

دانشگاه صنعتی خواجه نصیرالدین طوسی  
دانشکده مهندسی برق - کروه مهندسی کنترل

## درس یادگیری ماشین

### مینی پروژه سوم

نام و نام خانوادگی	علی مهرابی
شماره دانشجویی	۴۰۲۲۳۸۵۴
تاریخ	اردیبهشت ۱۴۰۳
استاد درس	دکتر علیاری

## فهرست مطالب

۶	لینک های مربوطه	
۶	.....	۱.۱ لینک مربوط به گیت هاب
۶	.....	۲.۱ لینک مربوط به گوگل کلب
۷	سوال اول	
۷	.....	۱.۲ بخش اول
۱۳	.....	۲.۲ بخش دوم
۱۴	.....	۳.۲ بخش سوم
۱۸	.....	۴.۲ بخش چهارم
۲۱	سوال سوم	
۲۱	.....	۱.۳ بخش اول
۲۴	.....	۲.۳ بخش دوم
۲۶	.....	۳.۳ بخش سوم
۲۹	.....	۴.۳ بخش چهارم
۳۰	.....	۵.۳ بخش پنجم
۳۱	.....	۶.۳ بخش ششم

## فهرست تصاویر

۷	نحوه فراخوانی داده Iris	۱
۷	نام و تعداد ویژگی ها و ابعاد دیتاست Iris	۲
۸	بررسی بالانس بودن داده ها	۳
۸	میانگین و واریانس ویژگی ها	۴
۸	میانگین و میانه ویژگی ها به تفکیک کلاس ها	۵
۹	شناسایی داده های Nan	۶
۹	بصری سازی شناسایی داده های Nan	۷
۹	نمایش توزیع ویژگی ها	۸
۱۰	کد نمودار جعبه ای	۹
۱۰	نمودار جعبه ای داده ها	۱۰
۱۰	ماتریس همبستگی دیتاست Iris	۱۱
۱۱	سه همبستگی برتر در داده ها	۱۲
۱۲	کد مربوط به بصری سازی با t-SNE	۱۳
۱۲	t-SNE	۱۴
۱۳	کد بصری سازی به کمک PCA	۱۵
۱۳	بصری سازی به کمک PCA	۱۶
۱۴	واریانس توضیح داده شده با توجه به Principal Components	۱۷
۱۴	ساخت طبقه بند SVM با کرnel خطی	۱۸
۱۴	نتایج طبقه بند SVM با کرnel خطی	۱۹
۱۵	ماتریس درهم ریختگی طبقه بند SVM با کرnel خطی	۲۰
۱۵	ناحیه تصمیم گیری طبقه بند SVM با کرnel خطی با کاهش بعد	۲۱
۱۶	کد ساختن ۱۰ طبقه بند با درجات ۱ تا ۱۰	۲۲
۱۶	نمودار دقت با توجه به درجه کرnel	۲۳
۱۶	ماتریس در هم ریختگی با توجه به درجه کرnel	۲۴
۱۷	توابع ایجاد مش و کانتور در حالت کاهش بافت	۲۵
۱۷	ناحیه تصمیم گیری طبقه بند با توجه به درجات مختلف poly	۲۶
۱۸	کد ذخیره نمودارها به صورت عکس	۲۷
۱۸	کد ذخیره نمودارها به صورت گیف	۲۸
۱۸	تعریف کرnel های مختلف	۲۹
۱۹	SVM	۳۰
۱۹	تعریف مش و کانتور کرnel polynomial	۳۱
۱۹	polynomial با درجه ۱ به صورت دستی	۳۲
۲۰	polynomial با درجه ۲ به صورت دستی	۳۳
۲۰	polynomial با درجه ۳ به صورت دستی	۳۴

۲۰	polynomial با درجه ۴ به صورت دستی	۳۵
۲۱	polynomial با درجه ۵ به صورت دستی	۳۶
۲۱	polynomial با درجه ۶ به صورت دستی	۳۷
۲۲	polynomial با درجه ۷ به صورت دستی	۳۸
۲۲	polynomial با درجه ۸ به صورت دستی	۳۹
۲۲	polynomial با درجه ۹ به صورت دستی	۴۰
۲۳	polynomial با درجه ۱۰ به صورت دستی	۴۱
۲۳	نمودار دقیق با توجه به درجه به صورت دستی	۴۲
۲۳	روند کلی الگوریتم های به کارفته در مقاله	۴۳
۲۴	معماری کلی و بخش های یک اتوانکودر	۴۴
۲۵	لایه ورودی و خروجی و داده اصلی در ساختمان اتو انکودر	۴۵
۲۵	نمایش توصیفی از لایه ها و نورون ها در شبکه عصبی طبقه بند	۴۶
۲۶	نمایش نامتعادل بودن برچسب های دیتابست کارت اعتباری	۴۷
۲۶	تقسیم داده به آموزش، اعتبار سنجدی و ارزیابی	۴۸
۲۷	نتیجه اعمال الگوریتم SMOTE	۴۹
۳۰	نتایج طبقه بندی	۵۰
۳۰	ماتریس در هم ریختگی طبقه بند به کمک لایه Auto encoder	۵۱
۳۱	تغییرات Accuracy و Recall با تغییر آستانه مدل	۵۲
۳۲	تغییرات Accuracy و Recall با تغییر آستانه SMOTE	۵۳
۳۲	تغییرات Accuracy و Recall با تغییر آستانه SMOTE به صورت جزئی	۵۴
۳۲	نتایج بدون استفاده از SMOTE	۵۵
۳۳	ماتریس در هم ریختگی بدون استفاده از SMOTE	۵۶
۳۳	تغییرات recall و accuracy بدون SMOTE و auto encoder	۵۷

## فهرست جداول



## فهرست برنامه‌ها

۲۷	.....	algorithm SMOTE	۱
۲۷	.....	encoding label	۲
۲۸	.....	noise Gaussian adding	۳
۲۸	.....	encoder auto	۴
۲۸	.....	epochs ۳ last encoder auto	۵
۲۸	.....	classifier network neural	۶
۲۸	.....	weights best classifier network neural	۷
۲۹	.....	data test on metrics	۸
۳۰	.....	model the for holds tresh different	۹
۳۱	.....	SMOTE for holds tresh different	۱۰



## ۱ لینک های مربوطه

### ۱.۱ لینک مربوط به گیت هاب

از این لینک گیت هاب ([Github](#)) می توانید برای دسترسی به صفحه Github مربوط به این پروژه استفاده کنید.

### ۲.۱ لینک مربوط به گوگل کلب

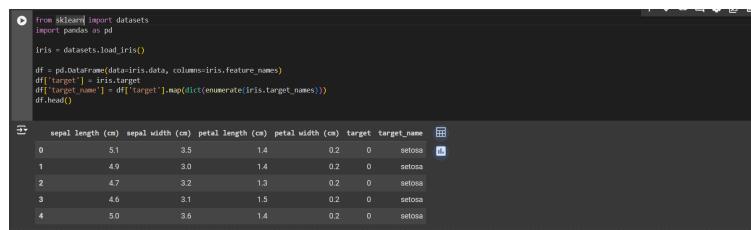
از این لینک گوگل کلب ([Google Colab](#)) می توانید برای دسترسی به notebook نوشته شده دسترسی پیدا کنید.

## ۲ سوال اول

دیتاست IRIS یک مجموعه ساده و بسیار معروف برای استفاده به عنوان اهداف آموزشی است. این دیتاست توسط Ronald A. Fisher در مقاله‌ای با عنوان "The use of multiple measurements in taxonomic problems." در ۱۹۶۳ معرفی گردید. این دیتاست دارای ۱۵۰ نمونه است و با معرفی چهار ویژگی طول و عرض گلبرگ و طول و عرض کاسبرگ قصد طبقه‌بندی داده‌ها در ۳ طبقه‌گیاه Iris شامل Iris-virginica، Iris-versicolor و Iris-setosa را دارد.

### ۱.۲ بخش اول

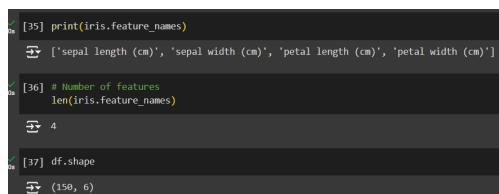
در ابتدا از کتابخوانه sklearn داده را به صورت **شکل ۱** فراخوانی می‌کنیم همانطور که دیده می‌شود ما هم کلاس داده را به صورت عددی



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	target_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

شکل ۱: نحوه فراخوانی داده Iris

و هم به صورت categorical آورده ایم. علاوه بر این گفته که ۴ ویژگی داریم. در کل داده ما ۱۵۰ نمونه دارد و می‌توانیم ابعاد داده و نام ویژگی‌ها را از روی **شکل ۲** ببینیم.



```
[35] print(iris.feature_names)
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

[36] # Number of features
len(iris.feature_names)

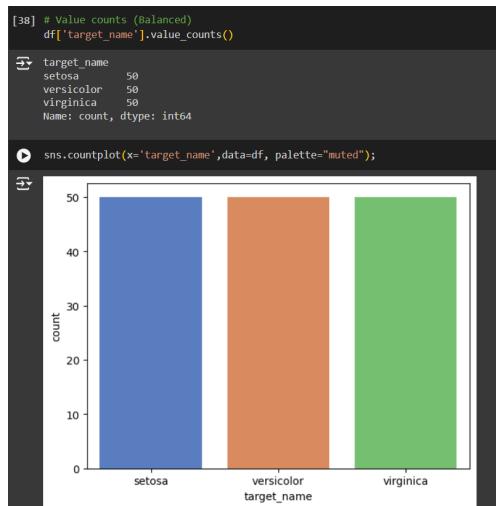
[37] df.shape
(150, 5)
```

شکل ۲: نام و تعداد ویژگی‌ها و ابعاد دیتاست Iris

در مرحله بعد balanced بودن دیتاست را چک می‌کنیم. مطابق **شکل ۳** دیده می‌شود که نمونه‌ها کاملا balanced هستند و نیازی به استفاده از روش‌هایی مانند Undersampling نیست.

در مرحله بعد به سراغ گرفتن میانگین و واریانس از ویژگی‌ها می‌رویم که در **شکل ۴** آمده است. همانطور که از **شکل ۴** مشخص است، میانگین طول کاسبرگ حدود ۶ سانتی متر، میانگین عرض کاسبرگ ۳ سانتی متر، میانگین طول گلبرگ حدود ۴ سانتی متر و میانگین عرض گلبرگ ۱ سانتی متر است. از لحاظ واریانس و پراکندگی ویژگی‌ها، ویژگی طول گلبرگ، طول کاسبرگ، عرض گلبرگ و عرض کاسبرگ به ترتیب بیشترین تا کمترین انحراف معیار را دارند. علاوه بر این بیشینه و کمینه این مقادیر نیز در **شکل ۴** آمده است.

در **شکل ۵** میانگین و میانه این ویژگی‌ها به تفکیک هر کلاس آمده است. از این شکل می‌توان برداشت که کلاس Iris-versicolor بیشترین طول کاسبرگ را دارد و در رتبه دوم Iris-virginica قرار دارد. نکته مهمی که از این نمودار می‌توان برداشت کرد این است که در ویژگی عرض گلبرگ چقدر سه کلاس با هم از نظر میانگین اختلاف دارد و می‌توان پیش‌بینی کرد که به کمک این ویژگی، می‌توان کلاس Iris-setosa را به راحتی از دو کلاس دیگر جدا کرد.



شکل ۳: بررسی بالانس بودن داده ها

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

شکل ۴: میانگین و واریانس ویژگی ها

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
mean	5.006	3.428	1.462	0.246	0.0
target_name	setosa	versicolor	virginica		
setosa	5.006	3.428	1.462	0.246	0.0
versicolor	5.936	2.770	4.260	1.326	1.0
virginica	6.588	2.974	5.552	2.026	2.0
Next steps: View recommended plots					
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
med	5.0	3.4	1.50	0.2	0.0
target_name	setosa	versicolor	virginica		
setosa	5.0	3.4	1.50	0.2	0.0
versicolor	5.9	2.8	4.35	1.3	1.0
virginica	6.5	3.0	5.55	2.0	2.0

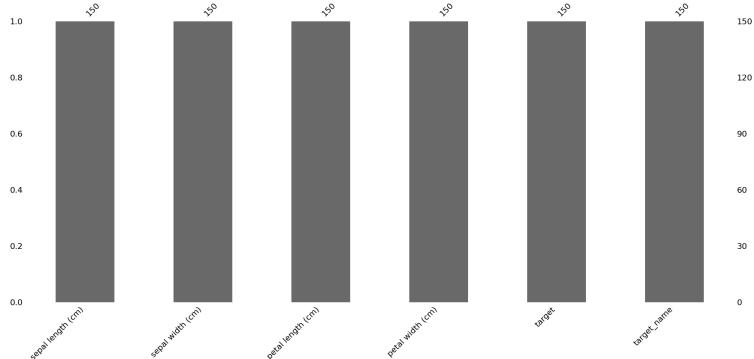
شکل ۵: میانگین و میانه ویژگی ها به تفکیک کلاس ها

در مرحله بعد داده های Nan را شناسایی می کنیم. از **شکل ۶** مشخص است که هیچ کدام از کلاس های ما داده Nan ندارد و ما value ای نداریم. این موضوع در **شکل ۷** به صورت بصری به کمک کتابخانه missingno نیز نشان داده شده. در **شکل ۸** مشخص است که طول و عرض کاسبرگ ها توزیع تقریبا نرمال با یک قله یا peak دارند در صورتی که توزیع طول و عرض گلبرگ ها یک توزیع Bimodel دارند به صورتی که دو قله در آن ها وجود دارد و با توجه به این موضوع احتمالاً این دو ویژگی از اهمیت بیشتری برخوردار باشند و می توانند حداقل دو کلاس از داده های ما را به خوبی جدا کنند.

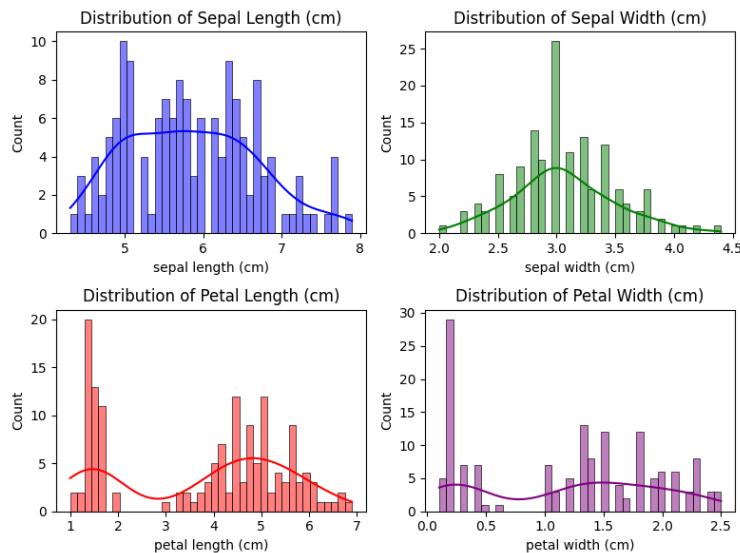


```
df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   sepal length (cm)    150 non-null   float64
 1   sepal width (cm)     150 non-null   float64
 2   petal length (cm)    150 non-null   float64
 3   petal width (cm)     150 non-null   float64
 4   target          150 non-null   int64  
 5   target name       150 non-null   object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

شکل ۶: شناسایی داده های Nan



شکل ۷: بصری سازی شناسایی داده های Nan



شکل ۸: نمایش توزیع ویژگی ها

در مرحله بعد با دسته بندی داده ها با توجه به هر کلاس، نمودار جعبه ای آن هارا رسم می کنیم که نگاهی دیگر به توزیع داده ها است. اینکار را به صورت [شکل ۹](#) انجام می دهیم و نتایج را در [شکل ۱۰](#) مشاهده می کنیم. مطابق [شکل ۱۰](#) نکته که در در نمودار میله ای بود نیز دیده می شود. یعنی اینکه با کمک ویژگی مربوط به گلبرگ، می توان به خوبی کلاس هارا از هم جدا کرد و این ویژگی ها می توانند ابزار قدرتمندی برای استفاده در مدل باشند. همانطور که دیده می شود Iris-setosa به خوبی از بقیه کلاس Iris-setosa جدا شدن است و انتظار می رود که با دقت ۱۰۰ درصد بتوانیم این کلاس را از بقیه کلاس ها جدا کنیم. دو کلاس دیگر از نظر ویژگی ها یمروط به کاسبرگ

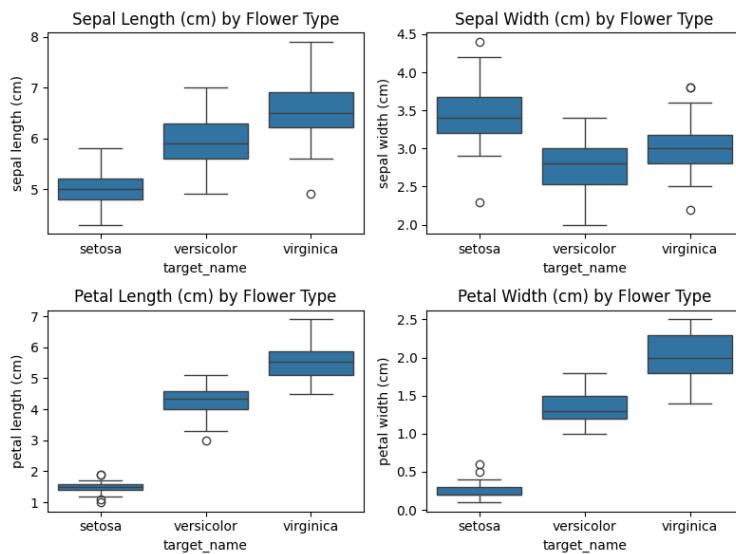


```

❶ fig, axs = plt.subplots(2, 2, figsize=(8, 6))
sns.boxplot(data=df, x='target_name', y='sepal length (cm)', ax=axs[0, 0])
axs[0, 0].set_title('Sepal Length (cm) by Flower Type')
sns.boxplot(data=df, x='target_name', y='sepal width (cm)', ax=axs[0, 1])
axs[0, 1].set_title('Sepal Width (cm) by Flower Type')
sns.boxplot(data=df, x='target_name', y='petal length (cm)', ax=axs[1, 0])
axs[1, 0].set_title('Petal Length (cm) by Flower Type')
sns.boxplot(data=df, x='target_name', y='petal width (cm)', ax=axs[1, 1])
axs[1, 1].set_title('Petal Width (cm) by Flower Type')
plt.tight_layout()
plt.show()

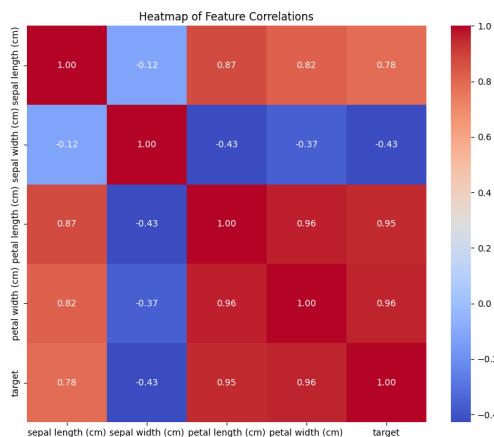
```

شکل ۹: کد نمودار جعبه ای



شکل ۱۰: نمودار جعبه ای داده ها

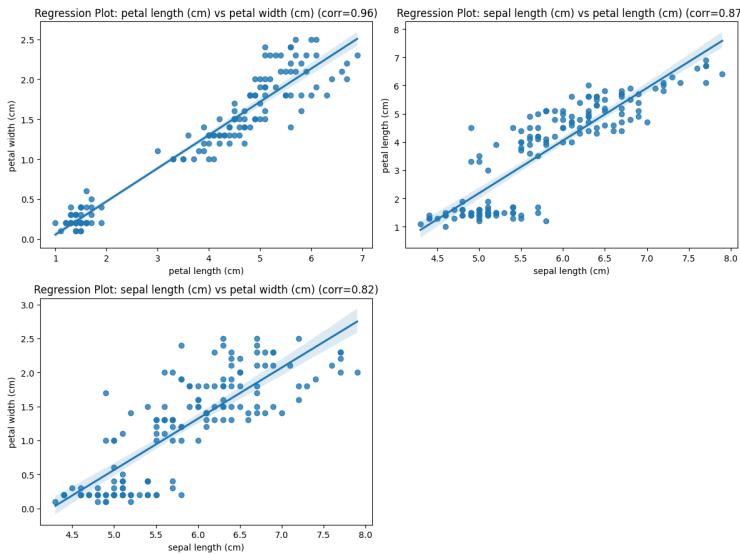
نزدیک زیادی به هم دارند ولی در ویژگی مربوط به گلبرگ می توان دید که این دو کلاس را می توان با دقت قابل قبولی از هم جدا کرد. در مرحله بعد در شکل ۱۱ ماتریس همبستگی مربوط به دیتاست آمده است. با توجه به شکل ۱۱ دیده می شود که بهترین ویژگی هایی که



شکل ۱۱: ماتریس همبستگی دیتاست Iris



با لیل ما بیشترین همبستگی را دارند به ترتیب عرض گلبرگ، طول گلبرگ و عرض کاسبرگ هستند. علاوه بر این دیده می شود ه دو ویژگی عرض گلبرگ و طول گلبرگ به ضریب  $0.96$  با یکدیگر هم بستگی خطی دارند. سه تا از بیشترین هم بستگی ها میان ویژگی ها در [شکل ۱۲](#) برای درک بهتر رسم شده است.



شکل ۱۲: سه همبستگی برتر در داده ها

در مرحله بعد از t-SNE استفاده می کنیم تا داده را بصری سازی کنیم. به طور کلی t-SNE یک روش غیرخطی برای به نمایش گذاشتن داده ها با ابعاد بالا است که از روش شباهت (similarity) بین داده ها استفاده می کند و با ادغام نظریه های احتمالاتی سعی می کند یک بصری سازی مناسب از داده انجام دهد. با انجام این الگوریتم میتوانیم کلاستر ها و الگوهایی که ممکن است در ابعاد بالاتر پنهان شوند را بینیم. در واقع با بصری سازی داده ها می توان outlier و anomaly مربوط به داده را بهتر مشاهده کرد. نکته منفی این روش این است که در ابعاد بالا می تواند از نظر توان محاسباتی بسیار مارا درگیر کند و در این حالات بهتر است از روش های خطی مانند PCA استفاده کنیم. در PCA الگوریتم سعی می کند جهت یا همان Principle Component هایی را پیدا کند که در آن واریانس داده ها ماکسیمایز می شود و این Component ها نسبت به هم عمود و uncorrelated هستند. در این الگوریتم ماتریس کوواریانس داده ها حساب می شود و سپس به کمک بردار های ویژه ها و مقادیر ویژه، داده به k بردار ویژه این فضای نگاشت می شود. الگوریتم t-SNE دو هایپرپارامتر مهم دارد که در [شکل ۱۳](#) آورده شده است. اولین هایپرپارامتر n-components است که تعداد بعدی است که میخواهیم دیتا را به آن نگاشت کنیم. هایپرپارامتر دوم perplexity است که تعداد نزدیکترین همسایگان که در سایر الگوریتم های یادگیری چندگانه استفاده می شود. در [شکل ۱۴](#) این بصری سازی به کمک t-SNE آورده شده.

علاوه بر این از PCA نیز در [شکل ۱۵](#) استفاده شده تا داده بصری سازی شود. توجه شود که این الگوریتم به صورت دستی پیاده شده مطابق روشی که بالاتر توضیح داده شد. ابتدا ماتریس کوواریانس داده های استاندار شده حساب شده و سپس مقادیر و بردارهای ویژه بدست آورده شده سپس از بزرگترین به کمترین مرتب شده و ۲ بردار ویژه بزرگ (دو بعد) انتخاب شده و به کمک این دو بردار ویژه دیتا مپ شده است. نتایج [شکل ۱۵](#) در [شکل ۱۶](#) آمدۀ است. مطابق [شکل ۱۴](#) و [شکل ۱۶](#) دیده می شود که کلاس Iris-setosa مطابق پیش یینی ها از توزیع ویژگی ها را می توان به راحتی از بقیه کلاس ها جدا کرد.

برای بررسی اینکه می توان از روش های کاهش ابعاد استفاده کرد یا نه از متدهای explained-variance-ratio. استفاده می کنیم تا بینیم تا کدام یک از Principal Components ها برای توصیف داده ها مناسب تر است (از نظر تعداد). سپس با چک کردن مقادیر مختلف Principal Components که می توان از ۱ تا  $3$  باشد. این نمودار در [شکل ۱۷](#) آمدۀ است. همانطور که از این نمودار مشخص است با



```
# t-SNE
from sklearn.manifold import TSNE

df_tsne = df.copy()

tsne = TSNE(n_components=2, perplexity=30, random_state=54)

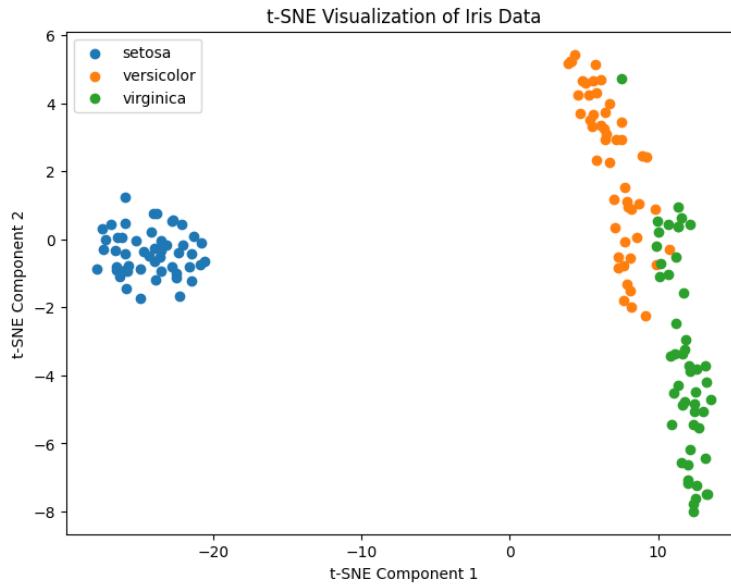
X_tsne = tsne.fit_transform(df[iris.feature_names])

df_tsne['tsne-2d-one'] = X_tsne[:, 0]
df_tsne['tsne-2d-two'] = X_tsne[:, 1]

plt.figure(figsize=(8, 6))
colors = ['C' + str(i) for i in range(len(iris.target_names))]
for target_name, color in zip(iris.target_names, colors):
    indices_to_plot = df_tsne['target_name'] == target_name
    plt.scatter(df_tsne.loc[indices_to_plot, 'tsne-2d-one'],
                df_tsne.loc[indices_to_plot, 'tsne-2d-two'],
                label=target_name)

plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.title('t-SNE Visualization of Iris Data')
plt.legend()
plt.show()
```

شکل ۱۳: کد مربوط به بصری سازی با t-SNE



شکل ۱۴: بصری سازی به کمک t-SNE

وجود دو ویژگی می توانیم ۸۱.۹۵ درصد از واریانس کل را توصیف می کند. و با ۳ ویژگی می توانیم ۹۹ درصد از واریانس را توصیف کنیم.

علاوه بر استناد به نمودار شکل ۱۱ می توان از ماتریس همبستگی در شکل ۱۷ می توان یکی از ویژگی هارا به علت همبستگی زیاد با یکی دیگر از ویژگی ها حذف کرد.



```
# PCA
X = df[iris.feature_names].values
X_mean = np.mean(X, axis=0)
X_std = np.std(X, axis=0)
X_standardized = (X - X_mean) / X_std

cov_matrix = np.cov(X_standardized, rowvar=False)

eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

sorted_index = np.argsort(eigenvalues)[::-1]
sorted_eigenvalues = eigenvalues[sorted_index]
sorted_eigenvectors = eigenvectors[:, sorted_index]

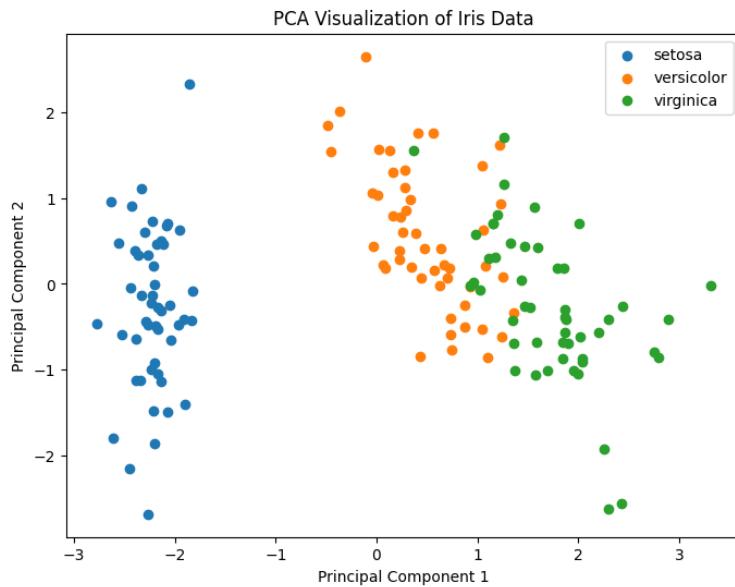
# Select the top k eigenvectors (here we select top 2 for 2D visualization)
k = 2
eigenvector_subset = sorted_eigenvectors[:, 0:k]
X_reduced = np.dot(X_standardized, eigenvector_subset)

df_pca = pd.DataFrame(data=X_reduced, columns=['PC1', 'PC2'])
df_pca['target'] = df['target']
df_pca['target_name'] = df['target'].map(dict(enumerate(iris.target_names)))

plt.figure(figsize=(8, 6))
colors = ['c' + str(i) for i in range(len(iris.target_names))]
for target_name, color in zip(iris.target_names, colors):
    indices_to_plot = df_pca['target_name'] == target_name
    plt.scatter(df_pca.loc[indices_to_plot, 'PC1'],
                df_pca.loc[indices_to_plot, 'PC2'],
                label=target_name)

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Visualization of Iris Data')
plt.legend()
plt.show()
```

شکل ۱۵: کد بصری سازی به کمک PCA



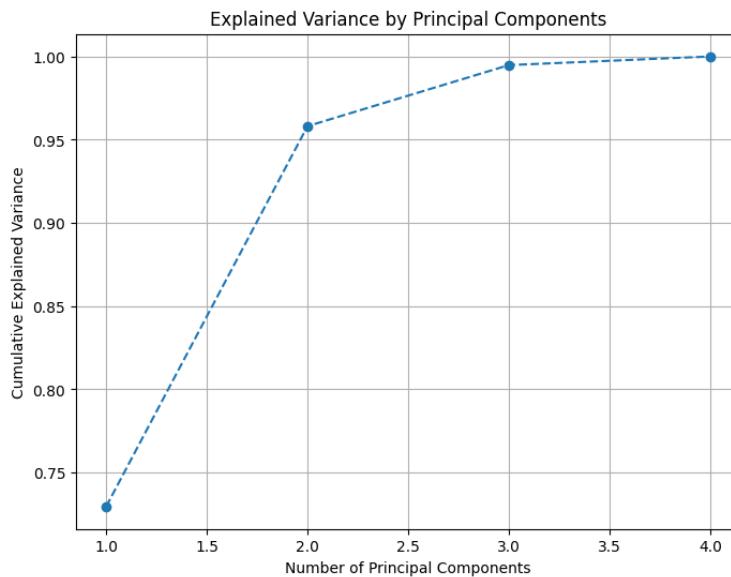
شکل ۱۶: بصری سازی به کمک PCA

## ۲.۲ بخش دو

ابتدا داده را به دو بخش آموزش و تست تقسیم می کنیم و ۲۰ درصد از داده ها را به تست اختصاص می دهیم. سپس داده ها را به کمک Standard Scaler می کنیم. سپس مطابق شکل ۱۸ مدل را با کردن خطی می سازیم.

در شکل ۱۹ نتایج طبقه بندی دیده می شود. دیده می شود که دقت کلی ۹۳ درصد است. ۸۵ درصد از داده هایی که طبقه بند در کلاس ۱ طبقه بندی کرده واقعا به این کلاس تعلق دارند و طبقه بند توانسته ۸۲ درصد از داده های کلاس ۲ را درست طبقه بندی کند. در بقیه موارد طبقه بند عملکرد بی نقص داشته است.

ماتریس در هم ریختگی نیز در شکل ۲۰ آورده شده است. همانطور که دیده می شود طبقه بند دو نمونه از کلاس ۲ را در کلاس ۱ طبقه بندی کرده و در سایر نمونه های تست به خوبی عمل می کند.



شکل ۱۷: واریانس توضیح داده شده با توجه به Principal Components

```
[36] from sklearn.svm import SVC
      svm_linear = SVC(kernel='linear', random_state=54)
      svm_linear.fit(X_train, y_train)

      y_pred = svm_linear.predict(X_test)
```

شکل ۱۸: ساخت طبقه بند SVM با کرنل خطی

```
[38] classification_rep_svm_linear = classification_report(y_test,y_pred)
      print(classification_rep_svm_linear)

      precision    recall  f1-score   support
      0       1.00     1.00     1.00      8
      1       0.85     1.00     0.92     11
      2       1.00     0.82     0.90     11

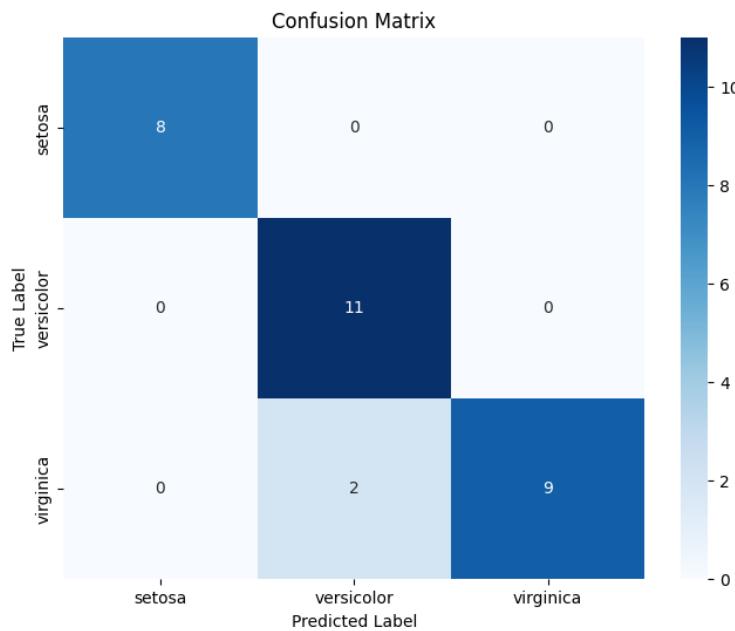
      accuracy                           0.93      30
      macro avg       0.95     0.94     0.94      30
      weighted avg    0.94     0.93     0.93      30
```

شکل ۱۹: نتایج طبقه بند SVM با کرنل خطی

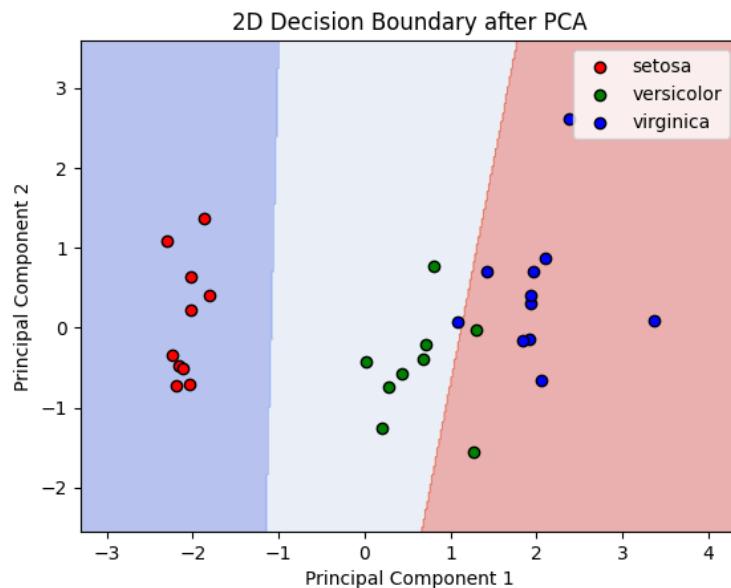
برای رسم مرز های تصمیم گیری می توانیم از PCA استفاده کنیم (برای مشاهده تحلیل چرایی استفاده از کاهش بعد به بخش ۱.۲ مراجعه شود) این ناحیه در [شکل ۲۱](#) آورده شده. همانطور که در [شکل ۲۱](#) دیده می شود، ناحیه تصمیم گیری به صورت خطی جدا شده است

## ۳.۲ بخش سوم

برای این بخش صرفا کافی است که یک حلقه for بنویسیم و ۱۰ تا مدل درست کنیم که درجه های poly در ان از ۱ تا ۱۰ متغیر است. این کار را به صورت [شکل ۲۲](#) انجام می دهیم و دقت با توجه هب هر درجه و همینطوری ماتریس درهم ریختگی را برای هر درجه نشان می دهیم. علاوه بر این در [شکل ۲۳](#) نمودار مربوط به دقت با توجه به درجه آمده است. در [شکل ۲۴](#) نیز ماتریس در هم ریختگی به ازای



شکل ۲۰: ماتریس درهم ریختگی طبقه بند SVM با کرنل خطی



شکل ۲۱: ناحیه تصمیم‌گیری طبقه بند SVM با کرنل خطی با کاهش بعد

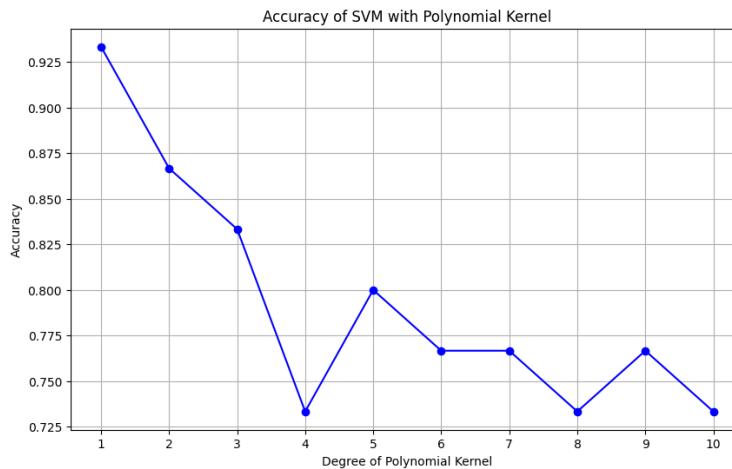
تمامی درجات از ۱ تا ۱۰ آمده است. با توجه به شکل ۲۳ دیده می‌شود که با زیاد شدن درجات poly قت طبقه بندی کم و کمتر می‌شد به طوری که در حالت درجه ۱ و به صورت خطی بهترین عملکرد را از مدل شاهد هستیم و با بال رفتن درجه میبینیم که دقت ما کمتر می‌شود به طوری که در حالت آخر و درجه ۱۰ دقت به ۶۷ درصد می‌رسد که اصلاً قابل قبول نیست. این موضوع را به کمک بصری سازی و کمک از PCA برای به تصویر کشیدن این عملکرد مدل طبقه بند و رسم ناحیه ها تصمیم‌گیری به کمک شکل ۲۵ و استفاده از مش و کانتور استفاده میکنیم توجه شود که برای چرازی استفاده از PCA به بخش ۱.۲ مراجعه شود. نتایج شکل ۲۵ به صورت شکل ۲۶ خواهد



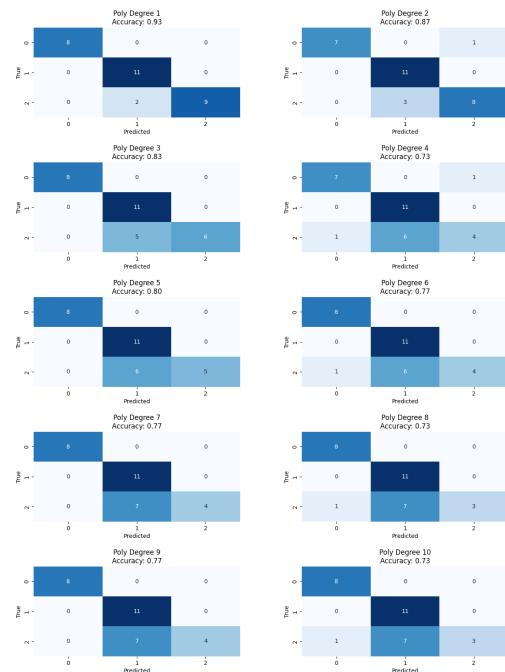
```
classifiers = [SVC(kernel='poly', degree=d, C=0.5) for d in range(1, 11)]
accuracies = []
degrees = range(1, 11)
conf_matrices = []

for clf in classifiers:
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
    conf_matrices.append(confusion_matrix(y_test, y_pred))
```

شکل ۲۲: کد ساختن ۱۰ طبقه بند با درجات ۱ تا ۱۰



شکل ۲۳: نمودار دقت با توجه به درجه کرnel



شکل ۲۴: ماتریس در هم ریختگی با توجه به درجه کرnel



```

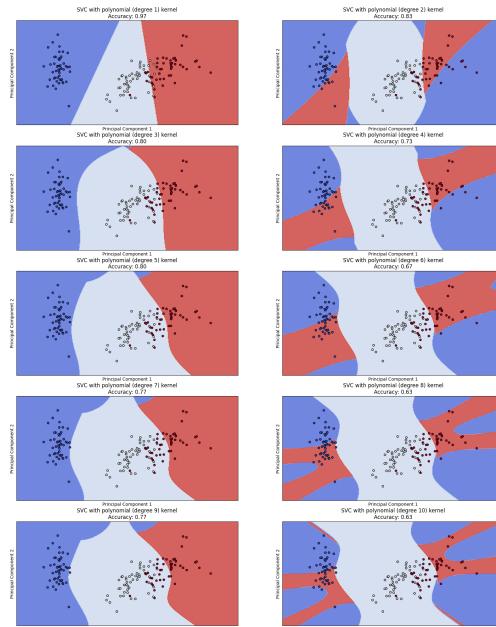
def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out

```

شکل ۲۵: توابع ایجاد مش و کاتور در حالت کاهش یافته

بود. همانطور که از این شکل پیداست، بهترین عملکرد جدایکردن کلاس‌ها و طبقه بندی در  $\text{degree} = 1$  رخ می‌دهد و با زیاد کردن درجه صرفاً به پیچیدگی مدل اضافه شده و مدل به سمت overfit شدن و یادگرفتن نویز‌ها می‌رود که هیچ! عملکرد طبقه بندی آن هم ضعیف‌تر می‌شود. به طوری کلی از [شکل ۲۶](#) مشخص است که با بالا رفتن درجه، انحنای نواحی تصمیم‌گیری بیشتر می‌شود و مدل بهتر می‌تواند پیچیدگی‌های مربوط به داده را یادگیرد. اما با توجه به نوع داده ما در اینجا، بهترین نتیجه با طبقه بندی خطی که از درجه ۱ بدست می‌آید حاصل می‌شود و در نتیجه بهترین دقت را در این مدل داریم.



شکل ۲۶: ناحیه تصمیم‌گیری طبقه بند با توجه به درجات مختلف poly



در مرحله بعد به کمک imageio و کنار هم قرار دادن نمودارهای شکل ۲۶ یک فایل gif از آن ها درست می کنیم. برای اینکار مطابق شکل ۲۷ تصویر هر حالت را ذخیره می کنیم سپس مطابق شکل ۲۸ این تصاویر را با فرمت gif به هم میچسبانیم و مدت هر فریم را ۲

```
image_files = []
for degree in range(1, 11):
    clf = SVC(kernel='poly', degree=degree, C=0.5)
    clf.fit(X_train, y_train)

    fig, ax = plt.subplots(figsize=(10, 8))
    plot.contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X1, X2, c=y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('Principal Component 1')
    ax.set_ylabel('Principal Component 2')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(f'SVC with polynomial (degree {degree}) kernel')

    filename = f"svc_poly_degree_{degree}.png"
    plt.savefig(filename)
    image_files.append(filename)
    plt.close()

images = []
for filename in image_files:
    images.append(imageio.imread(filename))
```

شکل ۲۷: کد ذخیره نمودارها به صورت عکس

ثانیه تعریف می کنیم و سپس آن را در گوگل درایو که اول آن را mount کرده بودیم ذخیره می کنیم. در نهایت از این لینک گیف (gif) می

```
# Save GIF to Google Drive
gif_path = '/content/drive/MyDrive/svc_poly_degrees.gif'
imageio.mimsave(gif_path, images, duration=2) # Duration in seconds for each frame
print("GIF saved to Google Drive at:", gif_path)
```

شکل ۲۸: کد ذخیره نمودارها به صورت گیف

توانید برای دسترسی به گیف استفاده کنید.

## ۴.۲ بخش چهارم

در این بخش از کد نوشته شده توسط حل تمرین که در اختیار مان قرار داده شده استفاده می کنیم و قسمتی از آن را گسترش می دهیم ولی بخش های اصلی مربوط به همان کد است. به طور کلی کد سه بخش دارد که شامل الگوریتم SVM، بخش بعدی شامل تعمیم این الگوریتم به یک الگوریتم چند کلاسه به روش One vs Rest و بخش آخر شامل بصری سازی الگوریتم است. برای تشکیل کلاس اول که مربوط به الگوریتم SVM می شود نیاز به تعریف کرنل داریم که به صورت شکل ۲۹ آن را تعریف می کنیم. مطابق این شکل ما کرنل poly را با ثابت ۱ و درجه به صورت پیش فرض ۳ تعریف کرده ایم. در قسمت بعد به سراغ ورودی های کلاس SVM می رویم که در

```
import cvxopt
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)
```

شکل ۲۹: تعریف کرنل های مختلف



شکل ۳۰ آمده است. دیده می وشد که باید داده ها و تارگت و نوع و پارامتر های کرنل را به این کلاس بدهیم. در نهایت این کلاس به ما مواردی از قبیل  $y$  و همینطور ثابت ها و نوع کرنل را برمی گرداند. علاوه بر این در این کلاس از کتابخانه CVX و از این بهینه ساز استفاده شده است.

```
def SVM1(X, X_t, y, C, kernel_type, poly_params=(1, 4), RBF_params=0.5, sigmoid_params=(1, 0.01)):
    kernel_and_params = (kernel_type, poly_params, RBF_params, sigmoid_params, C)
    n_samples, n_features = X.shape
```

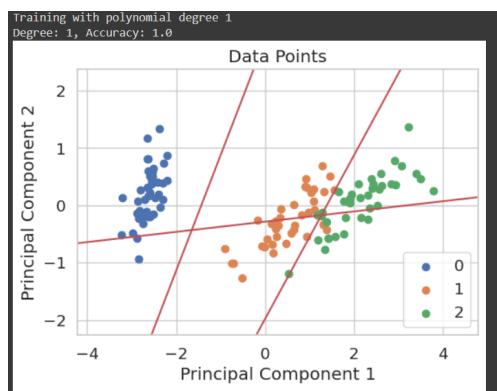
شکل ۳۰: ورودی های SVM

در مرحله خاصیت multi class classification به این الگوریتم اضافه می شود که گفته شد از الگوریتم One vs Rest استفاده می کند. این بخش نصب به کد های خام تعییری نداشته است. بخش بعدی بحث بصری سازی این الگوریتم است که به کمک کلاس visualize-multiclass-classification انجام می شود. در این بخش باید کرنل polynomial را به کلاس اضافه می کردیم که مطابق شکل ۳۱ انجام شده است.

```
if kernel_type == "polynomial":
    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
    xx, yy = np.mgrid[x_min:x_max, y_min:y_max]
    z = np.zeros(xx.shape)
    for i in range(len(classifiers)):
        for j in range(xx.shape[0]):
            for k in range(xx.shape[1]):
                sample_point = np.array([xx[j, k], yy[j, k]])
                decision_value, sv = classifiers[i].decision_function([sample_point])
                decision_value += 1 * np.sum(sv * sample_point, axis=1) + intercepts[i]
                decision_value -= bias
                z[j, k] = decision_value
    plt.contour(xx, yy, z, levels=[0], colors='r')
```

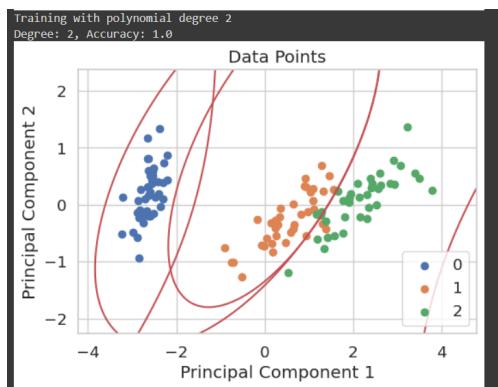
شکل ۳۱: تعریف مش و کانتور کرنل polynomial

در نهایت به کمک این سه کلاس ناحیه های تصمیم و دقت را با یک حلقه for رسم می کنیم. این کار در شکل ۳۲ تا شکل ۴۱ انجام شده است. در شکل ۴۲ نیز میزان دقت با توجه به هر درجه آورده شده که نتایجی مشابه قسمت قبل دارد، با پیچیده شدن طبقه بند و افزایش

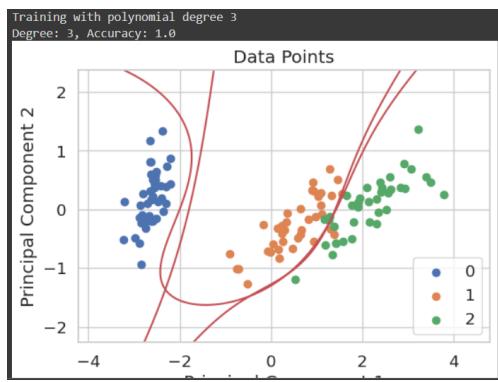


شکل ۳۲ : polynomial با درجه ۱ به صورت دستی

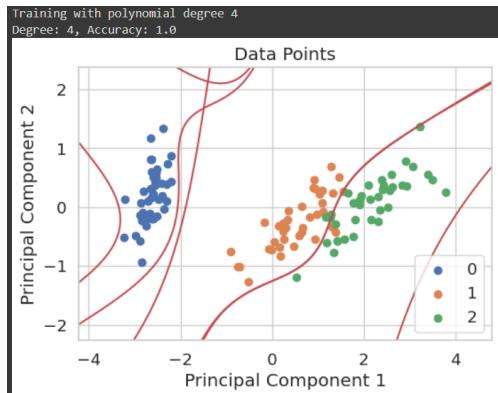
درجه، دقت افت محضوسی داشته است. در نهایت مانند بخش قبلی می توانید از این لینک گیف مربوطه را دانلود کنید.



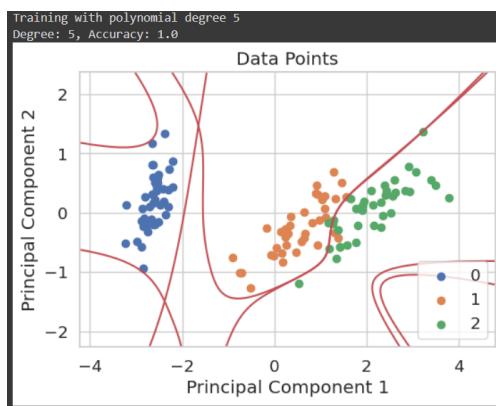
شكل ۳۳: با درجه ۲ به صورت دستی polynomial



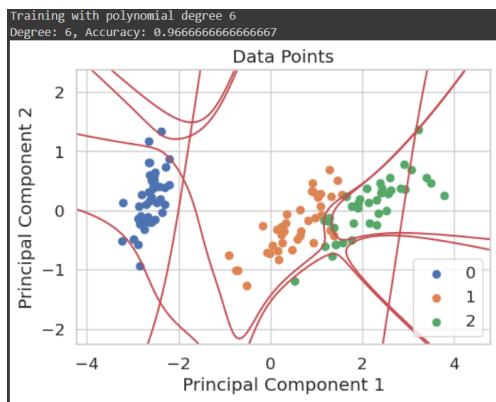
شكل ۳۴: با درجه ۳ به صورت دستی polynomial



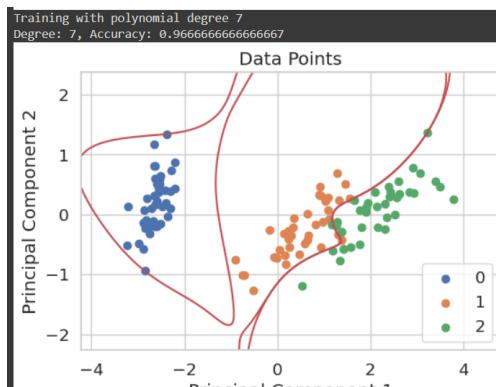
شكل ۳۵: با درجه ۴ به صورت دستی polynomial



شکل ۳۶: با درجه ۵ به صورت دستی polynomial



شکل ۳۷: با درجه ۶ به صورت دستی polynomial

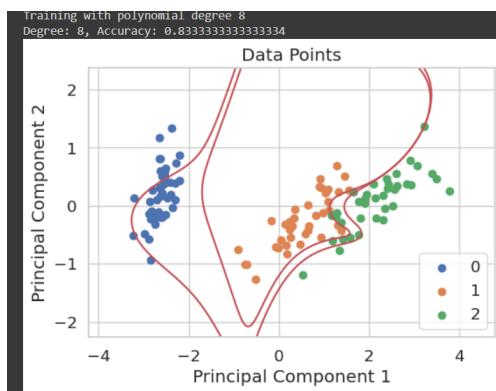


شکل ۳۸: با درجه ۷ به صورت دستی polynomial

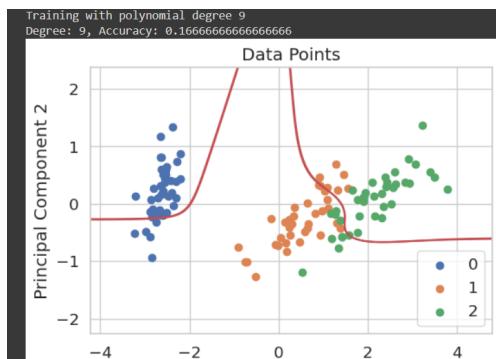
### سوال سوم ۳

#### ۱.۳ بخش اول

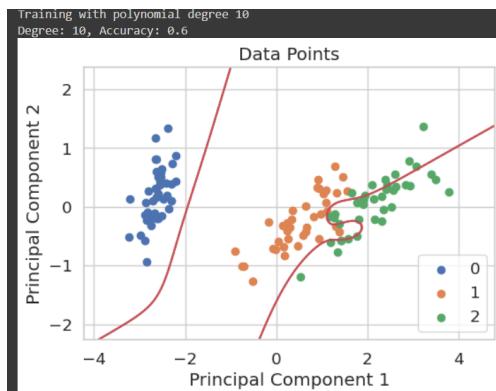
در کلام ساده، بزرگترین چالش مدل های شناسایی کلاهبرداری و تقلب بالانس نبودن داده ها در این حوزه است. می دانیم که مدل ها یهوش مصنوعی به همانقدری خوب عمل می کنند که وردی آن ها خوب باشد. در واقع ما اینجا داریم از داده آموزش با train صحبت ۴۰۲۲۳۸۵۴ علی مهرابی



شکل ۳۹: با درجه ۸ به صورت دستی polynomial

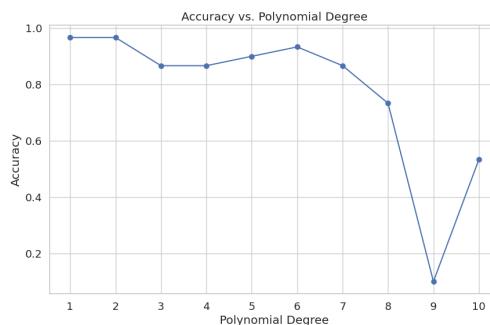


شکل ۴۰: با درجه ۹ به صورت دستی polynomial



شکل ۴۱: با درجه ۱۰ به صورت دستی polynomial

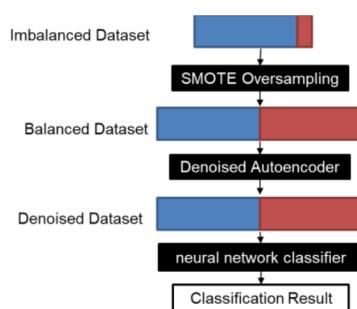
می کنیم. هر وقت داده train معمومتی و جامعیت خوبی داشته باشد می توانیم انتظار داشته باشیم که قابلیت پیش بینی این مدل هم به همان اندازه خوب است. این دقیقاً موضوعی است که ما در دیتا های imbalanced با آن طرف هستیم. فرض کنید نرخ کلاهبرداری در موضوع کارت های اعتباری ۱ درصد باشد (که در حالت های واقعی از این هم کمتر است) در این حالت ما با دادن داده های واقعی به مدل که تنها ۱ درصد از آن ها برچسب ۱ دارند، داریم به مدل این موضوع را القا می کنیم که در ۹۹ درصد موقع برچسب ۱ است و مدل های هوش مصنوعی که بر اساس اعداد و ارقام فکر می کنند و درکی از موضوع واقعی ندارند، تمامی داده های بعدی را با برچسب صفر



شکل ۴۲: نمودار دقت با توجه به درجه به صورت دستی

پیش بینی می کنند که این امری طبیعی است. در این موقع حتی استفاده از برخی متريک ها مانند دقت هم می تواند گول زنده باشد، به طوری که اگر مدل تمامی نمونه های بعدی را هم "عدم تقلب یا کلاهبرداری" پیش بینی کند در واقع دقت ۹۹ درصدی به ما می دهد. ولی ما به عنوان انسان که قدرت فکر دارد می دانیم که چنین مدلی هیچ وقت ارزش ندارد چون عملا برای ما کار تحلیل و پیش بینی را انجام نمی دهد و صرفا یک مدل بایاس است. در این مدل ها معمولا از ماتریس در هم ریختگی استفاده می شود تا عملکرد مدل به طور واقعی سنجیده شود. معیار های دیگری مانند recall هم می توانند کمک کننده باشند به طوری که به ما می گویند چند درصد از برجسب های واقعی، درست توسط مدل پیش بینی شده اند. در این حالت معمولا مشکلی که با آن سروکله میزیم پایین بودن درصد precision در کلاس minority است.

این مقاله برای مبارزه با این روش ابتدا می گوید که ما باید از روش های oversampling برای تولید داده مصنوعی در کلاس minority استفاده کنیم که این موضوع خود باعث به وجود آوردن نویز در داده ها می شود و اینکه این الگوریتم ها مانند SMOTE دقیقا چه کار با داده ما انجام می دهند و به چه شکلی داده به داده های اضافه می کنند به طور دقیق مشخص نیست (منظور این نیست که ما نمی دانیم الگوریتم چطور کار می کند بلکه نمی دانیم دقیقا چه بلافای سر داده می آورد). برای مقابله با این نویز ها، مقاله یک الگوریتم مبتنی بر اتوانکوادر (Autoencoder) معرفی می کند که در مرحله آموزش بعد از الگوریتم oversampling (در این مقاله SMOTE) قرار می گیرد که در نهایت به بهبود عملکرد مدل و طبقه بندی دوکلاسه کمک می کند. نقشه راه کلی این مقاله و روش ها پیاده سازی شده در شکل ۴۳ آورده شده. مطابق این شکل، ما در ابتدا یک دیتاست بسیار imbalanced داریم. در مرحله بعد به کمک SMOTE که یک مازول آماده است داده را بالانس می کنیم. سپس به کمک یک شبکه عصبی اتوانکوادر به داده ای که روی آن نویز گوسی سوار است داده را denoise می کنیم و در نهایت به کمک یک کمبل مبتنی بر شبکه عصبی طبقه بند خود را می سازیم. توجه شود که برای داده از الگوریتم کاهش بعد



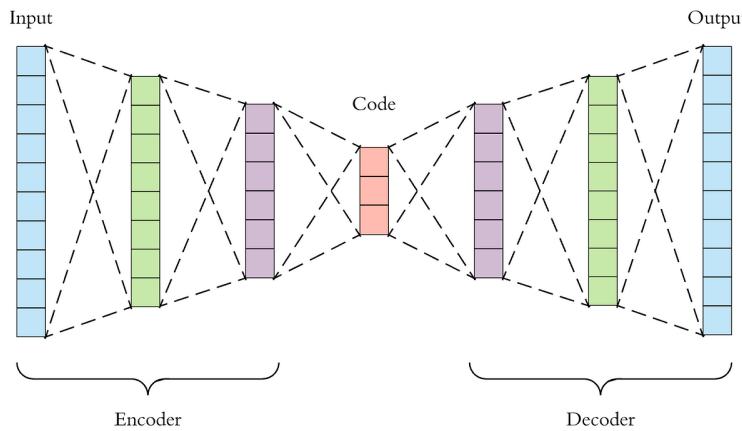
شکل ۴۳: روند کلی الگوریتم های به کارفته در مقاله

PCA استفاده شده تا در نهایت ۲۸ Principle Component داشته باشیم. البته برای دو ویژگی زمان و مقدار از PCA استفاده نشده است.



## ۲.۳ بخش دوم

همانطور که در [بخش ۱.۳](#) اشاره شد در این مقاله یک DAE معرفی شده که می تواند عملکردن کلی سیستم در مواجهه و طبقه بندی دیتاست های imbalanced را بهبود بخشد. شکل کلی این نوع از شبکه ها در [شکل ۴۴](#) آمده است. همانطور که از این شکل مشخص است، مشخصه اصلی اتوانکودر ها بر خلاف شبکه ای عصبی MLP این است که در اتوانکودر ها تعداد نورون ها در لایه اول و آخر برابر است. در واقع در استفاده از اتوانکودر ها هدف این است که ورودی ها بازسازی شوند و معمولاً در کاهش بعد به کار می رود.



شکل ۴۴: معماری کلی و بخش های یک اتوانکودر

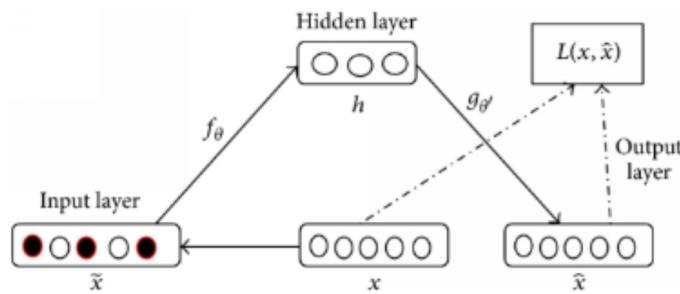
معماری شبکه ارائه شده در مقاله به صورت خلاصه به شرح زیر است

- لایه ورودی: لایه ورودی شامل ۲۹ نورون است که ویژگی های ورودی را نشان می دهد. این ویژگی ها شامل مولفه های اصلی و سایر ویژگی های تبدیل شده مجموعه داده هستند.
- لایه های پنهان: شبکه دارای چندین لایه پنهان متصل به طور کامل برای فرایندهای رمزگذاری و رمزگشایی است:
  - لایه پنهان اول: ۲۲ نورون
  - لایه پنهان دوم: ۱۵ نورون
  - لایه پنهان سوم: ۱۰ نورون
  - لایه پنهان چهارم: ۱۵ نورون
  - لایه پنهان پنجم: ۲۲ نورون
- لایه خروجی: لایه خروجی دارای ۲۹ نورون است تا با لایه ورودی مطابقت داشته باشد و هدف آن بازسازی داده های ورودی است.

**مطابق شکل ۴۵** لایه ورودی اتوانکودر دیتای همراه با نویز است و اگر دیتای اصلی را  $x$  بگیریم، هدف از تابع اتلاف مینیمموم کردن اختلاف خروجی و  $x$  است.

حال به سراغ مدل طبقه بند می رویم که آن هم از یک شبکه عصبی برای طبقه بندی استفاده می کند. پس از فرایند کاهش نویز، یک شبکه عصبی کاملاً متصل عمیق برای طبقه بندی نهایی استفاده می شود. معماری شامل موارد زیر است:

- لایه ورودی: لایه ورودی شامل ۲۹ نورون است که ویژگی های بدون نویز را نشان می دهد.



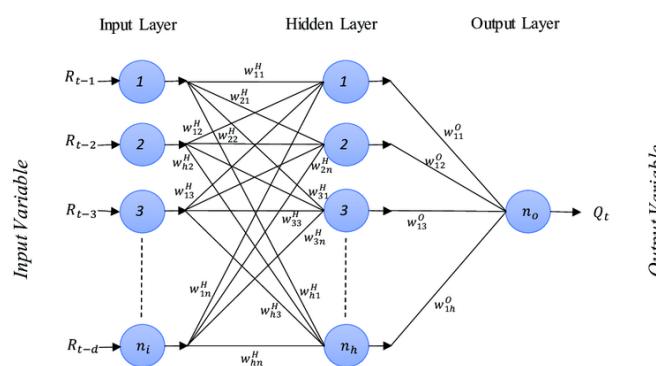
شکل ۴۵: لایه ورودی و خروجی و داده اصلی در ساختمان اتو انکودر

• لایه های پنهان:

- لایه پنهان اول: ۲۲ نورون
- لایه پنهان دوم: ۱۵ نورون
- لایه پنهان سوم: ۱۰ نورون
- لایه پنهان چهارم: ۵ نورون
- لایه پنهان پنجم: ۲ نورون

• لایه خروجی: لایه خروجی ازتابع فعال ساز سافت مکس با تابع هزینه آنتروپی متقطع برای طبقه بندی استفاده می کند و برای پیش بینی نهایی تراکنش های جعلی طراحی شده است.

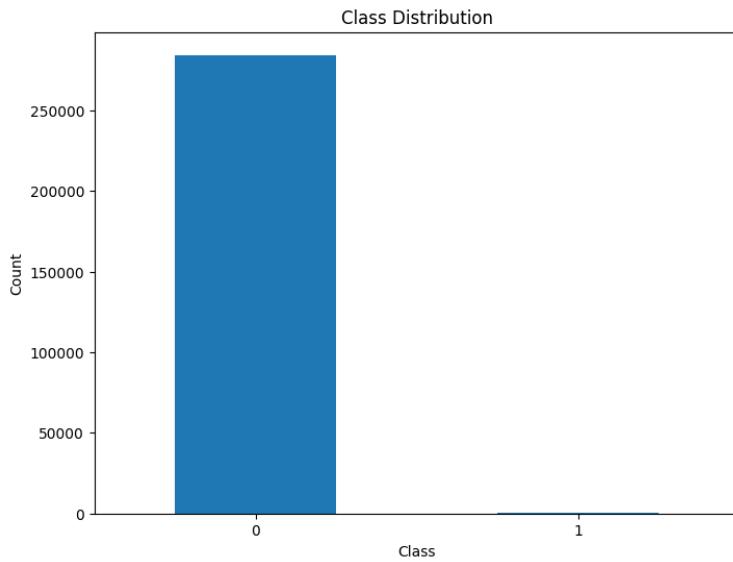
شبکه عصبی ای که برای طبقه بند طراحی می شود ساختمان متفاوتی از نظر تعداد نورون در هر لایه نسبت به اتو انکودر دارد و به طور کلی می توان آن را به صورت شکل ۴۶ توصیف کرد.



شکل ۴۶: نمایش توصیفی از لایه ها و نورون ها در شبکه عصبی طبقه بند

### ۳.۳ بخش سوم

ابتدا به کمک دستورز gdown داده را از گوگل درایو فراخوانی می کنیم. مطابق گفته مقاله ستون Time را حذف می کنیم و ستون Amount را به کمک StandardScaler اسکیل می کنیم. در [شکل ۴۷](#) توزیع داده برچسب نشان داده شده است. می بینیم که دیتاست به شدت روی لیل ۱ یا ۰ imbalanced است. به طور دقیق تر ما ۲۸۴۳۱۵ برچسب صفر داریم و ۴۹۲ هزارتا برچسب ۱ که نشان دهنده نامتعادل بودن برچسب است.



شکل ۴۷: نمایش نامتعادل بودن برچسب های دیتاست کارت اعتباری

در مرحله بعد دیتاست را به سه دسته آموزش، اعتبارسنجی و ارزیابی تقسیم می کنیم که نتایج آن در [شکل ۴۸](#) آمده است. دیده می شود که ۶۰ درصد از داده ها مربوط به آموزش هستند و ۴۰ درصد باقی مانده به طور مساوی بین اعتبارسنجی و ارزیابی تقسیم شده اند.

```
(('x_train_shape': (170883, 29),
  'x_val_shape': (56962, 29),
  'x_test_shape': (56962, 29),
  'y_train_shape': (170883,), 
  'y_val_shape': (56962,), 
  'y_test_shape': (56962,), 
  'train_percent': 59.99957866204131,
  'val_percent': 20.000210668979342,
  'test_percent': 20.000210668979342})
```

شکل ۴۸: تقسیم داده به آموزش، اعتبارسنجی و ارزیابی

در مرحله بعد نیاز است تا از الگوریتم SMOTE که مطابق گفته مقاله از ماذول آماده استفاده شده استفاده گردد. این الگوریتم با توجه به تعداد همسایه های نزدیک یک داده بین نقطه و همسایه اش به صورت رندوم تولید می کند که می توان این تعداد را به عنوان یک هایپر پارامتر تنظیم کرد. در [برنامه ۱](#) این بخش نوشته شده است. توجه شود که هایپر پارامتر sampling strategy می تواند یک عدد یا استرینگ باشد که در اینجا به معنای over sample کرد کلاس اقلیت است تا حدی که تعداد نمونه ها در هردو کلاس برابر شود. علاوه بر این، مطابق گفته مقاله این الگوریتم صرفا روی داده آموزش اجرا می شود.

نتایج [برنامه ۱](#) در [شکل ۴۹](#) آمده است و دیده می شود که داده کاملاً نامتعادل شده است.

در مرحله به صورت [برنامه ۲](#) برچسب را One Hot Encode می کنیم.



```

1 from imblearn.over_sampling import SMOTE
2 smote = SMOTE(sampling_strategy='minority', random_state = 54)
3 X_train, y_train = smote.fit_resample(X_train, y_train)

```

Code 1: SMOTE algorithm



شکل ۴۹: نتیجه اعمال الگوریتم SMOTE

سپس به صورت [برنامه ۳](#) به داده نویز گوسی اضافه می کنیم تا اتوانکودر را آموزش دهیم. یک نویز با شدت ۲۰ درصد داده ها به آن ها اضافه می کنیم که این هایپرپارامتر قابل تنظیم است و می تواند کم و زیاد شود. سپس به صورت [برنامه ۴](#) یک اتوانکودر مطابق مشخصات ذکر شده در مقاله می سازیم و به کمک checkpoint بهترین وزن هارا مطابق خواسته سوال ذخیره می کنیم تا در انتها آن را فراخوانی کنیم. این شبکه دارای تابع فعال ساز در لایه های پنهان ReLu و تابع فعال ساز لایه خروجی sigmoid است. برای بهینه سازی از adam اسفتاده شده و معیار خطای mse است.

در سه ایپاک نهایی نتایج به صورت [برنامه ۵](#) است.

از این شبکه برای denoise کردن استفاده می شود و خروجی آن به ورودی [برنامه ۶](#) داده می شود که طبقه بند ما است. توجه شود که مطابق گفته مقاله از soft max به عنوان تابع فعال ساز لایه آخر استفاده شده است و همانطور از categorical crossentropy به عنوان تابع هزینه استفاده شده و بهینه ساز همان adam است. به کمک [برنامه ۷](#) می توان بهترین ضرایب را بازگرداند.

```

1 from tensorflow.keras.utils import to_categorical
2 y_train = to_categorical(y_train, num_classes=2)

```

Code 2: label encoding



```

1 noise_factor = 0.2
2 X_train_noisy = X_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train.shape)
3 X_val_noisy = X_val + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_val.shape)
4 X_test_noisy = X_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_test.shape)

```

Code 3: adding Gaussian noise

```

1 input_dim = X_train.shape[1]
2
3 input_layer = Input(shape=(input_dim,))
4 encoded = Dense(22, activation='relu')(input_layer)
5 encoded = Dense(15, activation='relu')(encoded)
6 encoded = Dense(10, activation='relu')(encoded)
7 decoded = Dense(15, activation='relu')(encoded)
8 decoded = Dense(22, activation='relu')(decoded)
9 output_layer = Dense(input_dim, activation='sigmoid')(decoded)
10
11 autoencoder = Model(input_layer, output_layer)
12 autoencoder.compile(optimizer='adam', loss='mse')

```

Code 4: auto encoder

```

1 Epoch 48/50
2 1333/1333 [=====] - 4s 3ms/step - loss: 13.8506 - val_loss: 0.8654
3 Epoch 49/50
4 1333/1333 [=====] - 4s 3ms/step - loss: 13.8505 - val_loss: 0.8645
5 Epoch 50/50
6 1333/1333 [=====] - 3s 2ms/step - loss: 13.8505 - val_loss: 0.8651

```

Code 5: auto encoder last 3 epochs

```

1 classifier_input = Input(shape=(input_dim,))
2 x = Dense(22, activation='relu')(classifier_input)
3 x = Dense(15, activation='relu')(x)
4 x = Dense(10, activation='relu')(x)
5 x = Dense(5, activation='relu')(x)
6 x = Dense(2, activation='softmax')(x)
7
8 classifier = Model(classifier_input, x)
9 classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

```

Code 6: neural network classifier

```

1 classifier.load_weights('best_classifier.h5')

```

Code 7: neural network classifier best weights



## ۴.۳ بخش چهارم

به کمک برنامه ۸ می توانیم نتایج را بدست آوریم.

```

1 # Load the best classifier model
2 classifier.load_weights('best_classifier.h5')
3
4 # Predict on the denoised test set
5 y_test_pred = classifier.predict(X_test_denoised)
6 y_test_pred_classes = np.argmax(y_test_pred, axis=1)
7
8 y_test_true_classes = np.argmax(y_test, axis=1)
9
10 # Calculate metrics
11 conf_matrix = confusion_matrix(y_test_true_classes, y_test_pred_classes)
12 accuracy = accuracy_score(y_test_true_classes, y_test_pred_classes)
13 f1 = f1_score(y_test_true_classes, y_test_pred_classes)
14 recall = recall_score(y_test_true_classes, y_test_pred_classes)
15 precision = precision_score(y_test_true_classes, y_test_pred_classes)
16
17 # Print metrics
18 print(f'Accuracy: {accuracy*100:.2f}%')
19 print(f'F1 Score: {f1:.2f}')
20 print(f'Recall: {recall:.2f}')
21 print(f'Precision: {precision:.2f}')
22
23 print(classification_report(y_test_true_classes, y_test_pred_classes))
24 # Plot confusion matrix
25 plt.figure(figsize=(8, 6))
26 sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
27 plt.title('Confusion Matrix')
28 plt.xlabel('Predicted')
29 plt.ylabel('Actual')
30 plt.show()

```

Code 8: metrics on test data

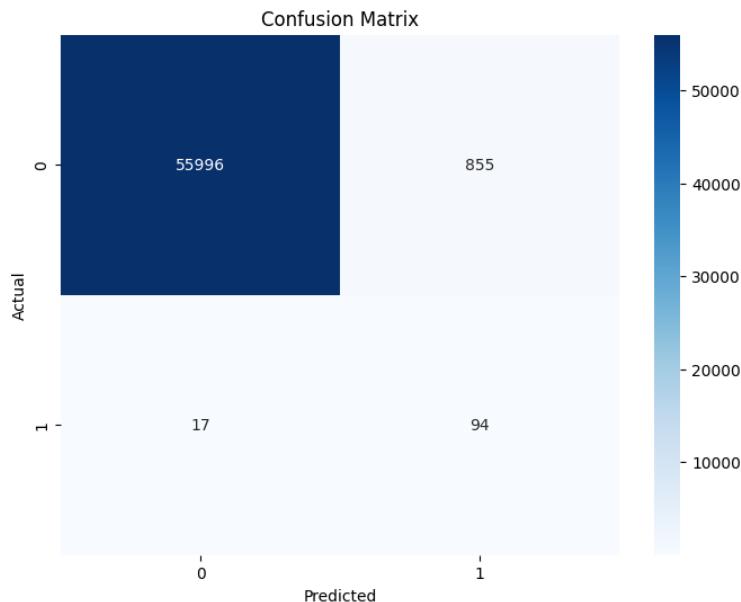
نتایج برنامه ۸ در شکل ۵۰ آمده است. ماتریس در هم ریختی مربوط به مدل نیز در شکل ۵۱ آمده است.

اما در پاسخ به سوالی که مطرح شده همانطور که در بخش ۱.۳ اشاره شد و مثال هم آورده شد، استفاده از Accuracy به تهایی کمکی به ما نمی کند. چون اگر ۹۹ درصد داده ها متعلق به کلاس ۰ باشند و ۱ درصد متعلق به کلاس ۱ و مدل ما تمامی ورودی های بعدی را به عنوان داده متعلق به کلاس ۰ پیش بینی کند، ۹۹ درصد دقت دارد و این عدد بالا نشان دهنده خوبی مدل ما نیست. در واقع مدل ها هیچ پیش بینی نمی کنند و خروجی مدل ما همواره مشخص خواهد بود. می توان از معیار های دیگری مانند precision که به ما می گوید چند درصد از پیش بینی های ماشین ما واقعاً متعلق به آن کلاس واقعی است. معیار دیگر recall است که به ما می گوید مدل توانسته چند درصد از نمونه های مربوط به یک کلاس را درست پیش بینی کند. اگر recall ما در کلاس اقلیت زیاد باشد یعنی مدل ما توانسته داده های



Accuracy: 98.47%
F1 Score: 0.18
Recall: 0.85
Precision: 0.10
precision recall f1-score support
0 1.00 0.98 0.99 56851
1 0.10 0.85 0.18 111
accuracy 0.98 56962
macro avg 0.55 0.92 0.58 56962
weighted avg 1.00 0.98 0.99 56962

شکل ۵۰: نتایج طبقه بندی



شکل ۵۱: ماتریس در هم ریختگی طبقه بند به کمک لایه Auto encoder

اقلیت را به خوبی تشخیص و طبقه بندی کند و در مسائل imbalanced ما معمولاً دنبال زیاد کردن این معیار هستیم. معیار f1 score هم برآیند این دو معیار است که با رسیدن به یک نقطه بهینه، این متريک زیاد خواهد شد. نگاه و قضاؤت از روی ماتریس در هم ریختگی نیز یکی از روش هایی است که در این مسائل بکار می رود تا مدل را بر اساس آن بهبود ببخشیم.

### ۵.۳ بخش پنجم

ما در این بخش دو آستانه را اصطلاحاً sweep می کنیم.

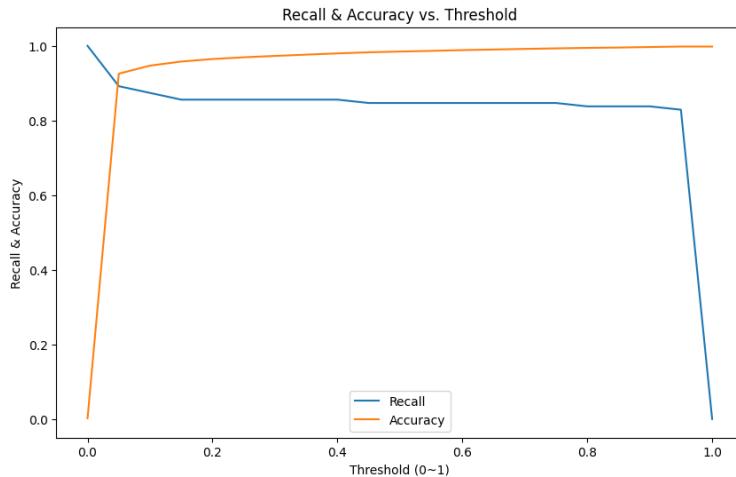
مورد اول تغییر thresh hold تصمیم گیری طبقه بند است. [برنامه ۹](#) آستانه های درنظر گرفته شده را نشان می دهد.

```
1 thresholds = np.arange(0.0, 1.05, 0.05)
```

Code 9: different thresh holds for the model



در شکل ۵۲ دقیق recall با توجه به آستانه‌ها مختلف رسم شده است. دیده می‌شود که باز یاد شدن آستانه مدل، مقدار accuracy زیاد می‌شود و recall به صورت جزئی کاهش می‌یابد.



شکل ۵۲: تغییرات Recall و Accuracy با تغییر آستانه مدل

حالت بعدی این است که ما sampling strategy را در الگوریتم SMOTE تغییر دهیم. اینکار را به صورت برنامه ۱۰ انجام میدهیم و در هر مرحله باید مدل را train کنیم.

```

1 sampling_strategies = [0.1, 0.25, 0.5, 0.75]
2
3 results = []
4
5 for strategy in sampling_strategies:
6     # Apply SMOTE with the current sampling strategy
7     smote = SMOTE(sampling_strategy=strategy, random_state=64)
8     X_train_res, y_train_res = smote.fit_resample(X_train, np.argmax(y_train, axis=1))
9     y_train_res = to_categorical(y_train_res, num_classes=2)

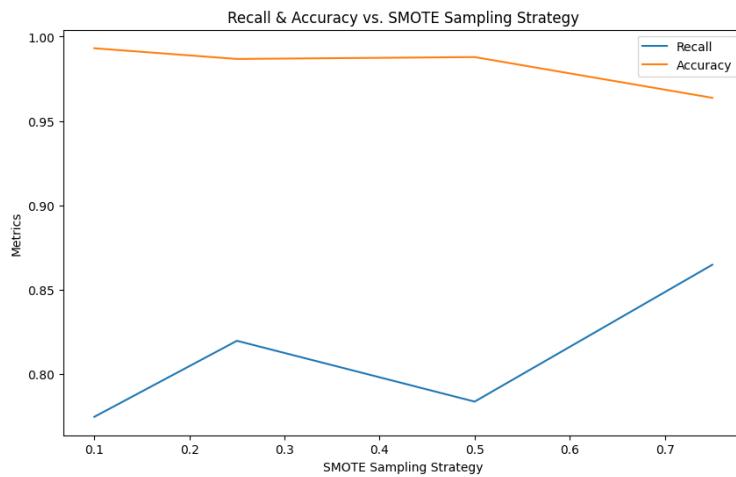
```

Code 10: different thresholds for SMOTE

و نتایج به صورت شکل ۵۳ و شکل ۵۴ است. در این حالت دیده می‌شود که با زیاد کردن آستانه SMOTE عملکرد سیستم بهبود می‌یابد. دیده می‌شود که نتایج مطابق با مقاله است.

### ۶.۳ بخش ششم

در این قسمت الگوریتم SMOTE را اجرا نمی‌کنیم و همینطور شبکه عصبی auto encoder هم حذف می‌کنیم. در واقع عمل denoise کردن داده‌ها و بالانس کردن آن‌ها نیز رخ نمی‌دهد. با اعمال تغییرات ذکر شده نتایج به صورت شکل ۵۵ و شکل ۵۶ است. مطابق این نتایج، دیده می‌شود که recall می‌شود و علاوه بر این دقیق recall می‌شود که به معنای گمراه شدن مدل به دلیل عدم بالانس بودن آن هاست. علاوه بر این از مقایسه شکل ۵۱ و شکل ۵۶ دیده می‌شود که تعداد نمونه‌هایی که مدل با SMOTE در کلاس ۱



شکل ۵۳: تغییرات Accuracy و Recall با تغییر آستانه SMOTE

SMOTE Threshold	Recall Rate	Accuracy
0	0.10	0.993118
1	0.25	0.986781
2	0.50	0.987904
3	0.75	0.963765

شکل ۵۴: تغییرات Accuracy و Recall با تغییر آستانه SMOTE به صورت جزئی

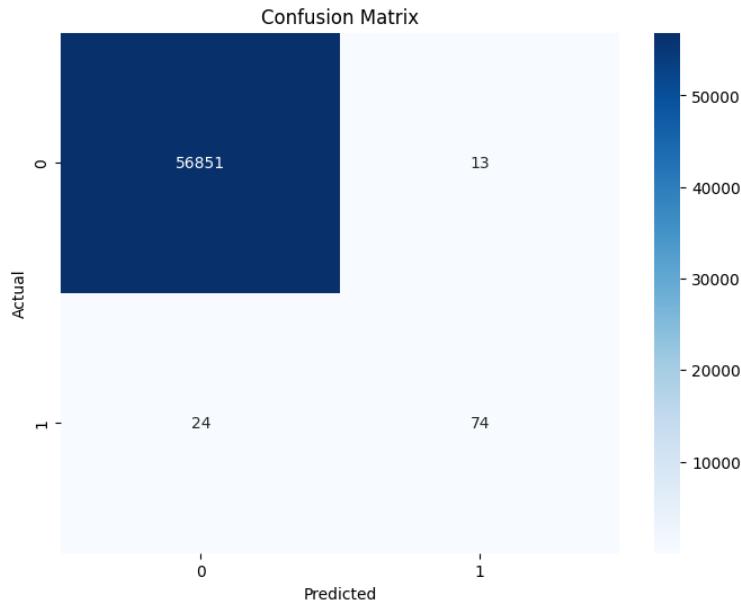
```
Accuracy: 99.94%
F1 Score: 0.80
Recall: 0.76
Precision: 0.85

      precision    recall   f1-score   support
0        1.00     1.00     1.00    56864
1        0.85     0.76     0.80      98

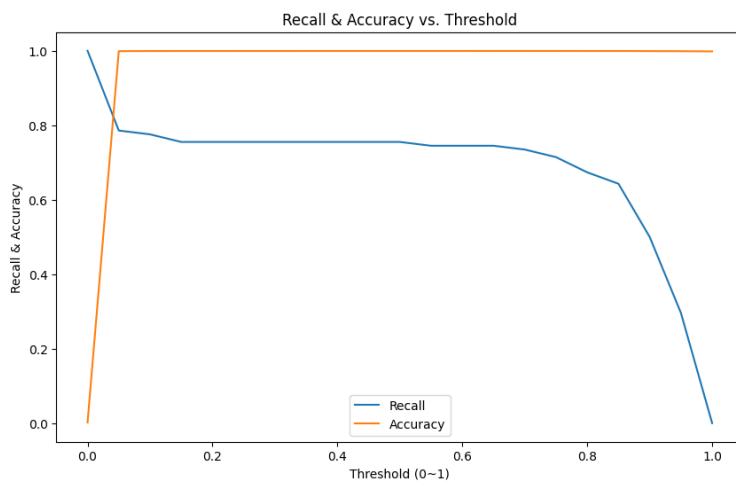
accuracy                           1.00    56962
macro avg                           0.93    56962
weighted avg                          1.00    56962
```

شکل ۵۵: نتایج بدون استفاده از SMOTE

طبقه بندی می کند ۱۰ برابر مدلی است که بدون SMOTE عمل می کند. می توانیم مانند قسمت قبلی استانه مدل را تغییر دهیم که نتیجه در این حالت به صورت شکل ۵۷ خواهد بود. دیده می شود که recall نسبت به شکل ۵۲ کمتر است و دقت مدل زودتر به ۱۰۰ نزدیک می شود که نشان می دهد فرآیند یادگیری مدل سریع تر رخ می دهد و این اثر مستقیم عدم انجام SMOTE است که generality مدل را کاهش می دهد.



شکل ۵۶: ماتریس در هم ریختگی بدون استفاده از SMOTE



شکل ۵۷: تغییرات recall و accuracy بدون SMOTE و auto encoder