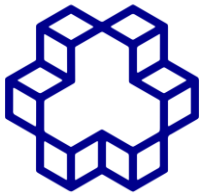
به نام خدا



دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

مینی پروژه سوم

Github link:

https://github.com/FatemehShokrollahiMoghadam

google drive link:

https://drive.google.com/drive/folders/13AzvgDBKOHcyuoBS8h4MY1iZfFxWNsr9?usp=sharing

نگارنده:

فاطمه شکراللهی مقدم

۴۰۲۰۷۳۶۴

بهار ۱۴۰۳

فهرست

پیشگفتار ۳

سوال اول ۴

آ ۴

ب ۹

ج ۱۴

د ۱۹

سوال سوم ۲۴

آ ۲۴

ب ۲۴

ج ۲۶

۱-ج پیش پردازش داده ۲۶

۲-ج Oversampling ۲۸

۳-ج اتوانکودر حذف نویز ۲۸

۴-ج مدل طبقه بند ۳۰

د ۳۳

ه ۳۵

و ۳۷

مراجع ۴۰

پیشگفتار

در صفحه اول گزارش لینگ گیت هاب و لینک گوگل درایو مربوط به نوت بوک های هر سوال آورده شده است.

در اینجا نیز لینک گوگل کولب نوت بوک هر سوال آورده می‌شود:

لینک نوت بوک سوال اول:

https://drive.google.com/file/d/1QeQ998b9IGYnm0JOiNav-aGfE7zAkmmD/view?usp=sharing

لینک نوت بوک سوال سوم:

https://colab.research.google.com/drive/1JG9ywL-Pq9f7xzfxlBN5kNwgLNvlLS-R?usp=sharing

سوال اول

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

آ

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

مجموعه داده زنبق نشان دهنده ۳ نوع گل زنبق (Setosa، VersicolourوVirginica) با ۴ ویژگی است: طول کاسبرگ، عرض کاسبرگ، طول گلبرگ و عرض گلبرگ. در این دیتاست انواع زنبق های نام برده شده با لیبل 0,1,2 مشخص شده اند.

پس از فراخوانی کتابخانه ها و مجموعه داده، با دستور ()info. به بررسی ابعاد و تعداد نمونه ها و همچنین وجود یا عدم وجود داده پوچ می‌پردازیم. با دستور ()head. نیز ۵ سط ابتدایی دیتاست قابل مشاهده است:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import datasets
import pandas as pd
from sklearn.datasets import load_iris
```

```
# Load the iris dataset
data = load_iris()
# Convert to pandas DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
# Add the target column
df['target'] = data.target
df.info()
df.head()
```

```
<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   target             150 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

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

همانطور که پیداست، مجموعه داده شامل ۱۵۰ نمونه و ۵ ستون دارد که ۴ ستون آن مربوط به ویژگی و ستون آخر برچسب نمونه ها را مشخص می‌کند. این دیتاست داده پوچی ندارد. با دستور groupby('target').size() تعداد نمونه در هر کلاس را محاسبه و بالانس بودن دیتاست را بررسی می‌کنیم:

```
df.groupby('target').size()
```

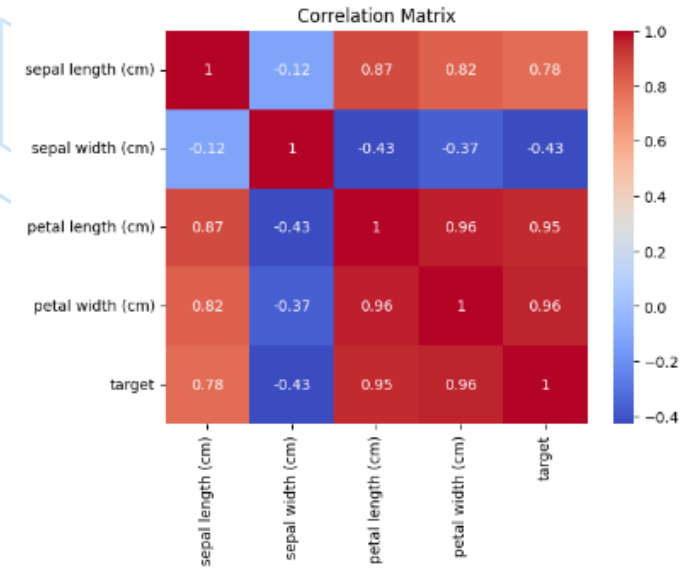
```
target
0    50
1    50
2    50
dtype: int64
```

تعداد نمونه در هر سه کلاس برابر و دیتاست بالانس است.

حال واریانس و میانگین را در هر ویژگی و ماتریس همبستگی را بدست می‌آوریم:

```
variances = df.var()
variances = variances.sort_values()
print("Variances of features:\n", variances)
means = df.mean()
means = means.sort_values()
print("means of features:\n", means)
# Correlation Matrix
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```

```
Variances of features:
 sepal width (cm)     0.189979
petal width (cm)      0.581006
target                0.671141
sepal length (cm)     0.685694
petal length (cm)     3.116278
dtype: float64
means of features:
 target                1.000000
petal width (cm)      1.199333
sepal width (cm)      3.057333
petal length (cm)     3.758000
sepal length (cm)     5.843333
dtype: float64
```



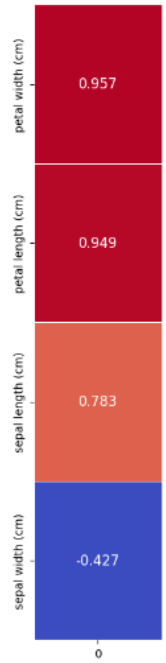
در بحث کاهش ابعاد، ویژگی مهم‌تر است که بیشترین واریانس را داشته باشد. نتایج واریانس و میانگین را از کم به زیاد نشان می دهند. همانطور که از نتایج پیداست، ویژگی petal length با اختلاف، بیشترین واریانس را در میان ویژگی ها دارد. همچنین طبق معیار فیشر دو ویژگی با بیشترین فاصله میانگین و کمترین مجموع واریانس دارای اطلاعات مفیدی در دیتاست هستند. با نگاه به میانگین ویژگی ها، میانگین های دو ویژگی sepal length و petal width بیشترین فاصله و کمترین مجموع واریانس ها را دارد. بنابراین با این تحلیل ویژگی های petal length، sepal length و petal width دارای بیشترین اطلاعات هستند.

در ماتریس همبستگی نیز دیده می شود که ویژگی هایpetal length و petal width دارای همبستگی زیادی با یکدیگر هستند. پس از آن petal length وsepal length و سپس petal width وsepal length دارای همبستگی شدید اند.

برای تحلیل بهتر ماتریس همبستگی، ویژگی ها با بیشترین میزان همبستگی با هدف را به ترتیب می نویسیم:ط

```
# Select columns to include in correlation matrix
cols = df.columns.tolist()
cols.remove('target')
# Calculate correlation matrix
corr_matrix = df[cols].corrwith(df['target']).sort_values(ascending=False)
# Create heatmap using seaborn
plt.figure(figsize=(2,10))
sns.heatmap(corr_matrix.to_frame(), annot=True, cmap='coolwarm', linewidths=0.5, annot_kws={"size": 12}, fmt='.3f', cbar=False)
# Rotate x-axis tick labels to be horizontal
plt.xticks(rotation=0)
```

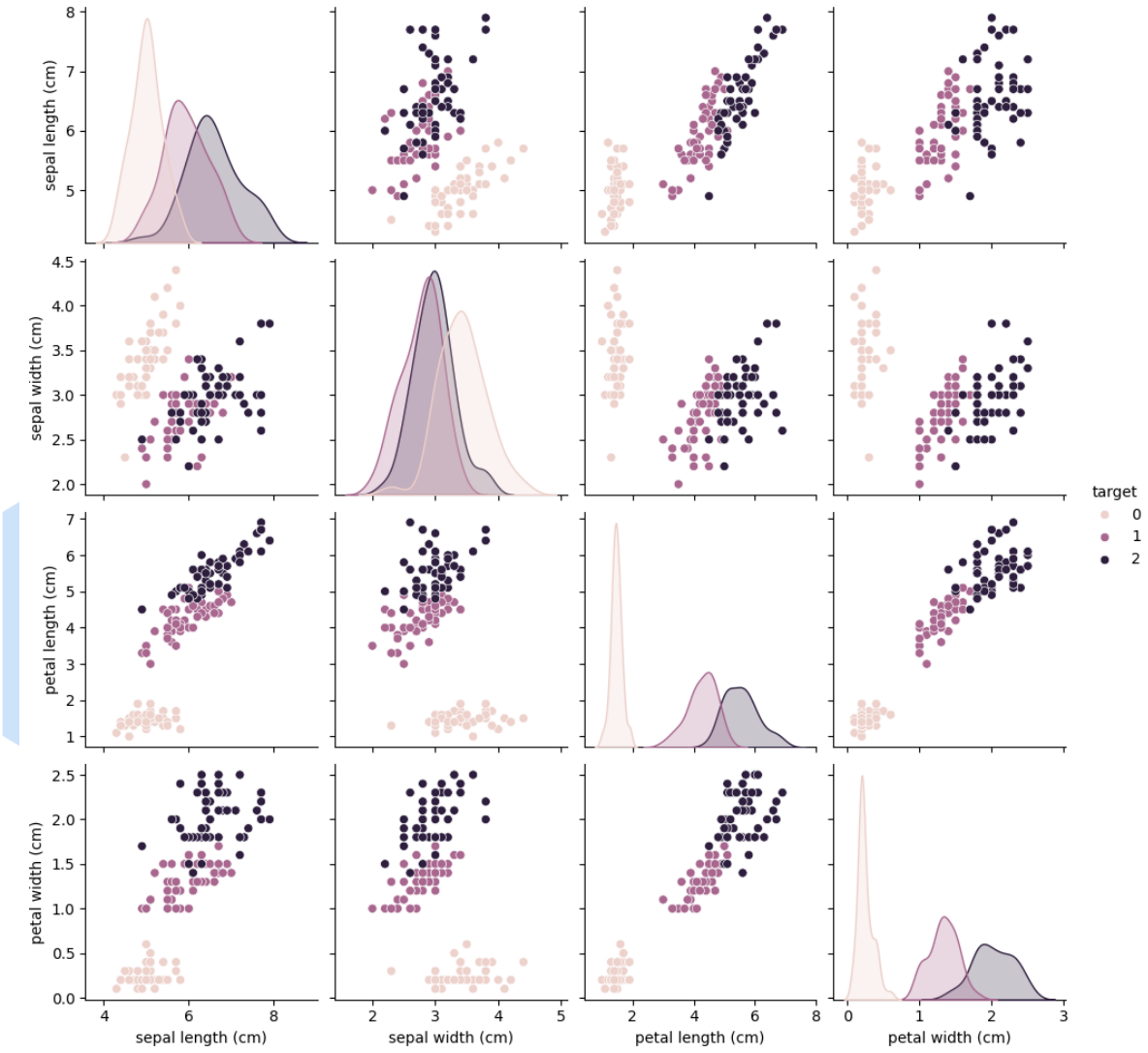
```
(array([0.5]), [Text(0.5, 0, '0')])
```



همانطور که پیداست، ویژگی petal width همبستگی شدیدی با target دارد. بنظر می‌آید با توجه به همبستگی این ویژگی با سایر ویژگی ها که پیش تر بحث شد و همبستگی آن با target کاهش ابعاد در

این دیتاست قابل انجام است. و همچنین می توان با حذف ویژگی sepal width که کمترین همبستگی را دارد، نتایج قابل قبولی بدست آورد.

برای نمایش بصری پراکندگی داده ها با توجه به ویژگی ها از کتابخانه seaborn و از دستور sns.pairplot(df,hue='target') استفاده می کنیم:



همانطور که از شکل بالا پیداست در ویژگی های petal width و petal length قابلیت تفکیک کلاس ها بیشتر از سایر ویژگی هاست.

روش t-SNE که در صورت سوال به آن اشاره شده، با کنار هم نگه‌داشتن نمونه‌های شبیه به هم و دور نگه‌داشتن نمونه‌های متفاوت از هم، ابعاد را کاهش میدهد. ابتدا داده ها را با دستور shuffle. مخلوط کرده و سپس با نسبت ۸۰٪و ۲۰٪ به آموزش و آزمون تقسیم می‌کنیم و از StandardScaler برای

استانداردسازی دیتا استفاده می کنیم. با استفاده از ماژول manifold کتابخانه sklearn ، t-SNE را پیاده سازی می کنیم:

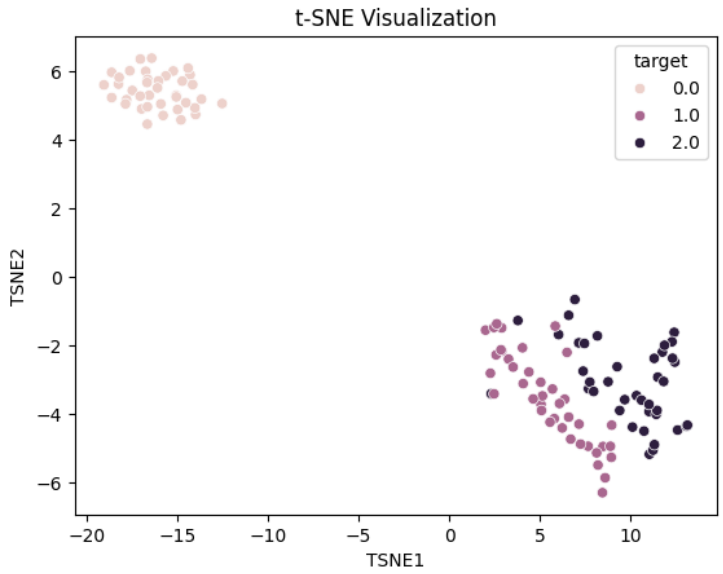
```
from sklearn.model_selection import train_test_split
np.random.seed(64)
array =df.values
np.random.shuffle(array)  # Shuffle the array
shuffled_df = pd.DataFrame(array, columns=df.columns)
X = shuffled_df.drop(columns='target').values
y = shuffled_df['target'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=64)
print('Train:', X_train.shape, y_train.shape, '\nTest:', X_test.shape, y_test.shape)
```

```
Train: (120, 4) (120,)
Test: (30, 4) (30,)
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Apply t-SNE
from sklearn.manifold import TSNE
np.random.seed(64)
tsne = TSNE(n_components=2, random_state=64)
tsne_results = tsne.fit_transform(X_train)
# Create a DataFrame with t-SNE results
tsne_df = pd.DataFrame(tsne_results, columns=['TSNE1', 'TSNE2'])
y_train_df = pd.DataFrame(y_train, columns=['target'])
final_tsne_df = pd.concat([tsne_df, y_train_df], axis=1)
ax=sns.scatterplot(x=final_tsne_df.iloc[:,0], y=final_tsne_df.iloc[:,1],hue='target', data=final_tsne_df, legend=True)
plt.title('t-SNE Visualization')
plt.show()
```

تعداد مولفه در t-SNE برابر دو در نظر گرفته شده و نتایج گزارش می شود:



همانطور که پیداست، کلاس صفر قابل تفکیک است اما برای ایجاد تفکیک پذیری در کلاس های اول و دوم می توان از روش های کاهش ابعاد استفاده کرد.

ب

با استفاده از الگوریتمSVM با هستهٔ خطی، داده ها را طبقه بندی کنید و ماتریس درهم ریختگی آن را بدست آورید و مرزهای تصمیم گیری را در فضای دوبعدی ترسیم کنید.

برای استفاده از SVM با هسته خطی از ماژولSVC با کرنل linear استفاده می کنیم. اینکار با استفاده ازLinearSVC نیز مقدور است.(در قسمت رسم مرز تصمیم گیری هر دو ماژول نمایش داده می شوند.)

مقدار c را ابتدا برابر یک قرار داده و خروجی احتمالاتی را True می گذاریم.

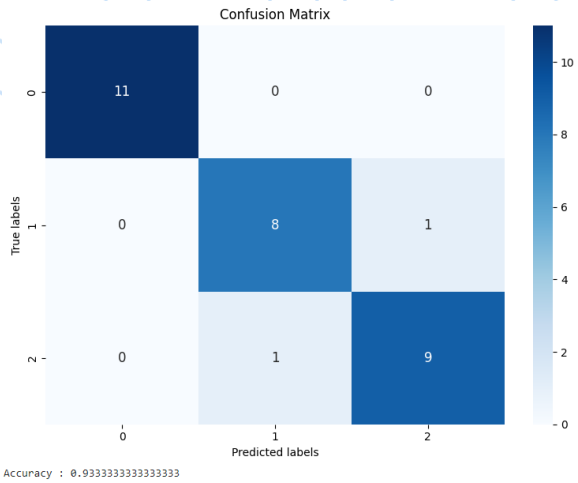
```
from sklearn.svm import SVC

model = SVC(C=1., kernel='linear', probability=True)
model.fit(X_train, y_train)
```

```
SVC(kernel='linear', probability=True)
```

حال با استفاده از متد predict داده های آزمون را به مدل داده و ماتریس در هم ریختگی و دقت مدل با معیار Accuracy گزارش می شود:

```
from sklearn.metrics import confusion_matrix, classification_report
y_pred = model.predict(X_test)
cf_matrix = confusion_matrix(y_test, y_pred)
# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 12})
# Get the axis to modify layout
plt.gca().set_ylim(len(np.unique(y_test)), 0)  # Fix for matplotlib 3.1.1 and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()
from sklearn.metrics import accuracy_score
print('Accuracy :',accuracy_score(y_test,y_pred))
```

نتایج به‌صورت زیر است:



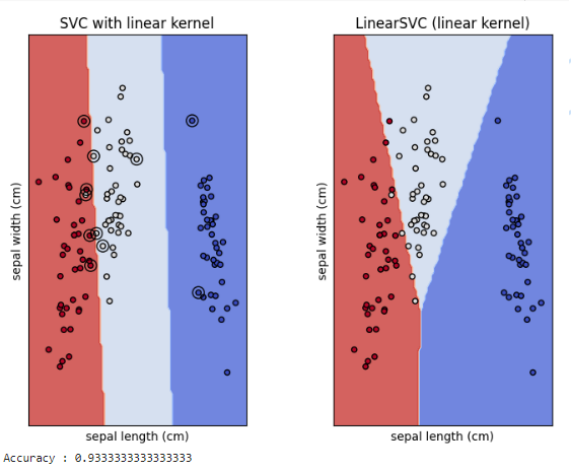

```
Accuracy : 0.9333333333333333
```

دو داده در کلاس درست قرار نگرفته اند و دقت مدل حدود ۹۳٪ است. برای رسم مرزهای تصمیم گیری ابتدا ابعاد را به دو بعد کاهش می دهیم. این کار با متد LDA انجام می شود چرا که در اینجا مسئله طبقه بندی است و به‌صورت Supervised انجام می شود و دیتاست لیبل دارد و ابعاد دیتاست بالا نیست.

سپس مدل ها با ماژول های SVC با کرنل خطی و LinearSVC با داده کاهش بعد یافته آموزش داده شده و مرز تصمیم با استفاده از DecisionBoundaryDisplay نمایش داده می‌شود.داده هایی که به عنوان بردار پشتیبان در نظر گرفته می‌شوند، متمایز شده اند. نهایتا دقت مدل روی داده های آزمون مجددا محاسبه شده است:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# Apply LDA to reduce dimensionality to 2
lda = LinearDiscriminantAnalysis(n_components=2)
X= lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)
C = 1.0  # SVM regularization parameter
models = (
    svm.SVC(kernel="linear", C=C),
    svm.LinearSVC(C=C, max_iter=10000),
)
models = (clf.fit(X, y_train) for clf in models)

# title for the plots
titles = (
    "SVC with linear kernel",
    "LinearSVC (linear kernel)",
)
# Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(1, 2, figsize=(8, 6))
plt.subplots_adjust(wspace=0.4, hspace=0.4)
X0, X1 = X[:, 0], X[:, 1]
for clf, title, ax in zip(models, titles, sub.flatten()):
    disp = DecisionBoundaryDisplay.from_estimator(
        clf,
        X,
        response_method="predict",
        cmap=plt.cm.coolwarm,
        alpha=0.8,
        ax=ax,
        xlabel=iris.feature_names[0],
        ylabel=iris.feature_names[1],
    )
    ax.scatter(X0, X1, c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
# Plot support vectors for SVC
    if isinstance(clf, SVC):
        sv = clf.support_vectors_
        ax.scatter(sv[:, 0], sv[:, 1], s=100, linewidth=1, facecolors='none', edgecolors='k')
# Plot support vectors
plt.show()
y_pred = model.predict(X_test)

print('Accuracy :',accuracy_score(y_test,y_pred))
```

نتایج به‌صورت زیر است:




```
Accuracy : 0.9333333333333333
```

برای عملکرد بهتر باید پارامتر تنظیم C را تغییر دهیم. برای یافتن بهترین مقدار C از الگوریتم GridSearch استفاده می کنیم:

مقادیر 0.1, 1, 10, 100, 1000 را برای c و مدل svc را معرفی می کنیم. نتایج این الگوریتم به‌صورت زیر است:

```
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'kernel': ['linear']}
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid.fit(X, y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV 1/5] END .............C=0.1, kernel=linear;, score=1.000 total time=   0.0s
[CV 2/5] END .............C=0.1, kernel=linear;, score=0.958 total time=   0.0s
[CV 3/5] END .............C=0.1, kernel=linear;, score=1.000 total time=   0.0s
[CV 4/5] END .............C=0.1, kernel=linear;, score=1.000 total time=   0.0s
[CV 5/5] END .............C=0.1, kernel=linear;, score=1.000 total time=   0.0s
[CV 1/5] END ...............C=1, kernel=linear;, score=1.000 total time=   0.0s
[CV 2/5] END ...............C=1, kernel=linear;, score=0.958 total time=   0.0s
[CV 3/5] END ...............C=1, kernel=linear;, score=1.000 total time=   0.0s
[CV 4/5] END ...............C=1, kernel=linear;, score=1.000 total time=   0.0s
[CV 5/5] END ...............C=1, kernel=linear;, score=1.000 total time=   0.0s
[CV 1/5] END ..............C=10, kernel=linear;, score=1.000 total time=   0.0s
[CV 2/5] END ..............C=10, kernel=linear;, score=0.958 total time=   0.0s
[CV 3/5] END ..............C=10, kernel=linear;, score=1.000 total time=   0.0s
[CV 4/5] END ..............C=10, kernel=linear;, score=1.000 total time=   0.0s
[CV 5/5] END ..............C=10, kernel=linear;, score=1.000 total time=   0.0s
[CV 1/5] END .............C=100, kernel=linear;, score=1.000 total time=   0.0s
[CV 2/5] END .............C=100, kernel=linear;, score=0.958 total time=   0.0s
[CV 3/5] END .............C=100, kernel=linear;, score=1.000 total time=   0.0s
[CV 4/5] END .............C=100, kernel=linear;, score=1.000 total time=   0.0s
[CV 5/5] END .............C=100, kernel=linear;, score=1.000 total time=   0.0s
[CV 1/5] END ............C=1000, kernel=linear;, score=1.000 total time=   0.0s
[CV 2/5] END ............C=1000, kernel=linear;, score=0.958 total time=   0.0s
[CV 3/5] END ............C=1000, kernel=linear;, score=1.000 total time=   0.0s
[CV 4/5] END ............C=1000, kernel=linear;, score=1.000 total time=   0.0s
[CV 5/5] END ............C=1000, kernel=linear;, score=1.000 total time=   0.0s
```

▸ GridSearchCV
▸ estimator: SVC
▸ SVC

بهترین مقدار پارامتر c مقدار 0.1 ارزیابی شده است:

```
grid.best_params_
```

```
{'C': 0.1, 'kernel': 'linear'}
```

```
grid.best_estimator_
```

▾ SVC
SVC(C=0.1, kernel='linear')

```
grid_predictions = grid.predict(X_test_lda)
print(confusion_matrix(y_test,grid_predictions))
```

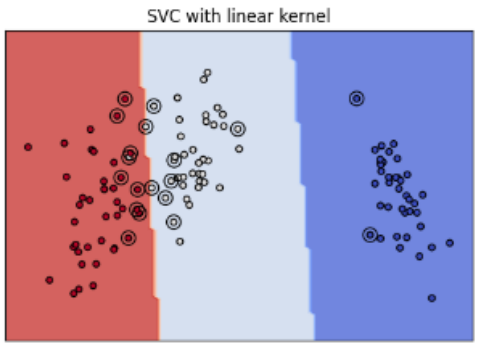
```
[[11  0  0]
 [ 0  8  1]
 [ 0  0 10]]
```

```
print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        11
         1.0       1.00      0.89      0.94         9
         2.0       0.91      1.00      0.95        10

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.96        30
weighted avg       0.97      0.97      0.97        30
```

مطابق نتایج بالا، ماتریس در هم ریختگی نشان می دهد تنها یک داده در کلاس نادرست قرار گرفته و دقت مدل به ۹۷٪ افزایش یافته است. جال با این پارامتر مرز تصمیم گیری را ترسیم می کنیم:

```
C = 0.1  # SVM regularization parameter
models = [
    SVC(kernel="linear", C=C)
]

# Train and fit models
for clf in models:
    clf.fit(X, y_train)

# Title for the plots
titles = (
    "SVC with linear kernel",
)

# Set-up 1x1 grid for plotting since you have only one model
fig, ax = plt.subplots(1, 1, figsize=(6, 4))
disp = DecisionBoundaryDisplay.from_estimator(
    models[0],  # Only one model in the list
    X,
    response_method="predict",
    cmap=plt.cm.coolwarm,
    alpha=0.8,
    ax=ax,
)

ax.scatter(X[:, 0], X[:, 1], c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
ax.set_xticks(())
ax.set_yticks(())
ax.set_title(titles[0])

# Plot support vectors for SVC
if isinstance(models[0], SVC):
    sv = models[0].support_vectors_
    ax.scatter(sv[:, 0], sv[:, 1], s=100, linewidth=1, facecolors='none', edgecolors='k')

plt.show()

# Make predictions on the test set
y_pred = models[0].predict(X_test_lda)

print('Accuracy :', accuracy_score(y_test, y_pred))
```




```
Accuracy : 0.9666666666666667
```

همانطور که از شکل بالا پیداست مدل بهبود یافته و داده ها بهتر از مدل قبل در نواحی مربوط به کلاسشان قرار گرفته اند.

برای دریافت وزن ها و بایاس های متناظر بردار های پشتیبان از coef. و intercept. استفاده می کنیم:

```
models[0].coef_, models[0].intercept_
```

```
(array([[0.37613279, 0.0520722 ],
        [0.19979248, 0.03650102],
        [0.83932381, 0.10527038]]),
 array([-1.24376165, -0.21011551,  2.892334  ]))
```

برای نمایش اندیس داده های که بردار پشتیبان هستند از _support. استفاده می کنیم. برای دریافت تعداد بردار پشتبان در هر کلاس از _n_support. و برای نمایش دقیق مختصات بردار های پشتیبان از _support_vectors. استفاده می کنیم.

```
models[0].support_, models[0].n_support_, models[0].support_vectors_
```

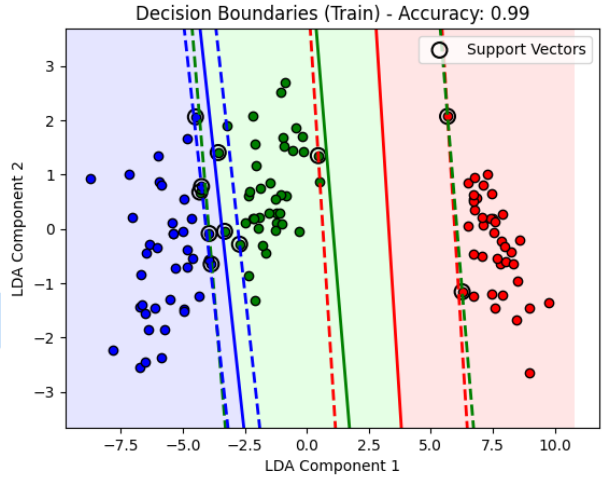
```
(array([ 61, 101,  17,  21,  28,  34,  42,  58,  80, 106, 119,  32,  45,
         48,  53,  54,  78,  85, 116], dtype=int32),
 array([2, 9, 8], dtype=int32),
 array([[ 5.67884286,  2.06948399],
        [ 6.26849989, -1.15912713],
        [-2.33073746,  0.6070995 ],
        [-3.31382563, -0.04688631],
        [-4.31600887,  0.67571794],
        [-3.22557147,  1.88911004],
        [-2.71123322, -0.28423388],
        [-2.35109681, -0.85663356],
        [-3.5754155 ,  1.40348149],
        [-2.46218393,  0.11653676],
        [ 0.46156583,  1.34719894],
        [-3.85721147, -0.65006807],
        [-4.48847511,  2.06000321],
        [-4.83322095,  1.65761409],
        [-3.93758964, -0.08687014],
        [-4.66218718,  0.19711797],
        [-4.33704566, -1.23495039],
        [-4.24735904,  0.77725681],
        [-3.97481429, -0.56027599]]))
```

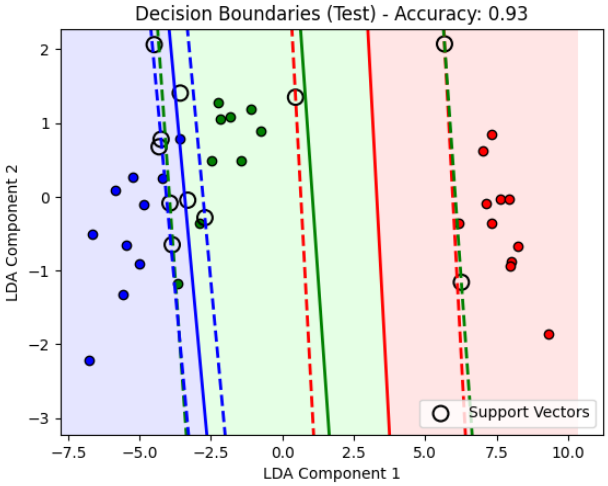
همانطور که پیداست ۱۹ داده بردار پشتیبان هستند که ۲ تای آن بردار پشتیبان کلاس صفر، ۹ تا بردار پشتیبان کلاس ۱ و ۸ تا بردار پشتیبان کلاس ۲ هستند.

به منظور رسم حاشیه ها در مرز تصمیم گیری از قطعه کد زیر استفاده می کنیم که در آن تابع plot_decision_boundaries با در نظر گرفتن مختصات بردار های پشتیبان از وزن ها و بایاس ها در رسم حاشیه تصمیم گیری استفاده شده است و برای کاهش بعد مجددا از LDA استفاده می شود. مرز تصمیم گیری برای داده های آموزش و آزمون رسم می شود:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from matplotlib.colors import ListedColormap
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)
    # Define custom colormap in background
    colors = ['#FFAAAA', '#AAFFAA', '#AAAAFF']
    cmap_background = ListedColormap(colors)
    plt.contourf(xx, yy, Z, alpha=0.3, cmap=cmap_background)
    # Define scatter plot colors
    scatter_colors = ['r', 'g', 'b']
    for i, color in zip(range(3), scatter_colors):
        idx = np.where(y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, edgecolors='k')
        w = model.coef_[i]
        b = model.intercept_[i]
        x0 = np.linspace(x_min, x_max, 200)
        decision_boundary = (-(w[0] / w[1]) * x0) - (b / w[1])
        margin = 1 / w[1]
        gutter_up = decision_boundary + margin
        gutter_down = decision_boundary - margin
        plt.plot(x0, decision_boundary, '-', c=scatter_colors[i], linewidth=2)
        plt.plot(x0, gutter_up, '--', c=scatter_colors[i], linewidth=2)
        plt.plot(x0, gutter_down, '--', c=scatter_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()
# Apply LDA to reduce dimensionality to 2
lda = LinearDiscriminantAnalysis(n_components=2)
X_train_lda = lda.fit_transform(X_train, y_train)
X_test_lda = lda.transform(X_test)
model = SVC(C=1., kernel='linear', probability=True)
model.fit(X_train_lda, y_train)
# Plot decision boundaries for the training set
plot_decision_boundaries(X_train_lda, y_train, model, "Decision Boundaries (Train)")
# Plot decision boundaries for the test set
plot_decision_boundaries(X_test_lda, y_test, model, "Decision Boundaries (Test)")
```

مرز تصمیم گیری به‌صورت زیر رسم شده است:





همانطور که از نتایج بالا پیداست، مدل svm طراحی شده به خوبی توانسته است داده های آزمون را طبقه بندی نماید. در این تصاویر داده های هر کلاس با رنگی متمایز نمایش داده شده اند.

ج

بخش قبلی را با استفاده از هسته های چند جمله ای و با استفاده از کتابخانهscikit-learn از درجه یک تا ۱۰ پیاده سازی کنید و نتایج را با معیارهای مناسب گزارش کرده و مقایسه و تحلیل کنید .در نهایت، با استفاده از کتابخانهimageio جداسازی ویژگی های اصلی را برای درجات ۱ تا ۱۰ در قالب یک GIF به تصویر بکشید و لینک دسترسی مستقیم به فایل GIF را درون گزارش خود قرار دهید.

ابتدا با استفاده از GridSearch مقادیر بهینه پارامتر های c و coef0 را بدست می آوریم:

```
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'kernel': ['poly'],'coef0': [0.1, 1, 2, 5, 1]}
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid.fit(X, y_train)
grid.best_params_
```

```
[CV 4/5] END ........C=10, coef0=5, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END ........C=10, coef0=5, kernel=poly;, score=1.000 total time=   0.0s
[CV 1/5] END ........C=10, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END ........C=10, coef0=1, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END ........C=10, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 4/5] END ........C=10, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END ........C=10, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 1/5] END .....C=100, coef0=0.1, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END .....C=100, coef0=0.1, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END .....C=100, coef0=0.1, kernel=poly;, score=1.000 total time=   0.0s
[CV 4/5] END .....C=100, coef0=0.1, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END .....C=100, coef0=0.1, kernel=poly;, score=1.000 total time=   0.0s
[CV 1/5] END .......C=100, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END .......C=100, coef0=1, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END .......C=100, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 4/5] END .......C=100, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END .......C=100, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 1/5] END .......C=100, coef0=2, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END .......C=100, coef0=2, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END .......C=100, coef0=2, kernel=poly;, score=1.000 total time=   0.0s
[CV 4/5] END .......C=100, coef0=2, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END .......C=100, coef0=2, kernel=poly;, score=1.000 total time=   0.0s
[CV 1/5] END .......C=100, coef0=5, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END .......C=100, coef0=5, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END .......C=100, coef0=5, kernel=poly;, score=1.000 total time=   0.0s
[CV 4/5] END .......C=100, coef0=5, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END .......C=100, coef0=5, kernel=poly;, score=1.000 total time=   0.0s
[CV 1/5] END .......C=100, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END .......C=100, coef0=1, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END .......C=100, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 4/5] END .......C=100, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END .......C=100, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 1/5] END ....C=1000, coef0=0.1, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END ....C=1000, coef0=0.1, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END ....C=1000, coef0=0.1, kernel=poly;, score=1.000 total time=   0.0s
[CV 4/5] END ....C=1000, coef0=0.1, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END ....C=1000, coef0=0.1, kernel=poly;, score=0.958 total time=   0.0s
[CV 1/5] END ......C=1000, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END ......C=1000, coef0=1, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END ......C=1000, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 4/5] END ......C=1000, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END ......C=1000, coef0=1, kernel=poly;, score=0.958 total time=   0.0s
[CV 1/5] END ......C=1000, coef0=2, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END ......C=1000, coef0=2, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END ......C=1000, coef0=2, kernel=poly;, score=1.000 total time=   0.0s
[CV 4/5] END ......C=1000, coef0=2, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END ......C=1000, coef0=2, kernel=poly;, score=0.958 total time=   0.0s
[CV 1/5] END ......C=1000, coef0=5, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END ......C=1000, coef0=5, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END ......C=1000, coef0=5, kernel=poly;, score=0.958 total time=   0.0s
[CV 4/5] END ......C=1000, coef0=5, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END ......C=1000, coef0=5, kernel=poly;, score=0.958 total time=   0.0s
[CV 1/5] END ......C=1000, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 2/5] END ......C=1000, coef0=1, kernel=poly;, score=0.958 total time=   0.0s
[CV 3/5] END ......C=1000, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 4/5] END ......C=1000, coef0=1, kernel=poly;, score=1.000 total time=   0.0s
[CV 5/5] END ......C=1000, coef0=1, kernel=poly;, score=0.958 total time=   0.0s
{'C': 0.1, 'coef0': 2, 'kernel': 'poly'}
```

مقدار c=0.1 و coef0=2 برای کرنل چند جمله ای انتخای شده اند. حال با این پارامترها مدل svm را آموزش می دهیم. حال با ایجاد حلقه for مقدار degree را از یک تا ۱۰ مدل svm را آموزش داده و آن را با داده های آزمون ارزیابی و معیار هایaccuracy وf1-score محاسبه و گزارش می شود:

```
# Polynomial degrees to evaluate
degrees = range(1, 11)
# Train SVM models with polynomial kernels
models = []
for degree in degrees:
    model = SVC(C=0.1, kernel='poly', degree=degree,coef0=2, random_state=64)
    model.fit(X_train_scaled, y_train)
    models.append(model)

# Evaluate and compare models
train_accuracies = []
test_accuracies = []
train_f1s = []
test_f1s = []
for model in models:
    train_pred = model.predict(X_train_scaled)
    test_pred = model.predict(X_test_scaled)

    train_acc = accuracy_score(y_train, train_pred)
    test_acc = accuracy_score(y_test, test_pred)

    train_f1 = f1_score(y_train, train_pred, average='weighted')
    test_f1 = f1_score(y_test, test_pred, average='weighted')

    train_accuracies.append(train_acc)
    test_accuracies.append(test_acc)
    train_f1s.append(train_f1)
    test_f1s.append(test_f1)

# Print results
for degree, train_acc, test_acc, train_f1, test_f1 in zip(degrees, train_accuracies, test_accuracies, train_f1s, test_f1s):
    print(f"Degree {degree}: Train Accuracy = {train_acc:.3f}, Test Accuracy = {test_acc:.3f}, Train F1 = {train_f1:.3f}, Test F1 = {test_f1:.3f}")
```

نتایج به صورت زیر است:

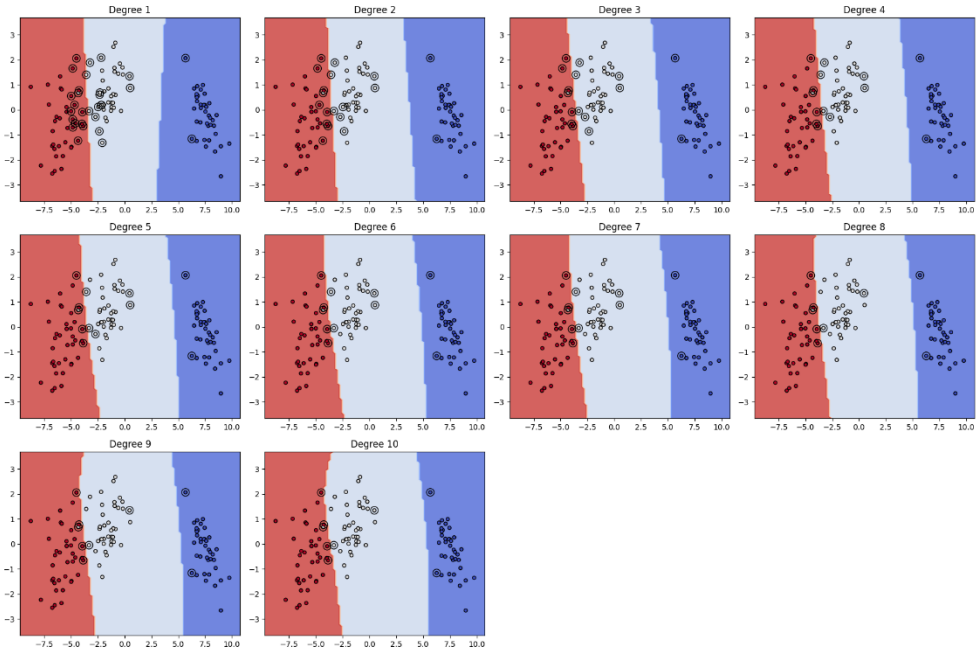
```
Degree 1: Train Accuracy = 0.950, Test Accuracy = 0.933, Train F1 = 0.950, Test F1 = 0.933
Degree 2: Train Accuracy = 0.950, Test Accuracy = 0.967, Train F1 = 0.950, Test F1 = 0.967
Degree 3: Train Accuracy = 0.992, Test Accuracy = 0.967, Train F1 = 0.992, Test F1 = 0.967
Degree 4: Train Accuracy = 0.992, Test Accuracy = 0.967, Train F1 = 0.992, Test F1 = 0.967
Degree 5: Train Accuracy = 0.983, Test Accuracy = 0.933, Train F1 = 0.983, Test F1 = 0.933
Degree 6: Train Accuracy = 0.992, Test Accuracy = 0.933, Train F1 = 0.992, Test F1 = 0.933
Degree 7: Train Accuracy = 0.992, Test Accuracy = 0.933, Train F1 = 0.992, Test F1 = 0.933
Degree 8: Train Accuracy = 1.000, Test Accuracy = 0.933, Train F1 = 1.000, Test F1 = 0.933
Degree 9: Train Accuracy = 1.000, Test Accuracy = 0.933, Train F1 = 1.000, Test F1 = 0.933
Degree 10: Train Accuracy = 1.000, Test Accuracy = 0.933, Train F1 = 1.000, Test F1 = 0.933
```

معیار های ارزیابی روی داده های آموزش و آزمون نشان می دهد انتخاب درجه های ۳ و ۴ برای مدل svm عملکرد بهتری هم روی داده آموزش و هم آزمون خواهند داشت و درجات پایین تر و مدل ساده تر تعمیم پذیری بهتری نیز خواهد داشت. بنابراین چند جمله ای درجه ۳ انتخاب بهینه ای است.

حال با اعمال LDA ابعاد داده را کاهش داده و مرز تصمیم گیری را برای مدل های SVM با کرنل چندجمله ای از درجات ۱تا۱۰ رسم می شود. این کار با ایجاد یک حلقه روی درجات مدنظر برای کرنل چند جمله ای انجام می گیرد.

```python
# Apply LDA for dimensionality reduction to 2 components
lda = LinearDiscriminantAnalysis(n_components=2)
X_train_lda = lda.fit_transform(X_train_scaled, y_train)
X_test_lda = lda.transform(X_test_scaled)
degrees = range(1, 11)
# Train SVM models with polynomial kernels
models = []
for degree in degrees:
    model = SVC(C=1, kernel='poly', degree=degree,coef0=1, random_state=64)
    model.fit(X_train_lda, y_train)
    models.append(model)
plt.figure(figsize=(18, 12))
for i, model in enumerate(models):
    plt.subplot(3, 4, i + 1)
    disp = DecisionBoundaryDisplay.from_estimator(
        model,
        X_train_lda,
        response_method="predict",
        cmap=plt.cm.coolwarm,
        alpha=0.8,
        ax=plt.gca()
    )
    plt.scatter(X_train_lda[:, 0], X_train_lda[:, 1], c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    plt.title(f"Degree {i + 1}")
    # Plot support vectors for SVC
    if hasattr(model, 'support_vectors_'):
        sv = model.support_vectors_
        plt.scatter(sv[:, 0], sv[:, 1], s=100, linewidth=1, facecolors='none', edgecolors='k')
plt.tight_layout()
plt.show()
```

مرز تصمیم گیری در هر مدل SVM با درجات متفاوت کرنل چندجمله ای بصورت زیر رسم می شود:



حال برای ایجاد یک GIF که بتواند مرز تصمیم گیری را با افزایش درجه چند جمله ای نشان دهد، از کتابخانهimageio استفاده کرده و با استفاده از حلقه for روی درجات کرنل چندجمله ای مرز تصمیم گیری برای هر مدل رسم می شود. سپس این مرز های رسم شده را به عنوان تصویر ذخیره می کند و در نهایت، این تصاویر را در قالب یک GIF متحرک ترکیب می کند.

```
images = []
for degree, model in zip(degrees, models):
    plt.figure(figsize=(6, 4))
    disp = DecisionBoundaryDisplay.from_estimator(model, X_train_lda, response_method="predict", cmap=plt.cm.coolwarm, alpha=0.8)
    disp.ax_.scatter(X_train_lda[:, 0], X_train_lda[:, 1], c=y_train, edgecolor="k")
    plt.title(f"Degree {degree}")
    plt.tight_layout()
    # Save each plot as an image
    filename = f'degree_{degree}.png'
    plt.savefig(filename)
    images.append(imageio.imread(filename))
    plt.close()
# Save images as a GIF
gif_filename = 'q1-c-gif.gif'
imageio.mimsave(gif_filename, images, duration=1)

print(f"GIF created: {gif_filename}")
```

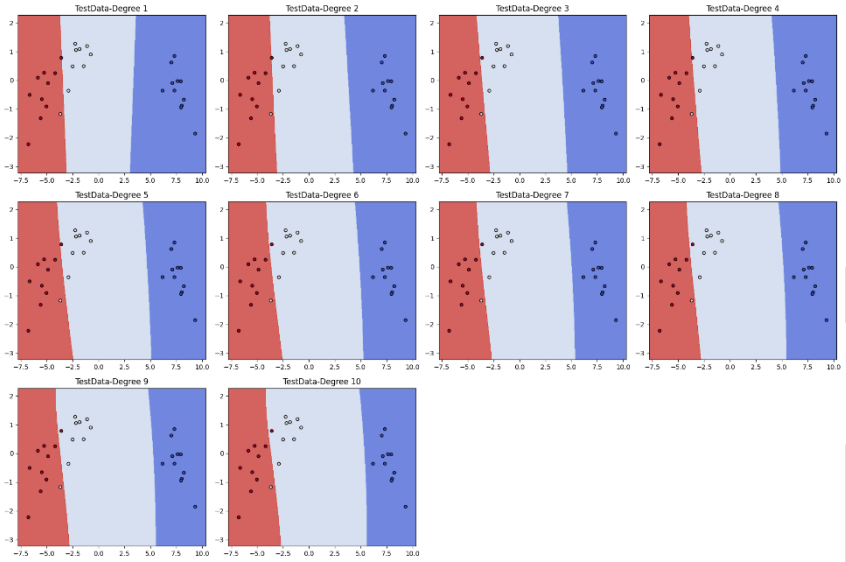
لینک GIF ساخته شده برای داده های آموزش:

https://drive.google.com/file/d/1MsWP9hMQYzrLHgkczqSLJLhpwV4pbFkr/view?usp=sharing

برای داده های آزمون مراحل بالا، برای رسم مرز تصمیم گیری و پس از آن ایجاد یک فایل GIF که نشان دهنده عملکرد مدل با درجات مختلف برای کرنل چندجمله ای است تکرار می شود:

```python
# Plot decision boundaries for each model in test data
plt.figure(figsize=(18, 12))
x_min, x_max = X_test_lda[:, 0].min() - 1, X_test_lda[:, 0].max() + 1
y_min, y_max = X_test_lda[:, 1].min() - 1, X_test_lda[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))

for i, model in enumerate(models):
    plt.subplot(3, 4, i + 1)
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    plt.scatter(X_test_lda[:, 0], X_test_lda[:, 1], c=y_test, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    plt.title(f"TestData-Degree {i + 1}")

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

مرزهای تصمیم گیری مدل برای داده های آزمون به صورت زیر است:



همانطور که پیداست مدل عملکرد نسبتا مطلوبی روی داده های آزمون دارد.

برای ایجاد فایل GIF داده های تست مطابق توضیحات داده شده برای ایجاد فایل GIF داده های آزمون پیش می رویم:

```python
images = []
for degree, model in zip(degrees, models):
    plt.figure(figsize=(6, 4))
    disp = DecisionBoundaryDisplay.from_estimator(model, X_test_lda, response_method="predict", cmap=plt.cm.coolwarm, alpha=0.8)
    disp.ax_.scatter(X_test_lda[:, 0], X_test_lda[:, 1], c=y_test, edgecolor="k")
    plt.title(f"Degree {degree}")
    plt.tight_layout()
    # Save each plot as an image
    filename = f'test_degree_{degree}.png'
    plt.savefig(filename)
    images.append(imageio.imread(filename))
    plt.close()
# Save images as a GIF
gif_filename = 'q1-c-test-gif.gif'
imageio.mimsave(gif_filename, images, duration=1)

print(f"GIF created: {gif_filename}")
```

لینک GIF ساخته شده برای داده های آزمون:

https://drive.google.com/file/d/1QNG6FHe7WknSZXAPvqFcuNeu0raL5s4Y/view?usp=sharing

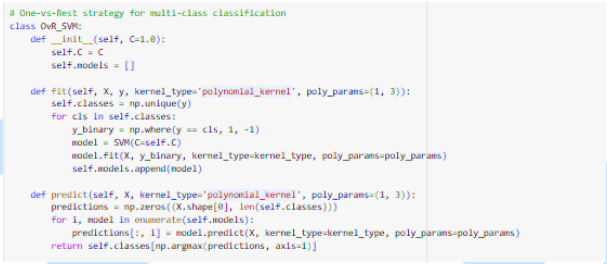
د

الگوریتم SVM را برای مورد قبل به‌صورت From Scratch پیاده سازی کنید .در این بخش لازم است که یک کلاس SVM تعریف کنید .این کلاس می بایســـت حداقل دارای ســـه متد Predict، Fit، Polynomial_kernelباشد. متد Polynomial_kernel می بایست با دریافت درجه های ۱ تا ۱۰ ، هسته های چندجمله ای را محاســبه کند .دقت الگوریتم را با افزایش درجه گزارش کنید و نتایج حاصــل را با بخش قبلی مقایسه کنید .در این قسمت نیز جداسازی ویژگی های اصلی را برای درجات ۱ تا ۱۰ در قالب GIFیک به تصویر بکشید پیوند دسترسی مستقیم آن را در گزارش خود قرار دهید .

در این قسمت ابتدا کلاس svm تعریف شده است که دارای سه متد fit,predict, polynomial_kernel است. متد polynomial_kernel ورودی x,y را می گیرد و r و degree عملکرد کرنل چند جمله ای را کنترل می کنند. متد fit مدل svm را با داده x و برچسبy آموزش میدهد. در این متد ماتریسGram با استفاده از تابع کرنل مشخص شده محاسبه می شود. ضرایب لاگرانژ با حل Quadratic Programming با استفاده از cvxopt بدست می آید. سپس بردارهای پشتیبان با ضرایب لاگرانژ غیر صفر شناسایی می شود. متد predict خروجی را برای داده آزمون با استفاده از ضرایب لاگرانژ (self.a و بردار های پشتیبان پیش بینی می کند.

```
class SVM:
    def __init__(self, C=1.0):
        self.C = C
        self.w = None
        self.b = 0
        self.a = None
        self.sv_x = None
        self.sv_y = None

    @staticmethod
    def polynomial_kernel(X, Y, r=2, degree=3):
        return (r + np.dot(X, Y.T)) ** degree

    def fit(self, X, y, kernel_type='polynomial_kernel', poly_params=(1, 3)):
        n_samples, n_features = X.shape

        # Compute the Gram matrix
        if kernel_type == 'polynomial_kernel':
            K = self.polynomial_kernel(X, X, poly_params[0], poly_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), 'd')
        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) * self.C)))

        # Solve QP problem
        cvxopt.solvers.options['show_progress'] = False
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)

        # Lagrange multipliers
        self.a = np.ravel(solution['x'])

        # Support vectors have non-zero Lagrange multipliers
        sv = self.a > 1e-5
        self.a = self.a[sv]
        self.sv_x = X[sv]
        self.sv_y = y[sv]

        # Bias
        if len(self.a) > 0:
            K_sv = K[sv][:, sv]
            self.b = np.mean(self.sv_y - np.dot((self.a * self.sv_y), K_sv))
        else:
            self.b = 0

        self.w = None  # Not used for non-linear kernels

    def predict(self, X_t, kernel_type='polynomial_kernel', poly_params=(1, 3)):
        if kernel_type == 'polynomial_kernel':
            K = self.polynomial_kernel(X_t, self.sv_x, poly_params[0], poly_params[1])
        else:
            raise ValueError("Invalid kernel type")

        y_predict = np.dot(K, self.a * self.sv_y) + self.b
        return np.sign(y_predict)
```

کلاس دیگری که تعریف شده است، ovr_svm است که استراتژی One-Vs-Rest را برای طبقه بندی چند کلاسه با استفاده از SVM پیاده سازی می کند. این کلاس دارای متدهای fit و predict است. متد fit برای هر کلاس یک مدل svm تعریف می کند و آنها را ذخیره می کند. متد predict لیبل داده آزمون را با ترکیب پیش بینی های مدل های svm، پیش بینی می کند.

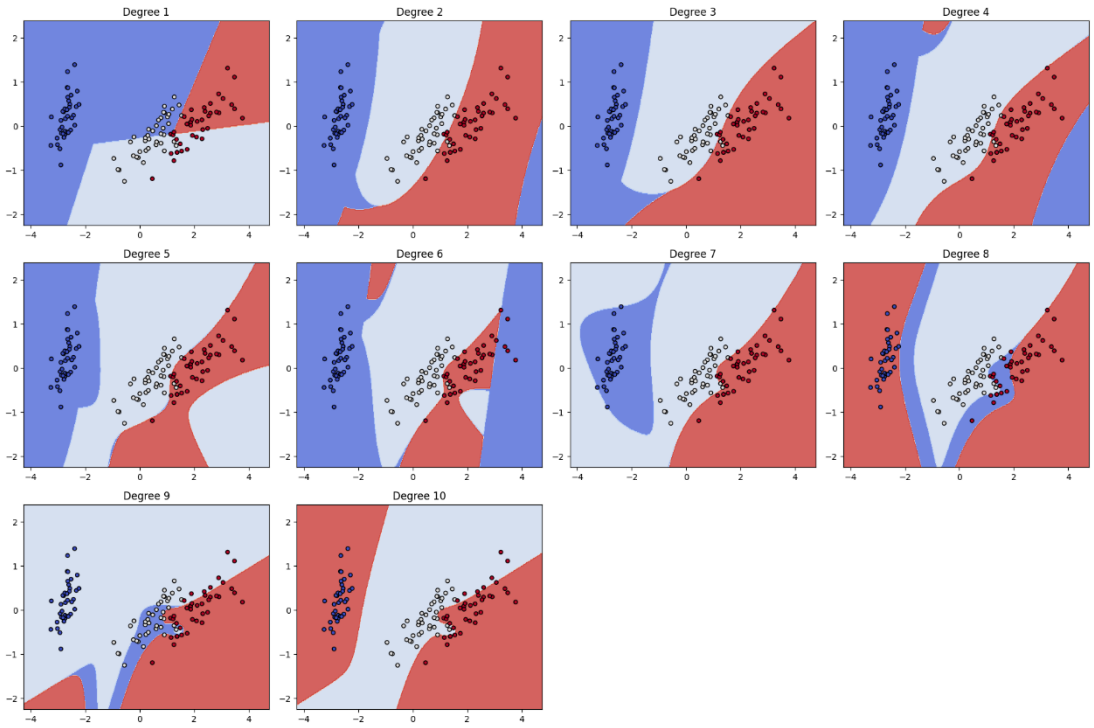


```
# One-vs-Rest strategy for multi-class classification
class OvR_SVM:
    def __init__(self, C=1.0):
        self.C = C
        self.models = []

    def fit(self, X, y, kernel_type='polynomial_kernel', poly_params=(1, 3)):
        self.classes = np.unique(y)
        for cls in self.classes:
            y_binary = np.where(y == cls, 1, -1)
            model = SVM(C=self.C)
            model.fit(X, y_binary, kernel_type=kernel_type, poly_params=poly_params)
            self.models.append(model)

    def predict(self, X, kernel_type='polynomial_kernel', poly_params=(1, 3)):
        predictions = np.zeros((X.shape[0], len(self.classes)))
        for i, model in enumerate(self.models):
            predictions[:, i] = model.predict(X, kernel_type=kernel_type, poly_params=poly_params)
        return self.classes[np.argmax(predictions, axis=1)]
```

در ادامه برای آموزش دادن مدل، ابتدا با استفاده از PCA ابعاد را کاهش می دهیم(LDA در اینجا با خطای Rank مواجه می‌شود.). مدل های svm با کلاس ovr_svm برای هر درجه چند جمله ای با استفاده از داده های آموزشی کاهش بعد یافته آموزش داده می شود. سپس مدل ها با داده های آزمون ارزیابی می شوند. ترسیم مرزهای تصمیم برای هر مدل SVM آموزش دیده با درجات چند جمله ای متفاوت، با استفاده از یک حلقه for صورت می گیرد.

```
# Apply PCA for dimensionality reduction to 2 components
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Polynomial degrees to evaluate
degrees = range(1, 11)

models = []
accuracies = []
f1_scores = []

for degree in degrees:
    model = OvR_SVM(C=1)
    model.fit(X_train_pca, y_train, kernel_type='polynomial_kernel', poly_params=(1, degree))
    y_pred = model.predict(X_test_pca, kernel_type='polynomial_kernel', poly_params=(1, degree))
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='macro')
    accuracies.append(accuracy)
    f1_scores.append(f1)
    models.append(model)

    print(f'Degree {degree}: Accuracy = {accuracy:.4f}, F1 Score = {f1:.4f}')

plt.figure(figsize=(18, 12))
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))

for i, model in enumerate(models):
    plt.subplot(3, 4, i + 1)
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()], kernel_type='polynomial_kernel', poly_params=(1, degrees[i]))
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    plt.title(f"Degree {i + 1}")
# Plot support vectors for SVC
    if hasattr(model, 'support_vectors_'):
        sv = model.support_vectors_
        plt.scatter(sv[:, 0], sv[:, 1], s=100, linewidth=1, facecolors='none', edgecolors='k')


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

نتایج برای داده های آزمون به‌صورت زیر است:



برای ایجاد فایل GIF برای داده های آزمون مانند قسمت قبل از قطعه کد زیر استفاده می کنیم:

```
images = []
x_min, x_max = X_train_pca[:, 0].min() - 1, X_train_pca[:, 0].max() + 1
y_min, y_max = X_train_pca[:, 1].min() - 1, X_train_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
for i, model in enumerate(models):
    plt.figure(figsize=(6, 4))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()], kernel_type='polynomial_kernel', poly_params=(1, degrees[i]))
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    plt.title(f"Degree {i + 1}")
    plt.tight_layout()
    # Save each plot as an image
    filename = f'degree_{i + 1}.png'
    plt.savefig(filename)
    images.append(imageio.imread(filename))
    plt.close()
# Save images as a GIF
gif_filename = 'q1_d.gif'
imageio.mimsave(gif_filename, images, duration=1)
```

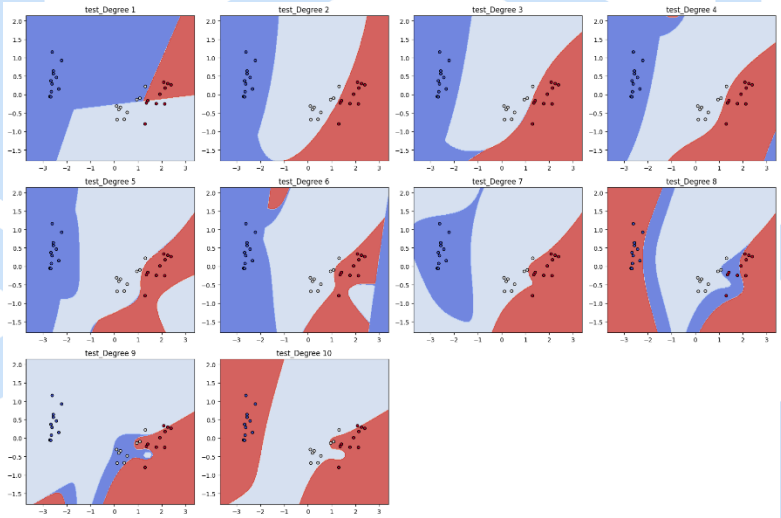
لینک GIF ساخته شده برای داده های آموزش:

https://drive.google.com/file/d/1muge39pWBadv8gnVP8yEQYHSFwyhVWMw/view?usp=sharing

مرز تصمیم برای داده های آزمون با قطعه کد زیر ترسیم می شود:

```
plt.figure(figsize=(18, 12))
x_min, x_max = X_test_pca[:, 0].min() - 1, X_test_pca[:, 0].max() + 1
y_min, y_max = X_test_pca[:, 1].min() - 1, X_test_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))

for i, model in enumerate(models):
    plt.subplot(3, 4, i + 1)
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()], kernel_type='polynomial_kernel', poly_params=(1, degrees[i]))
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_test, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    plt.title(f"test_Degree {i + 1}")
    # Plot support vectors for SVC
    if hasattr(model, 'support_vectors_'):
        sv = model.support_vectors_
        plt.scatter(sv[:, 0], sv[:, 1], s=100, linewidth=1, facecolors='none', edgecolors='k')

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

مرز تصمیم گیری برای داده آزمون به صورت زیر رسم شده است:



نتایج با معیارهای ارزیابی Accuracy و f1-score برای داده های تست به‌صورت زیر است:

```
Degree 1: Accuracy = 0.7333, F1 Score = 0.7156
Degree 2: Accuracy = 1.0000, F1 Score = 1.0000
Degree 3: Accuracy = 1.0000, F1 Score = 1.0000
Degree 4: Accuracy = 1.0000, F1 Score = 1.0000
Degree 5: Accuracy = 1.0000, F1 Score = 1.0000
Degree 6: Accuracy = 1.0000, F1 Score = 1.0000
Degree 7: Accuracy = 1.0000, F1 Score = 1.0000
Degree 8: Accuracy = 0.4667, F1 Score = 0.4619
Degree 9: Accuracy = 0.3000, F1 Score = 0.2957
Degree 10: Accuracy = 0.5333, F1 Score = 0.4620
```

نتایج داده های آزمون حاکی از عملکرد مناسب مدل تا درجه ۷ است و پس از آن عملکرد مدل به شدت ضعیف گزارش شده است. همچنین مدل از درجه ۵ به بعد بسیار حالت پیچیده ای می گیرد و تعمیم پذیر آن کاهش می یابد.

برای ایجاد فایل GIF داده های تست از قطعه کد مشابه برای قسمت آزمون استفاده می کنیم:

```python
images = []
x_min, x_max = X_test_pca[:, 0].min() - 1, X_test_pca[:, 0].max() + 1
y_min, y_max = X_test_pca[:, 1].min() - 1, X_test_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
for i, model in enumerate(models):
    plt.figure(figsize=(6, 4))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()], kernel_type='polynomial_kernel', poly_params=(1, degrees[i]))
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
    plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_test, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
    plt.title(f"Degree {i + 1}")
    plt.tight_layout()
    # Save each plot as an image
    filename = f'test_degree_{i + 1}.png'
    plt.savefig(filename)
    images.append(imageio.imread(filename))
    plt.close()
# Save images as a GIF
gif_filename = 'q1_d_test.gif'
imageio.mimsave(gif_filename, images, duration=1)
print(f"GIF created: {gif_filename}")
```

لینک GIF ساخته شده برای داده های آزمون:

https://drive.google.com/file/d/1Q67NCTqVOkAHCvi6Vy0xX4kBdkOkm4J0/view?usp=sharing

برای مقایسه نتایج بخش قبل و این قسمت می بینیم نتایج قسمت قبل بگونه ای بود که بردارهای پشتیبان بگونه ای انتخاب و رسم شدند که تعمیم پذیری مدل بالا بود. اما در این قسمت با اینکه دیتاست را به خوبی و با دقت بالایی طبقه بندی کرده است، مدل مخصوصا با افزایش درجه چندجمله ای، پیچیدگی دارد و بنظر می آید تعمیم پذیری خوبی نداشته باشد. تا جایی که می بینیم در درجات بالاتر(۸و ۹و۱۰) دقت مدل بسیار کاهش پیدا کرده است و به ۳۰٪ نیز رسیده است. حال آنکه مدل ها با درجات مذکور در مدل قسمت قبل دقتی حدود ۹۳٪ دارد.

سوال سوم

آ

بزرگ ترین چالش ها در توسعهٔ مدل های تشخیص تقلب چیست؟ این مقاله برای حل این چالش ها از چه روش هایی استفاده کرده است؟

مطابق مقاله مرجع سوال، بزرگ‌ترین چالش‌ها در توسعه مدل‌های تشخیص تقلب شامل برخورد با مجموعه داده‌های نامتعادل یا کم دسترس، تشخیص تراکنش‌های تقلبی از معاملات قانونی به دلیل شباهت تراکنش تقلبی با تراکنش مشروع، انتخاب ویژگی بهینه برای مدل ها و معیار مناسب برای ارزیابی عملکرد بر روی داده های تقلب کارت اعتباری و اطمینان از دقت بالای طبقه‌بندی برای کلاس های اقلیت است.

برای حل این چالش ها، این مقاله از ترکیبی از نمونه برداری بیش از حد(oversampling) برای افزایش مقدار نمونه در کلاس های اقلیت و یک شبکه عصبی انکودر خودکار حذف نویز استفاده کرده است. هدف الگوریتم پیشنهادی افزایش دقت طبقه‌بندی کلاس‌های اقلیت در مجموعه داده‌های نامتعادل، با نمونه‌برداری بیش از حد برای افزایش نمونه‌های کلاس اقلیت و استفاده از انکودر خودکار حذف نویز برای حذف نویز است. علاوه بر این، مقاله با اشاره به کارهای گذشته، تکنیک‌های مختلف داده‌کاوی مانند شبکه‌های عصبی و درخت‌های تصمیم‌گیری را که در تشخیص تقلب کارت اعتباری برای مقابله با این چالش‌ها به کار گرفته شده‌اند، مورد بحث قرار می‌دهد. هدف این مقاله اجرای روش‌های محاسباتی پیشرفته مانند انکودر خودکار حذف نویز و نمونه‌برداری بیش از حد برای بهبود عملکرد مدل‌های تشخیص تقلب در صنعت مالی است.

هدف اتوانکودر یادگیری نمایشی برای بازسازی ویژگی‌ها برای مجموعه‌ای از داده‌ها به منظور کاهش ابعاد است.

ب

در مورد معماری شبکهٔ ارائه شده در مقاله به صورت مختصر توضیح دهید.

در این مقاله ابتدا از oversampling برای تبدیل مجموعه داده نامتعادل به مجموعه داده متعادل استفاده می شود. سپس از اتوانکودر حذف نویز برای دریافت مجموعه داده بدون نویز استفاده می شود. مجموعه داده در نهایت با استفاده از مدل شبکه عصبی عمیق طبقه بندی می شود.

بنابراین شبکه سه بخش اصلی دارد:

۱- OverSampling

OverSampling راهکاری است برای ایجاد یک دیتاست متعادل. برای ایجاد یک نقطه داده مصنوعی، ابتدا باید خوشه _kنزدیکترین همسایه (KNNرا در فضای ویژگی پیدا کنیم، سپس به طور تصادفی یک نقطه را در این خوشه پیدا کنیم، در نهایت از میانگین وزنی برای ساختن داده جدید استفاده می کنیم.

۲- Denoising Autoencoder

اتوانکودرها شبکه های عصبی مصنوعی برای یادگیری بدون نظارت هستند. تفاوت مهم اتوانکودر و MLP این است که اتوانکودر در لایه خروجی به تعداد نورون لایه ورودی، نورون دارد. چراکه به جای پیش بینی مقدار هدف از ورودی های داده شده، میخواهد ورودی های خود را بازسازی کند. آموزش اتوانکودر بر مبنای بهینه سازی خطای بازسازی با استفاده از نمونه های داده شده است. تابع هزینه استفاده شده به‌صورت زیر است:

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

در اتوانکودر با حذف نویز، تابع هزینه سعی می کند تفاوت بین خروجی و داده های اصلی را به حداقل برساند تا اتوانکودر توانایی حذف اثر نویز و استخراج ویژگی ها از داده های نویزی را داشته باشد.

۳- Autoencoder Neural Network

این شبکه شامل یک لایهٔ ورودی و چند لایهٔ پنهان، و یک لایهٔ خروجی است. این قسمت از شبکه عمل طبقه بندی را انجام می دهد. در واقع داده ها پس از حذف نویز وارد شبکه عصبی می شوند و طبقه بندی داده صورت می گیرد. شبکه های عصبی عمیق با تابع هزینه کراس انتروپی و فعالساز SoftMAX دقت بالایی دارند. تابع SoftMax برای تبدیل توزیع احتمال بکار می رود و مقداری بین صفر و یک دارد. انتروپی معیاری برای محتویات اطلاعات است و می تواند نشان دهنده میزان غیرقابل پیش بینی بودن یک رویداد باشد. بنابراین، هرچه احتمال بیشتر باشد، غیرقابل پیش بینی بودن آن کوچکتر است، به این معنی که محتوای اطلاعات نیز بسیار کم است. کراس انتروپی با ترکیب SoftMax می تواند درمسائل طبقه بندی چند کلاسه بکار رود.

در نهایت ارزیابی شبکه صورت می گیرد. معیار ارزیابی مدل در دیتاست های نامتعادل معمولا ماتریس در هم ریختگی است. از معیار دقت به تنهایی نمی توان در این دیتاست ها استفاده کرد. زیرا اگر مدل تمام نمونه ها را با برچسب کلاس بیشتر لیبل گذاری کند، دقت مدل بالا خواهد بود در حالی که تشخیص

ناهنجاری در این مدل قابل انجام نیست. معیار recall نسبت تعداد ناهنجاری‌های به‌درستی شناسایی‌شده و تعداد کل ناهنجاری‌ها است و ارزیابی می‌کند که چه مقدار از ناهنجاری‌ها را می‌توان در این مدل طبقه‌بندی شناسایی کرد.

ج

مدل ارائه شده را پیاده سازی کرده و با استفاده از این دیتاست آموزش دهید. برای جلوگیری از بیش برازش، آموزش مدل را طوری تنظیم کنید که در انتهای آموزش، بهترین وزن های مدل بر اساس خطای قسمت اعتبارسنجی بازگردانده شود.

دیتاستی که مقاله روی آن مدل را آموزش داده است، 28315 نمونه دارد که 0.5% آن را داده تقلب تشکیل داده است. اما دیتاست معرفی شده در صورت سوال 284,807 داده دارد که0.172% آن داده تقلب است. بنابراین طبیعی است نتایج این گزارش با نتایج مقاله مطابقت نداشته باشد. مراحل را گام به گام مطابق مقاله پیش می‌بریم:

۱-ج پیش پردازش داده

دیتاست را برای وجود داده پوچ بررسی می کنیم. پیداست دیتاست داده پوچی ندارد. همچنین ابعاد دیتاست ۲۸۴۸۰۷در ۳۱ است:

```
# Load the dataset
data = pd.read_csv('/content/creditcard.csv')
data.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

```
data.dropna(inplace=True)
data.isnull().sum()
data.shape
```

```
(284807, 31)
```

مطابق قسمت 4.1 مقاله، ابتدا ستون Time را از دیتاست حذف و سپس ستون AMOUNT را نرمال می کنیم.سایر ویژگی ها بدون نیاز به نرمال سازی با pca بدست آورده می‌شوند. سپس دیتاست را پس از مخلوط کردن، با نسبت ۸۰٪ و ۲۰٪ به آموزش و آزمون تقسیم می کنیم. همچنین ۱۰٪ از داده ها را برای اعتبار سنجی نگه می داریم. (نرمال سازی دیتاست با استفاده از standardscaler انجام شده است. بوسیله pca دیتا ست(بجز ستون مربوط به برچسب و ستون AMOUNT) را به ویژگی های ا سا سی آن کاهش بعد می دهیم).

```
# Drop the "Time" column
data.drop(columns=['Time'], inplace=True)
# Normalize the "Amount" column
scaler = StandardScaler()
data['Amount'] = scaler.fit_transform(data[['Amount']])
# Apply PCA to the features (excluding "Amount" and "Class")
features = data.drop(columns=['Class','Amount'])
pca = PCA()
pca_features = pca.fit_transform(features)
# Convert PCA features to a DataFrame
pca_features_df = pd.DataFrame(pca_features, columns=[f'V_{i+1}' for i in range(pca_features.shape[1])])
# Combine the PCA features with the "Class" column
data_pca = pd.concat([pca_features_df, data[['Amount','Class']]], axis=1)
print(data_pca)
array = data_pca.values  #  numpy array
np.random.seed(64)
np.random.shuffle(array)  # Shuffle the array
data = pd.DataFrame(array, columns=data_pca.columns)
print(data)
# Split the data into training and test sets
train_data, test_data = train_test_split(data, test_size=0.2, random_state=64)
train_data, valid_data = train_test_split(train_data, test_size=0.1, random_state=64)
# Separate features and labels
X_train = train_data.drop(columns=['Class'])
y_train = train_data['Class']
X_test = test_data.drop(columns=['Class'])
y_test = test_data['Class']
X_valid = test_data.drop(columns=['Class'])
y_valid = test_data['Class']
```

مطابق مقاله پایه، ویژگی های V1, V2, ... V28 مولفه های اساسی اند و ستون های class AMOUNT, با PCA تبدیل نشده اند:

```
           V_1       V_2       V_3       V_4       V_5       V_6       V_7  \
0    -0.615475  1.065290 -1.377195 -0.187755  0.636940  0.473939 -0.932501
1    -0.668679 -2.862764 -0.728884  1.114219 -1.010398  1.306962 -0.965169
2     0.283389  1.243382 -0.234182 -0.546643 -0.119500 -0.075661 -0.287140
3    -0.430387  1.220483  1.055408 -0.567334  1.040400  1.224084  0.826189
4    -0.379266  1.082665 -1.130233 -0.988784 -1.124743 -3.218155  3.330159
...        ...       ...       ...       ...       ...       ...       ...
31774 -1.085241  1.274855  1.550730 -1.451406  0.399366 -0.732285 -2.306148
31775 -0.445527  1.266761  0.685833 -0.580367 -1.525743  0.174606 -2.365772
31776 -0.622016  1.071438 -1.395371 -0.193938  0.471873  0.273794 -0.680829
31777 -0.653842  1.405357  0.948030  0.757925 -0.911711 -0.511970 -0.463505
31778 -0.392485 -2.797816 -0.871669  0.032144  1.348589  0.710160  0.139414

           V_8       V_9      V_10  ...      V_21      V_22      V_23  \
0    -0.801773  0.469624  0.099867  ...  0.169444  0.748194  0.338439
1     0.309183  0.140898 -1.224857  ...  0.191915  0.434552 -0.664132
2    -0.377204 -0.565668 -0.266059  ...  0.046328 -0.440466 -0.063954
3     2.884680 -1.433055 -0.459950  ...  0.653298  0.029696 -0.197558
4    -0.317342  0.499903  0.106938  ...  0.111989  0.957139 -0.856682
...        ...       ...       ...  ...       ...       ...       ...
31774  1.078704 -1.152395  0.817128  ...  0.295553  0.477277 -0.951410
31775  0.477048 -0.912284 -0.399102  ...  0.027387  0.423453 -1.134674
31776 -0.821956  0.409577  0.075712  ...  0.157157  0.433302  0.352494
31777  0.362292 -0.626464 -0.348652  ... -0.221771 -0.756967  1.007190
31778 -0.335846  0.208875 -0.334481  ...  0.244345  0.329807 -0.607518

           V_24      V_25      V_26      V_27      V_28    Amount  Class
0    -0.268764 -0.311886 -0.094684 -0.021219 -0.010999 -0.354625    0.0
1     0.088473 -0.666755 -0.061857 -0.107964 -0.012930 -0.310395    0.0
2    -0.021091  0.700609 -0.151966 -0.214281 -0.067325 -0.356776    0.0
3     0.088635 -0.749541 -0.368803 -0.380654 -0.023595  0.967439    0.0
4     0.120264  0.577252  0.237846 -0.161368  0.198852 -0.316490    0.0
...        ...       ...       ...       ...       ...       ...    ...
31774  0.660164 -0.130892 -0.576111  0.719123 -0.656583 -0.296279    0.0
31775  0.566923  0.013376  0.262421 -0.284909  0.007872 -0.359017    0.0
31776 -0.277223 -0.309010 -0.092135 -0.024269 -0.009052 -0.354625    0.0
31777 -0.252317  0.435090  0.166592  0.323539 -0.002736 -0.157808    0.0
31778  0.016258 -0.017444  0.099151  0.164302 -0.014183 -0.356821    0.0

[31779 rows x 30 columns]
```

برای عملیات over sampling از داده های آموزش استفاده می کنیم. در اینجا تعداد داده تقلبی مجموعه داده آموزش ۳۷۴ است:

```
print(X_train.shape)
unique, counts = np.unique(y_train, return_counts=True)
print("Class counts:", dict(zip(unique, counts)))
```

```
(205060, 29)
Class counts: {0.0: 204686, 1.0: 374}
```

۲-ج Oversampling

برای oversampling از متد SMOTE و از کتابخانه imblearn.over_sampling استفاده می کنیم.با این کار تعداد داده را در کلاس کمتر افزایش داده تا دیتاست بالانس شود. نتایج نشان می دهند پس از oversampling تعداد داده در کلاس تقلب و کلاس نرمال برابر خواهند شد:

```
from imblearn.over_sampling import SMOTE
# Perform oversampling on the training dataset using SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
unique, counts = np.unique(y_train_resampled, return_counts=True)
print("Class counts:", dict(zip(unique, counts)))
```

```
Class counts: {0.0: 204686, 1.0: 204686}
```

۳-ج اتوانکودر حذف نویز

در این قسمت از مقاله یک اتوانکودر ۷ لایه برای فرآیند حذف نویز طراحی شده است. ابتدا نویز گاوسی به مجموعه داده آموزشی اضافه می‌شود، سپس در مرحله آموزش، مجموعه داده آموزشی به اتوانکودر حذف نویزداده می‌شود. پس از آموزش مدل، توانایی اتوانکودر در حذف نویز داده‌های آزمون بررسی می شود.

برای ایجاد نویز از دستور np.random.normal که درای توزیع نرمال گوسی با میانگین صفر و واریانس ۱است، استفاده می‌کنیم:

```
# Add Gaussian noise to the training dataset
np.random.seed(64)
noise_factor = 0.5
X_train_noisy = X_train_resampled + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train_resampled.shape)
```

حال مطابق آنچه در قسمت 5.1 مقاله گفته شده است، برای ایجاد اتوانکودر از کتابخانه TensorFlow

استفاده می‌کنیم. ابعاد لایه ها را مطابق جدول ۲ مقاله تنظیم می‌کنیم. از تابع فعال‌ساز relu در لایه های میانی و در لایه خروجی از sigmoid استفاده می‌کنیم. ابعاد لایه ورودی و خروجی این شبکه ۲۹ (ابعاد دیتای ورودی) است. و پس از لایه ورودی، ۳ لایه بعدی عمل انکود را انجام می دهند و از بعد ۲۹ به بعد ۲۲ و پس از آن ۱۵ و سپس به بعد ۱۰ می رویم در سه لایه بعدی عمل دیکود انجام می شود و از بعد ۱۰ به ترتیب به ابعاد ۱۵، ۲۲ و در لایه خروجی به بعد ۲۹ باز می گردیم. در فرآیند آموزش از بهینه ساز آدام با نرخ یادگیری پیشفرض0.001 و تابع هزینه میانگین مربعات خطا استفاده می شود:

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

# Define the 7-layer autoencoder architecture
input_dim = X_train.shape[1] #input_dim=29
input_layer = Input(shape=(input_dim,))
encoder_1 = Dense(22, activation='relu')(input_layer)
encoder_2 = Dense(15, activation='relu')(encoder_1)
latent_layer = Dense(10, activation='relu')(encoder_2)
decoder_1 = Dense(15, activation='relu')(latent_layer)
decoder_2 = Dense(22, activation='relu')(decoder_1)
output_layer = Dense(input_dim, activation='sigmoid')(decoder_2)

autoencoder = Model(inputs=input_layer, outputs=output_layer)

# Compile the model
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
```

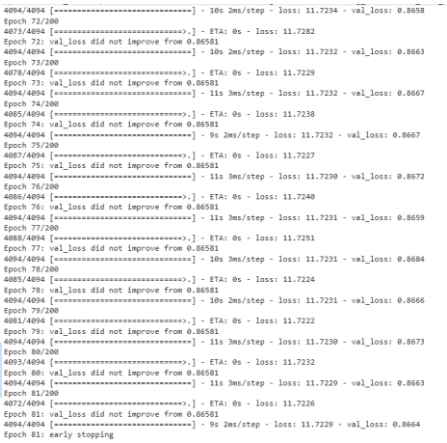
حال مدل را با داداه های آموزش، آموزش می‌دهیم!

در این فرآیند ازModelCheckpoint وEarlyStopping برای ذخیره بهترین وزن مدل بر اســاس خطای قسمت اعتبارسنجیاستفاده می کنیم تا از بیش برازش جلوگیری کنیم. برای ذخیره بهترین وزن از فرمت HDF5 که فرمت پیش فرض Keras برای ذخیره معماری و وزن مدل است، استفاده می کنیم.

مقادیر Epoch و batch_size در مقاله تعیین نشده است. در اینجا تعداد epoch را ۲۰۰ و batch_size را ۱۰۰ در نظر می گیریم.

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import matplotlib.pyplot as plt
np.random.seed(64)
tf.random.set_seed(64)
# Define the checkpoint callback
checkpoint = ModelCheckpoint('denoising_autoencoder_best_model.h5', monitor='val_loss', mode='min', save_best_only=True, verbose=1)
# Early stopping to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
# Train the model
history = autoencoder.fit(X_train_noisy, X_train_resampled,
                          epochs=200,
                          batch_size=100,
                          shuffle=True,
                          validation_data=(X_valid, X_valid),
                          callbacks=[checkpoint, early_stopping])
```

بخشی از نتایج آموزش مدل به‌صورت زیر است:



همانطور که پیداســـت، early-stopping فرآیند آموزش را در ایپاک ۸۱ متوقف کرده اســـت و خطای اعتبارسنجی تا 0.8664 کاهش یافته است.

حال مدل را روی داده های آزمون تست می‌کنیم:

```
# Load the best model
autoencoder.load_weights('denoising_autoencoder_best_model.h5')
# Denoise the test data
X_test_denoised = autoencoder.predict(X_test)
# Evaluate the model on test data
test_loss = autoencoder.evaluate(X_test, X_test)
print(f'Test loss: {test_loss}')
```

```
1781/1781 [==========] - 2s 1ms/step
1781/1781 [==========] - 2s 1ms/step - loss: 0.8658
Test loss: 0.8658053278923035
```

خطای داده تست نیز تا 0.8658 کاهش یافته است.

۴-ج مدل طبقه بند

تا اینجا مجموعه داده آموزشی حذف نویز شده را از اتوانکودر حذف نویز دریافت کردیم. حال یک مدل طبقه بند با ۶ لایه طراحی می کنیم که ابعاد هر لایه مطابق جدول ۳ مقاله تنظیم شده است. لایه ورودی با ۲۹ بعد و لایه های بعد به ترتیب ابعاد ۲۲و ۱۵و۱۰و۵ و لایه خروجی ۲(به تعداد کلاس ها) دارند. مدل طبقه‌بند شبکه عصبی عمیق را با مجموعه داده آموزشی حذف نویز شده آموزش می دهیم. در لایه آخر، از تابع فعالساز SoftMax و در سایر لایه ها از فعالساز relu استفاده می شود.تابع هزینه کراس آنتروپی و بهینه ساز آدام با نرخ یادگیری پیشفرض0.001 مورد استفاده قرار گرفته اند.

طراحی این مدل مطابق قسمت5.1 مقاله با استفاده از کتابخانه TensorFlow صورت گرفته است.

ابتدا داده های حذف نویز شده را دریافت می کنیم:

```
# Get the denoised training data
X_train_denoised = autoencoder.predict(X_train_noisy)
X_test_denoised = autoencoder.predict(X_test)
X_valid_denoised = autoencoder.predict(X_valid)
```

حال طبقه بند شبکه عصبی عمیق را با توضیحات فوق و با معیار Accuracy طراحی می کنیم:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
# Build the classifier
classifier = Sequential([
    Dense(22, input_dim=29, activation='relu'),
    Dense(15, activation='relu'),
    Dense(10, activation='relu'),
    Dense(5, activation='relu'),
    Dense(2, activation='softmax')
])
# Compile the classifier
classifier.compile(optimizer=Adam(), loss='BinaryCrossentropy', metrics=['accuracy'])
```

حال مدل را با در نظر گرفتن ۲۰۰ ایپاک و batch_size برابر ۱۰۰ آموزش می‌دهیم. در این فرآیند از ModelCheckpointوEarlyStopping برای ذخیره بهترین وزن مدل بر اساس خطای قسمت اعتبارسنجی و جلوگیری ازبیش برازش استفاده می‌شود.

```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
# Define the checkpoint callback
checkpoint1 = ModelCheckpoint('classifier_best_model.h5', monitor='val_loss', mode='min', save_best_only=True, verbose=1)
# Early stopping to avoid overfitting
early_stopping1 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
# Train the classifier
history_classifier = classifier.fit(X_train_denoised, y_train_resampled,
                                    epochs=200,
                                    batch_size=100,
                                    shuffle=True,
                                    validation_data=(X_valid_denoised, y_valid),
                                    callbacks=[checkpoint1, early_stopping1])
```

نتایج به‌صورت زیر است:

```
4094/4094 [==============================] - 8s 2ms/step - loss: 0.0942 - accuracy: 0.9631 - val_loss: 0.0464 - val_accuracy: 0.9882
Epoch 11/200
4070/4094 [============================>.] - ETA: 0s - loss: 0.0936 - accuracy: 0.9634
Epoch 11: val_loss did not improve from 0.04645
4094/4094 [==============================] - 10s 2ms/step - loss: 0.0935 - accuracy: 0.9634 - val_loss: 0.0690 - val_accuracy: 0.9775
Epoch 12/200
4086/4094 [============================>.] - ETA: 0s - loss: 0.0929 - accuracy: 0.9636
Epoch 12: val_loss did not improve from 0.04645
4094/4094 [==============================] - 10s 2ms/step - loss: 0.0929 - accuracy: 0.9636 - val_loss: 0.0679 - val_accuracy: 0.9796
Epoch 13/200
4088/4094 [============================>.] - ETA: 0s - loss: 0.0923 - accuracy: 0.9640
Epoch 13: val_loss did not improve from 0.04645
4094/4094 [==============================] - 9s 2ms/step - loss: 0.0923 - accuracy: 0.9640 - val_loss: 0.0783 - val_accuracy: 0.9732
Epoch 14/200
4073/4094 [============================>.] - ETA: 0s - loss: 0.0918 - accuracy: 0.9639
Epoch 14: val_loss did not improve from 0.04645
4094/4094 [==============================] - 10s 3ms/step - loss: 0.0918 - accuracy: 0.9639 - val_loss: 0.0765 - val_accuracy: 0.9759
Epoch 15/200
4076/4094 [============================>.] - ETA: 0s - loss: 0.0912 - accuracy: 0.9642
Epoch 15: val_loss did not improve from 0.04645
4094/4094 [==============================] - 10s 2ms/step - loss: 0.0913 - accuracy: 0.9642 - val_loss: 0.0981 - val_accuracy: 0.9685
Epoch 16/200
4090/4094 [============================>.] - ETA: 0s - loss: 0.0908 - accuracy: 0.9641
Epoch 16: val_loss did not improve from 0.04645
4094/4094 [==============================] - 10s 2ms/step - loss: 0.0908 - accuracy: 0.9641 - val_loss: 0.0807 - val_accuracy: 0.9745
Epoch 17/200
4078/4094 [============================>.] - ETA: 0s - loss: 0.0903 - accuracy: 0.9648
Epoch 17: val_loss did not improve from 0.04645
4094/4094 [==============================] - 11s 3ms/step - loss: 0.0903 - accuracy: 0.9647 - val_loss: 0.0761 - val_accuracy: 0.9765
Epoch 18/200
4087/4094 [============================>.] - ETA: 0s - loss: 0.0899 - accuracy: 0.9646
Epoch 18: val_loss did not improve from 0.04645
4094/4094 [==============================] - 10s 3ms/step - loss: 0.0899 - accuracy: 0.9646 - val_loss: 0.0791 - val_accuracy: 0.9738
Epoch 19/200
4074/4094 [============================>.] - ETA: 0s - loss: 0.0896 - accuracy: 0.9650
Epoch 19: val_loss did not improve from 0.04645
4094/4094 [==============================] - 9s 2ms/step - loss: 0.0896 - accuracy: 0.9650 - val_loss: 0.0563 - val_accuracy: 0.9833
Epoch 20/200
4076/4094 [============================>.] - ETA: 0s - loss: 0.0893 - accuracy: 0.9650
Epoch 20: val_loss did not improve from 0.04645
4094/4094 [==============================] - 11s 3ms/step - loss: 0.0892 - accuracy: 0.9651 - val_loss: 0.0786 - val_accuracy: 0.9742
Epoch 20: early stopping
```

همانطور که پیداست early-stopping در ایپاک ۲۰ فرآیند را متوقف و مدل با دقت حدود 97% آموزش دیده است. حال مدل را با در نظرگرفتن وزن های ذخیره شده، روی داده های آزمون تست می‌کنیم:

```
# Load the best model
classifier.load_weights('classifier_best_model.h5')
# Denoise the test data
test_loss, test_accuracy = classifier.evaluate(X_test_denoised, y_test)
print(f'Test Loss: {test_loss}')
print(f'Test Accuracy: {test_accuracy}')
```

```
1781/1781 [==============================] - 2s 1ms/step - loss: 0.0464 - accuracy: 0.9882
Test Loss: 0.046448908746242 52
Test Accuracy: 0.9882377982139587
```

همانطور که پیداست دقت مدل 98.8% است.

برای دریافت وزن های مناسب ذخیره شده(در فایل هایی با فرمت HDF5) برای مدل انکودر حذف نویز و طبقه بند از قطعه کد زیر استفاده می کنیم تا وزن های مناسب شبکه را باز گرداند:

```
# Load the best weights for the autoencoder
autoencoder.load_weights('denoising_autoencoder_best_model.h5')
# Inspect the weights of the autoencoder
autoencoder_weights = autoencoder.get_weights()
# Print the weights of each layer in the autoencoder
for i, weight in enumerate(autoencoder_weights):
    print(f"Layer {i} weights: {weight}")
# Load the best weights for the classifier
classifier.load_weights('classifier_best_model.h5')
# Inspect the weights of the classifier
classifier_weights = classifier.get_weights()
# Print the weights of each layer in the classifier
for i, weight in enumerate(classifier_weights):
    print(f"Layer {i} weights: {weight}")
```

بخشی از نتایج مربوط به وزن لایه های شبکه در مدل انکودر حذف نویز و مدل طبقه بند به صورت زیر است:

```
Layer 3 weights: [ 0.07778433  0.13918094  0.03514281 -0.06155046 -0.07253502  0.1245767
  0.13497278  0.1322026   0.10551494 -0.20217383 -0.21862839 -0.36893174
 -0.04823646 -0.06481752 -0.04131066]
Layer 4 weights: [[ 0.44345114 -0.2086769   0.24308246  0.33349097 -0.24482234  0.01861697
  -0.35720623  0.27495348  0.27656618  0.12429209]
 [ 0.329865    1.7656105   1.1632193   0.03549083 -0.21895108 -0.35815215
  -0.15826868  0.10731301  1.0738851  -0.71557933]
 [ 0.44148478  0.06370696  0.34707615  0.34354445 -0.6669157   0.01889822
    0.37073216  0.02444483  0.19359155  0.07013734]
 [-0.1959832  -0.36827895  0.19852376 -0.0264688   0.23347688  0.8648315
  -1.1324766   0.44303015 -0.79634356  0.42909116]
 [-1.5105871  -0.29765105 -0.01025786  0.10143658  0.29074     0.96275973
  -0.07748453  0.9028058  -0.7377127   0.59533954]
 [-0.09651667 -0.17166318 -0.18241246  0.2042318  -0.10634588  0.32696867
  -0.01898557  0.2531663  -0.33296725  0.26901814]
 [-0.24993661 -0.57955605 -0.3453414   0.04469348  0.5292749  -0.17601664
  -0.41780928 -0.32041654 -0.13532098  0.6203744 ]
 [ 1.1108239   0.43791485 -0.31792766 -0.24754892 -0.17842971 -0.51630473
  -0.04007677 -0.84027904  0.09710521 -0.16266699]
 [ 0.20767601 -0.09037714  0.28003094 -0.05820798 -0.2425268   0.55890656
    0.28741348  0.46120822  0.37800318  0.15984459]
 [-0.3612607  -0.8290363   0.57107556  0.8225854   0.25382245  0.56509054
  -0.2397703   0.78606296 -0.8106903   0.36670762]
 [-1.1230202  -1.2995086   0.38304904  0.5601879   0.0049984   0.61190265
    0.07974247  1.172034   -0.31990832  0.6432037 ]
 [ 0.27551952  1.0470293   0.6947729  -0.74086404 -0.44978178 -0.22930232
    0.07777628  0.31246144  0.94102865 -0.5801238 ]
 [-0.9926331  -1.0312406   0.34320858 -0.9955303  -0.31976485  0.8642458
  -0.9321783   1.2437819  -0.52965295  0.703543  ]
 [-0.04932986  0.68204814  0.9713424   0.47059217  0.07997954 -0.4054549
    0.4930853   0.10975994  0.48718408  0.08207272]
 [ 0.37799078  0.25018686 -0.03714423 -0.53690505 -0.07416768  0.07070225
    0.39520738 -0.3738554   0.03491168 -0.12691072]]
Layer 5 weights: [ 0.11422551  0.0616082   0.14558218 -0.31595725 -0.02601363 -0.00401804
 -0.09281346 -0.11035723 -0.05723151  0.07290915]
Layer 6 weights: [[-0.7282761  -0.2037293  -0.8628171   0.92250127  0.2900879 ]
 [-0.44772705 -0.18381749  0.5312636   1.0835774  -0.20759162]
 [ 0.31173947 -0.22315697  0.66095686  0.8142599   0.17215693]
 [ 0.19016941 -0.19259326  0.83091    -0.3471732  -0.11406747]
 [-0.21663094  0.2610403   0.09018795 -0.09373762  0.6659653 ]
 [-0.64599836  0.24703658  0.14983465 -0.40102467  0.23898801]
 [-0.5997974   0.21248068 -0.68216574  0.59388196 -0.46372256]
 [-0.5973574  -0.46622777  0.46012512 -0.25881162  0.81998885]
 [-0.11933745 -0.47698146  0.424647    0.8832795  -0.27761456]
 [ 0.02896627 -0.07782089  0.6508713  -0.64708483  0.3756935311
```

برای هر لایه یک وزن کرنل و یک بایاس گزارش شده است.

در قسمت بعد سوال مدل را با معیارهای مختلف ارزیابی خواهیم کرد.

د

ماتریس درهم ریختگی را روی قسمت آزمون داده ها رسم کنید و مقادیر Accuracy،Precision ، Recall و f1-score را گزارش کنید. فکر می کنید در مسائلی که توزیع برچسب ها نامتوازن است، استفاده از معیاری مانند Accuracy به تنهایی عمل کرد مدل را به درستی نمایش می دهد؟ چرا؟ اگر نه، کدام معیار می تواند به عنوان مکمل استفاده شود؟

در این قسمت داده های تست را به شبکه داده و ماتریس درهم ریختگی و معیارهای ارزیابی مدنظر صورت سوال را با استفاده از کتابخانه sklearn گزارش می کنیم:

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# Load the best weights
classifier.load_weights('classifier_best_model.h5')
# Predict the labels for the test data
y_pred_prob = classifier.predict(X_test_denoised)
# Get the class with the highest probability
y_pred = np.argmax(y_pred_prob, axis=1)
# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
# Calculate metrics
recall = recall_score(y_test, y_pred, average='binary')
precision = precision_score(y_test, y_pred, average='binary')
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='binary')
# Print metrics
print(f'Recall: {recall}')
print(f'Precision: {precision}')
print(f'Accuracy: {accuracy}')
print(f'F1-Score: {f1}')
# Print classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=['Normal', 'Fraud']))
# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Normal', 'Fraud'], yticklabels=['Normal', 'Fraud'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

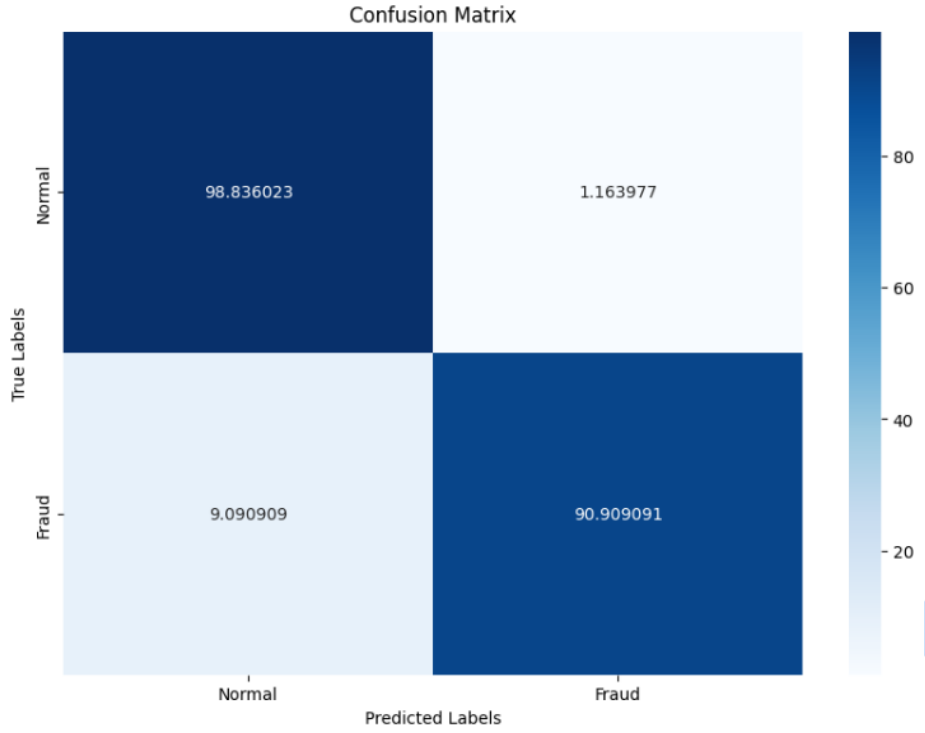
نتایج به‌صورت زیر است:

Recall	**90.9%**
Precision	**10.78%**
Accuracy	**98.82%**
F1-Score	**19.27%**

Classification Report:

```
Classification Report:
              precision    recall  f1-score   support

      Normal       1.00      0.99      0.99     56874
       Fraud       0.11      0.91      0.19        88

    accuracy                           0.99     56962
   macro avg       0.55      0.95      0.59     56962
weighted avg       1.00      0.99      0.99     56962
```

ماتریس در هم ریختگی به صورت زیر است وتعداد داده های کلاس هایی که به درستی تشخیص داده شده و نشده اند را نشان می‌دهد:



98.8% از داده های کلاس نرمال و 90.9% از داده های کلاس تقلب به درستی تشخیص داده شده اند. حال آنکه 1.2% از داده های نرمال و 9.1% از داده های کلاس تقلب اشتباه شناسایی شده اند. این مقادیر نشان دهنده عملکرد نسبتا مطلوب شبکه است.

در پاسخ به بخش دوم سوال مطابق آنچه در بخش ب بحث شد، از معیار دقت به تنهایی نمی توان در دیتاست های نامتوازن استفاده کرد. زیرا اگر مدل تمام نمونه ها را با برچسب کلاس بیشتر لیبل گذاری کند، دقت مدل بالا خواهد بود در حالی که تشخیص ناهنجاری(کلاس اقلیت) در این مدل به خوبی قابل انجام نیست. معیار ارزیابی مدل در دیتاست های نامتعادل معمولا ماتریس در هم ریختگی است که با استفاده از آن می توان از معیار recall استفاده کرد. معیار recall نسبت تعداد داده های به‌درستی شناسایی‌شده یک کلاس به تعداد کل داده های آن کلاس است و ارزیابی می‌کند که چه درصدی از داده های هر کلاس را می‌توان در این مدل طبقه‌بندی شناسایی کرد. بنابراین در مواجهه با دیتاست نامتوازن بهتر است از معیار recall در کنار معیار Accuracy استفاده کرد. بالاتر دیدیم معیار Accuracy، 98.74% و معیار recall، برای کلاس تقلب 90.9% روی شبکه گزارش شد که به معنی عملکرد مطلوب شبکه طراحی شده است. چراکه هر دو معیار مقادیر قابل قبولی دارند.

۵

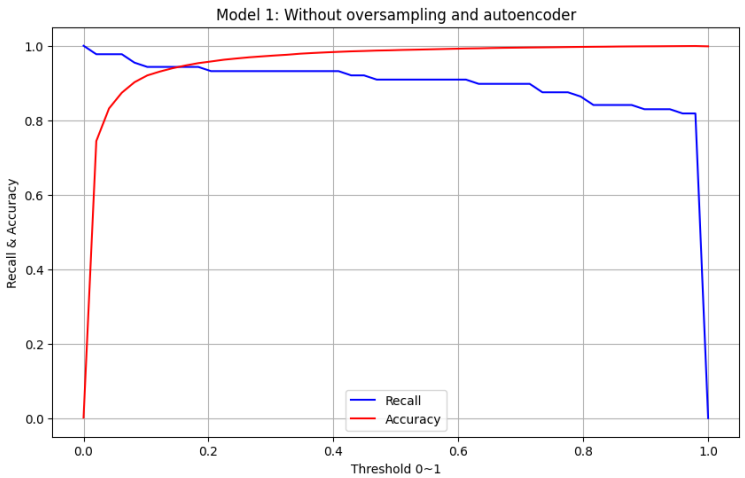
با آستانه های مختلف برای oversampling عمل کرد مدل را بررسی کرده و نمودار Recall & Accuracy را مانند شکل ۷ مقاله ترسیم کنید.

شکل ۷ مقاله ارزیابی شبکه oversampling و اتوانکودر حذف نویز و طبقه بند را با داده آزمون نشان می دهد. در این شکل مقدار آستانه از صفر تا یک تغییر یافته است. نکته قابل توجه این است که این مقدار آستانه، نرخ oversampling نیست. زیرا در شکل ۶ که oversampling هم صورت نگرفته است شاهد محور آستانه هستیم. بنابراین آستانه در اینجا روی احتمال کلاس پیش بینی شده توسط شبکه قرار می گیرد.(پس از پاسخ به سوال اثر نرخ oversampling نیز بررسی می شود.)

بنابراین مقدار آستانه را روی خروجی احتمالاتی شبکه در نظر می گیریم و داده تست را به شبکه داده (وزن های بدست آمده را بارگذاری وpredict انجام می دهیم.) و تغییرات معیار ارزیابی Accuracy و Recall را در هر آستانه رسم می کنیم:

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from imblearn.over_sampling import SMOTE
from sklearn.metrics import recall_score, accuracy_score, precision_score, f1_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
# Set random seeds for reproducibility
np.random.seed(64)
tf.random.set_seed(64)
# Define your oversampling ratios
thresholds = np.linspace(0, 1, 50)
recalls = []
accuracies = []
# Load the best model weights
autoencoder.load_weights('denoising_autoencoder_best_model.h5')
classifier.load_weights('classifier_best_model.h5')
# Denoise the test data using the trained autoencoder
X_test_denoised = autoencoder.predict(X_test)
# Evaluate the model at different thresholds
y_pred_prob = classifier.predict(X_test_denoised)
for threshold in thresholds:
    y_pred = (y_pred_prob[:, 1] >= threshold).astype(int)
    recall = recall_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    recalls.append(recall)
    accuracies.append(accuracy)
# Plot Recall and Accuracy on the same plot
plt.figure(figsize=(10, 6))
plt.plot(thresholds, recalls, label='Recall', color='b')
plt.plot(thresholds, accuracies, label='Accuracy', color='r')
plt.xlabel('Threshold 0~1')
plt.ylabel('Recall & Accuracy')
plt.title('Model 1: Without oversampling and autoencoder')
plt.legend()
plt.grid(True)
plt.show()
```

معیارهایAccuracy و Recall با آستانه های متفاوت روی خروجی به صورت زیر نشان داده می شود:



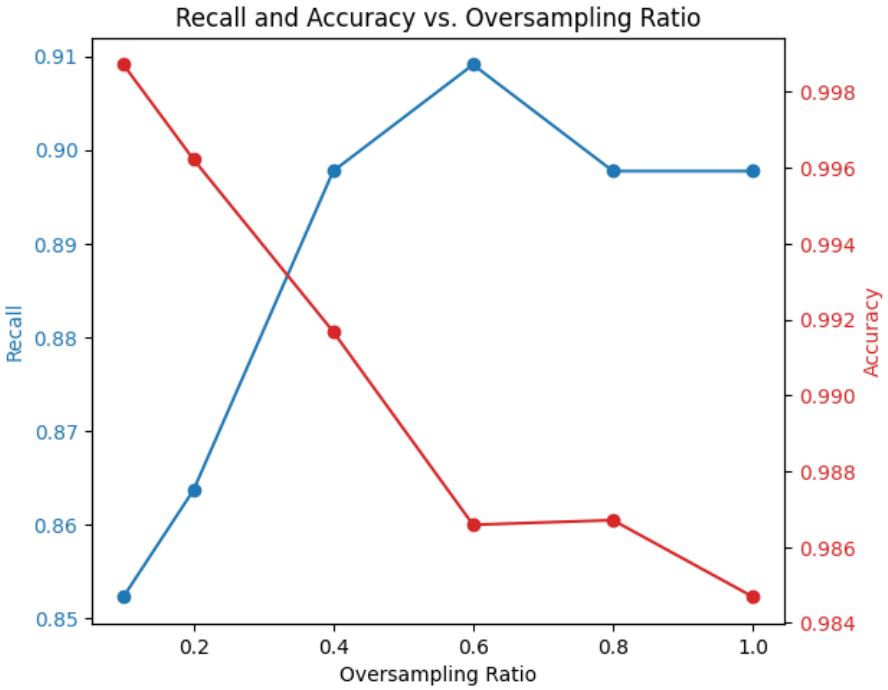
نمودار بالا با شکل ۷ مقاله مطابقت تقریبی دارد و نشان می دهد oversampling و ایجاد دیتای متوازن و استفاده از اتوانکودر حذف نویز، مقادیر بالای معیارهای recall وAccuracy را به همراه دارد.

حال اگر بخواهیم نرخ oversampling را تغییر و تاثیر آن را روی معیار های ارزیابی ذکر شده بررسی کنیم، از sampling_strategy در ماژولSMOTE کتابخانه imblearn استفاده می کنیم. نرخ sampling_strategy میزان ایجاد داده کلاس اقلیت را کنترل می کند برای مثال اگر sampling_strategy=0.5 پس از oversampling داده های کلاس اقلیت نصف داده های کلاس دیگر خواهد بود. بنابراین با رسیدن این نرخ به مقدار یک دیتاست متوازن می شود و انتظار داریم مقدار معیار ارزیابیRecall با افزایش نرخ oversampling بهبود یابد.

برای کد این قسمت ابتدا یک حلقه for برای مقادیر نرخ نمونه برداری ایجاد و در هر مورد پس از انجام نمونه برداری شبکه را آموزش می دهیم و داده های تست را روی شبکه آموزش داده شده ارزیابی می کنیم. در نهایت مقادیر معیارهای Accuracy و Recall را گزارش می کنیم.(برای افزایش سرعت فرآیند، تعداد ایپاک 10 و batch-size=150 در نظر گرفته شد.)

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from imblearn.over_sampling import SMOTE
from sklearn.metrics import recall_score, accuracy_score, precision_score, f1_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
# Set random seeds for reproducibility
np.random.seed(64)
tf.random.set_seed(64)
# Define your oversampling ratios
oversampling_ratios = [0.1, 0.3, 0.4,0.6,0.8,1]
recalls = []
accuracies = []
# Preprocess data
for ratio in oversampling_ratios:
    # Apply SMOTE
    smote = SMOTE(sampling_strategy=ratio, random_state=64)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

    # Add Gaussian noise to the training dataset
    noise_factor = 0.5
    X_train_noisy = X_train_resampled + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train_resampled.shape)

    # Denoise the training data using the trained autoencoder
    from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
    np.random.seed(64)
    tf.random.set_seed(64)
    # Define the checkpoint callback
    checkpoint2 = ModelCheckpoint('denoising_autoencoder_model.h5', monitor='val_loss', mode='min', save_best_only=True, verbose=1)
    # Early stopping to avoid overfitting
    early_stopping2 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
    # Train the model
    autoencoder.fit(X_train_noisy, X_train_resampled,
                                epochs=10,
                                batch_size=150,
                                shuffle=True,
                                validation_data=(X_valid, X_valid),
                                callbacks=[checkpoint2, early_stopping2])
    X_train_denoised = autoencoder.predict(X_train_noisy)
    X_valid_denoised = autoencoder.predict(X_valid)

    # Define callbacks
    checkpoint3 = ModelCheckpoint('classifier_model.h5', monitor='val_loss', mode='min', save_best_only=True, verbose=1)
    early_stopping3 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)

    # Train the classifier
    classifier.fit(X_train_denoised, y_train_resampled,
                                    epochs=10,
                                    batch_size=150,
                                    shuffle=True,
                                    validation_data=(X_valid_denoised, y_valid),
                                    callbacks=[checkpoint3, early_stopping3])
    # Load the best model weights
    autoencoder.load_weights('denoising_autoencoder_model.h5')
    classifier.load_weights('classifier_model.h5')
    # Denoise the test data using the trained autoencoder
    X_test_denoised = autoencoder.predict(X_test)
    # Predict the labels for the test data
    y_pred_prob = classifier.predict(X_test_denoised)
    y_pred = np.argmax(y_pred_prob, axis=1)
    # Calculate metrics
    recall = recall_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    recalls.append(recall)
    accuracies.append(accuracy)
# Plot Recall and Accuracy on the same plot
fig, ax1 = plt.subplots()
color = 'tab:blue'
ax1.set_xlabel('Oversampling Ratio')
ax1.set_ylabel('Recall', color=color)
ax1.plot(oversampling_ratios, recalls, marker='o', color=color, label='Recall')
ax1.tick_params(axis='y', labelcolor=color)
ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
color = 'tab:red'
ax2.set_ylabel('Accuracy', color=color)  # we already handled the x-label with ax1
ax2.plot(oversampling_ratios, accuracies, marker='o', color=color, label='Accuracy')
ax2.tick_params(axis='y', labelcolor=color)
fig.tight_layout()  # otherwise the right y-label is slightly clipped
plt.title('Recall and Accuracy vs. Oversampling Ratio')
plt.show()
# Evaluate the best model on test data
print("Best Model Evaluation on Test Data:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=['Not Fraud', 'Fraud']))
```

نتیجه به صورت زیر رسم می شود:



نرخ نمونه برداری حدود 0.1 که نسبت به سایر نرخ های مورد بررسی کمترین تعداد نمونه را اضافه می کند، recall پایینی دارد که نشان دهنده عدم توانایی مدل در تشخیص کلاس اقلیت است.

با افزایش نرخ نمونه برداری بهبود قابل توجهی در معیار recall داریم. به طوری که با نرخ حدود 0.1 مقداری معادل 85% دارد و با افزایش نرخ نمونه برداری به 90% و بالاتر می رسد.

با افزایش نرخ نمونه برداری، recall بهبود می یابد. معیار Accuracy اگرچه در برخی نرخ ها کاهش داشته، اما به طور کلی مقدار قابل قبولی بین 98% تا 99% برای مدل گزارش می کند.

و

مدل را با استفاده از داده های نامتوازن و بدون حذف نویز، آموزش داده و موارد بخش قبلی را گزارش کنید و نتایج دو مدل را با هم مقایسه کنید.

مدل طبقه بند در اینجا به صورت یک شبکه عصبی کاملاً پیوسته (fully-connected neural network) تعریف شده است و دیگر ازoversampling برای بالانس کردن دیتاست و از autoencoder برای حذف نویزاستفاده نشده است. داده های آماده شده در قسمت الف را به آموزش و آزمون تقسیم می کنیم و ورودی نویزی را به مدل طبقه بند داده و آن را آموزش می دهیم. سپس با استفاده از داده های تست و ماتریس در هم ریختگی مدل را ارزیابی می کنیم:

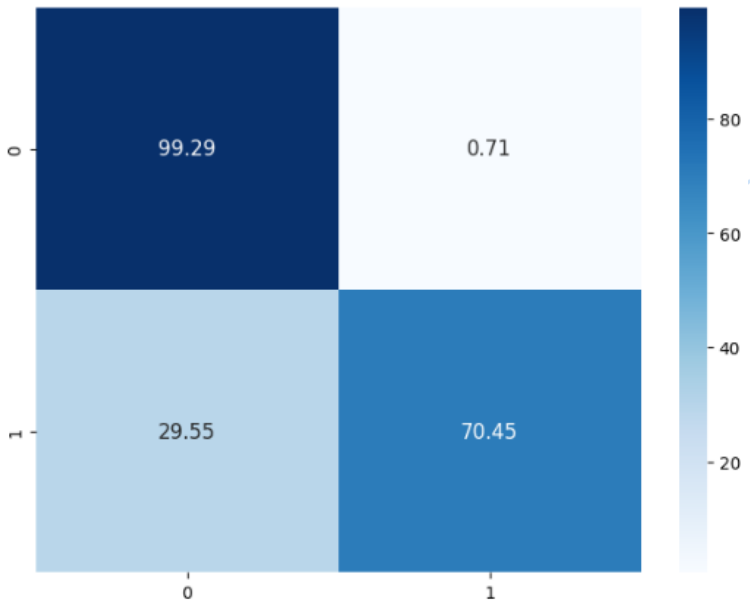
```python
train_data, test_data = train_test_split(data, test_size=0.2, random_state=64)
train_data, valid_data = train_test_split(train_data, test_size=0.1, random_state=64)
# Separate features and labels
X_train = train_data.drop(columns=['Class'])
y_train = train_data['Class']
X_test = test_data.drop(columns=['Class'])
y_test = test_data['Class']
X_valid = test_data.drop(columns=['Class'])
y_valid = test_data['Class']
noise_factor = 0.5
X_train_noisy = X_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=X_train.shape)
classifier = Sequential([
    Dense(22, input_dim=29, activation='relu'),
    Dense(15, activation='relu'),
    Dense(10, activation='relu'),
    Dense(5, activation='relu'),
    Dense(2, activation='softmax')
])
# Compile the classifier
classifier.compile(optimizer=Adam(), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
np.random.seed(64)
tf.random.set_seed(64)
# Define the checkpoint callback
checkpoint1 = ModelCheckpoint('classifier.h5', monitor='val_loss', mode='min', save_best_only=True, verbose=1)
# Early stopping to avoid overfitting
early_stopping1 = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
# Train the classifier
history_classifier = classifier.fit(X_train_denoised, y_train_resampled,
                                    epochs=200,
                                    batch_size=100,
                                    shuffle=True,
                                    validation_data=(X_valid, y_valid),
                                    callbacks=[checkpoint1, early_stopping1])
classifier.load_weights('classifier.h5')
y_pred = classifier.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
cf_matrix = confusion_matrix(y_test, y_pred)
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)
print("\nClassification Report:\n", class_report)
```

نتایج این مدل به صورت زیر است:

Classification Report:

	precision	recall	f1-score	support
0.0	1.00	0.99	1.00	56874
1.0	0.13	0.70	0.22	88
accuracy			0.99	56962
macro avg	0.57	0.85	0.61	56962
weighted avg	1.00	0.99	1.00	56962

ماتریس درهم ریختگی مدل:



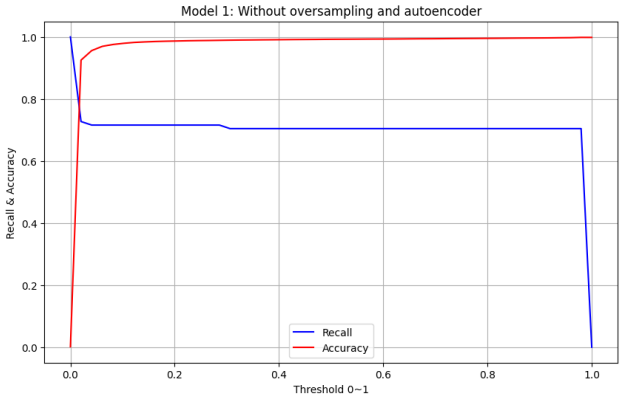
در ماتریس درهم ریختگی مشاهده می شود که 99.3% از داده های کلاس صفر(داده های نرمال) به درستی تشخیص داده شده اند اما 29.55% از داده های تقلب اشتباها سالم تشخیص داده شده است. همچنین 70.45% از داده های کلاس یک(داده های تقلب) به درستی تشخیص داده شده است و 0.7% داده های نرمال اشتباها تقلب تشخیص داده شده است.

با مقایسه نتایج این قسمت و قسمت د، در می یابیم شبکه با حذف قسمتoversampling و اتوانکودر حذف نویز عملکرد ضعیفی از خود نشان داد است. در واقع مدل داده های کلاس اکثریت را بیشتر دیده است و به همین خاطر بسیاری از داده های کلاس اقلیت را نیز، نرمال تشخیص داده است. بنابراین اگر دیتاست نامتوازن باشد، احتمال overfit روی داده های کلاس اکثریت افزایش می یابد.

برای رسم نمودار های Accuracy و recall با در نظر گرفتن آستانه های متفاوت روی خروجی احتمالاتی مانند قسمت قبل داریم:

```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from imblearn.over_sampling import SMOTE
from sklearn.metrics import recall_score, accuracy_score, precision_score, f1_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
# Set random seeds for reproducibility
np.random.seed(64)
tf.random.set_seed(64)
# Define your oversampling ratios
thresholds = np.linspace(0, 1, 50)
recalls = []
accuracies = []
# Load the best model weights
classifier.load_weights('classifier.h5')
# Evaluate the model at different thresholds
y_pred_prob = classifier.predict(X_test)
for threshold in thresholds:
    y_pred = (y_pred_prob[:, 1] >= threshold).astype(int)
    recall = recall_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    recalls.append(recall)
    accuracies.append(accuracy)
# Plot Recall and Accuracy on the same plot
plt.figure(figsize=(10, 6))
plt.plot(thresholds, recalls, label='Recall', color='b')
plt.plot(thresholds, accuracies, label='Accuracy', color='r')
plt.xlabel('Threshold 0~1')
plt.ylabel('Recall & Accuracy')
plt.title('Model 1: Without oversampling and autoencoder')
plt.legend()
plt.grid(True)
plt.show()
```

نمودار به صورت زیر رسم می شود:



همانطور که مشاهده می شود بدون استفاده از oversampling و اتوانکودر حذف نویز، معیار recall مقدار نامطلوبی دارد، زیرا مدل تمام نمونه ها را به عنوان کلاس نرمال طبقه بندی می کند، و اکثر تراکنش های تقلب شناسایی نمی شوند. نتیجه گیری حاصل از نمودار رسم شده با شکل ۶ مقاله مطابقت دارد.

مراجع

[1] https://github.com/MJAHMADEE/MachineLearning2024W
[2] https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html