منجر شود؛ که در اینجا به دلیل بیشتر بودن داده ها کلاس صفر، overfitting برای این کلاس رخ داده است.