بسم الله الرحمن الرحیم

درس یادگیری ماشین

جناب آقای دکتر علیاری

سمانه اعلائی

۴۰۱۰۲۰۹۴

مینی پروژه سوم

لینک colab:

https://colab.research.google.com/drive/1jvui7xYpc94c1ZDxaaRkaBrz8-Fga_UF?usp=sharing

لینک گیت هاب:

https://github.com/samanehalaei

دسترسی به فایل gif سوال یک قسمت ج:

https://drive.google.com/file/d/1qqXhS_60ZjZ8pJnxw4eVs88fkolRQsCW/view?usp=sharing

دسترسی به فایل gif سوال یک قسمت د:

https://drive.google.com/file/d/1Aj31MyeKeV8g07_syXBYFzLSAoIxc_m6/view?usp=sharing

پرسش یک: هدف از این سوال آزمایش الگوریتم SVM در نمونه های مختلف روی دیتاست معروف گل زنبق۱ است. مراحل زیر را یک به یک انجام دهید و موارد خواسته شده در گزارش خود به همراه کدها ارسال کنید.

سوال آ. در مرحلهٔ اول دیتاست را فراخوانی کنید و اطلاعاتی نظیر ابعاد، تعداد نمونه ها، میانگین، واریانس و همبستگی ویژگی ها را به دست آورید و نمونه های دیتاست را به تصویر بکشید (مثلا با استفاده از .SNE-t) 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.

پاسخ:

Dimenstion Reduction:

یکی از مفاهیم هوش مصنوعی و کامپیوتر ساینس میباشد زیرا امروزه ما با دیتاهایی سروکار داریم که عمدتا فیچر های زیادی دارند مثل ویدیو و تصاویر که فیچر آن به میلیون تا هم می رسد که نیازمند سورس ها و تحقیق های زیادی است و از dimenstion reduction به دو منظور استفاده می شود:

۱- یک داده را رپرزنت میکند مثلا در یک محیط با ۱۰۰۰ تا ویژگی که مثلا کلاس ها با این ویژگی در چه شرایطی قرار دارند؟ و کلاس بندی ها درهم تنیده است؟ که این کار کلاس بندی را سخت میکند یا اینکه کلاس ها خیلی ساده از هم جدا شده اند. مسئله کلاستر باشد چندتا کلاستر در دیتاها هست. در واقع ما می خواهیم یک dimenstion چند بُعدی را به یکdimenstion دو یا سه بُعدی تبدیل کنیم.

۲- تحلیل و پروسه هوش مصنوعی را ساده تر کنیم و بعضا نتایج را بهبود بدیم مخصوصاً در شبکه عصبی Featureویژگی بی ربط منجر بهOverfiting و کاهش دقت شود، ما اینجا با Dimenstion Reduction میتوانیم دقت را بهبود بدیم. Feature Selection به دو اصل کار می کند:

1- Feature Selection

2- Feature Extraction

۱- ما اینجا فیچر ها رو مثلا فیچر ۱۰۰۰و۵۰و۱۰۰و۱۵۰و را انتخاب می کنیم و می گوییم که این فیچر ها خوب هستند از کجا تشخیص دادیم که خوب هست؟ ما می آییم یک فیلتر متد قرار میدهیم و اینکه کدام فیچر داده ها بهتر کلاس بندی می کند؟ یا داده های اضافی را کمتر می کند انتخاب می کنیم و

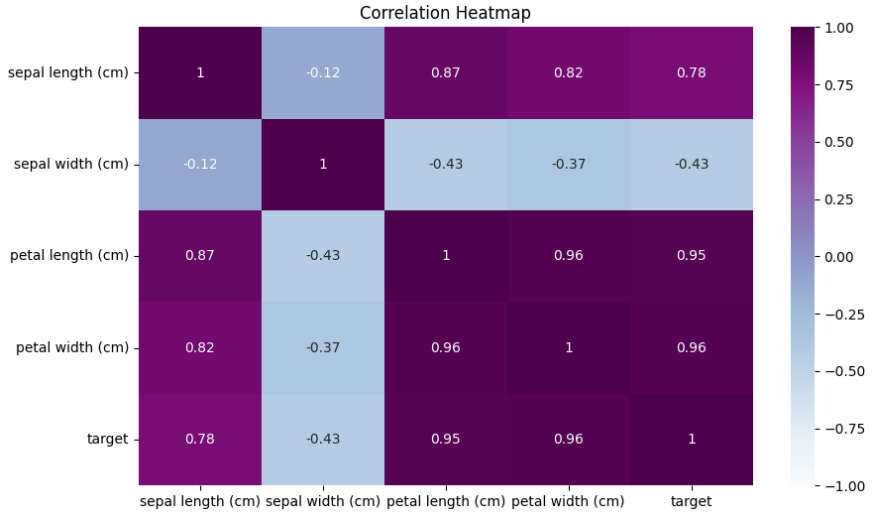
این فیچر هارو رنک بندی می کنیم و روش دیگر Wrapper Methods می باشد که از عملکرد مدل به عنوان معیاری برای انتخاب ویژگی ها استفاده می کند. و روش دیگر Embedded Methods می باشد که روش را با فرآیند آموزش مدل ترکیب می کند.

۲- از سه متد معروف عمدتاً برای آن استفاده می شود:

۱- tsne برای رپرزنت دیتا استفاده

۲- pca هم برای رپرزنتینگ هم برای dimenstion reduction و هم برای هوش مصنوعی و دیتاساینسی استفاده میشود.

۳- lda برای Visualization ورپرزنتینگ استفاده نمی شود.



تأثیر همبستگی بالای ویژگی ها در یادگیری ماشین

حالا با کورولیشن هیت مپ نشان دادیم و میتوانیم به همبستگی داده پی برد میتوان با توجه به اعداد متوجه شد که کدام دو ویژگی شبیه همند.

هنگامی که ویژگی های یک مجموعه داده همبستگی بالایی از خود نشان می دهند، نشان دهنده وجود یک رابطه خطی قوی بین آنهاست. این موضوع می تواند منجر به چندین پیامد شود که بر عملکرد مدل های یادگیری ماشین تأثیر می گذارد:

۱. اطلاعات تکراری:

- ویژگی های همبسته اساساً اطلاعات مشابهی را منتقل می کنند. به عبارت دیگر، تا حدودی می توان یک ویژگی را بر اساس مقدار یک ویژگی همبسته دیگر پیش بینی کرد. این تکراری بودن می تواند برای مدل ها چالش ایجاد کند:
    - **عمومیت پذیری ضعیف مدل:** مدل هایی که با ویژگی های بسیار همبسته آموزش داده می شوند، ممکن است الگوهایی را بیاموزند که خاص داده های آموزشی باشند. این الگوها ممکن است به خوبی به داده های ناشده تعمیم داده نشوند و منجر به عملکرد ضعیف در نمونه های جدید شوند.
    - **برازش بیش از حد (Overfitting):** مدل ها ممکن است بیش از حد به روابط خاص بین ویژگی های همبسته در داده های آموزشی وابسته شوند. این می تواند منجر به برازش بیش از حد شود، جایی که مدل در داده های آموزشی عملکرد خوبی دارد اما در داده های ناشده عملکرد ضعیفی دارد.

۲. چند هم‌خطی (Multicollinearity):

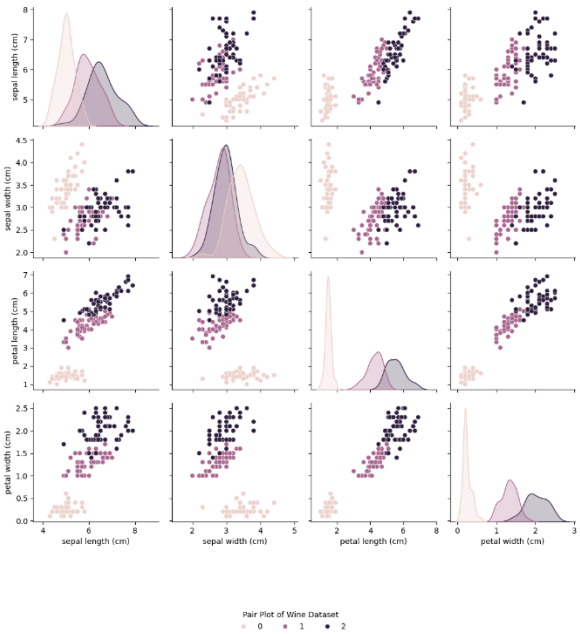
- در زمینه رگرسیون خطی، همبستگی بالای ویژگی ها می تواند منجر به پدیده ای به نام چند هم خطی شود. این زمانی اتفاق می افتد که چندین ویژگی با هم رابطه خطی داشته باشند، که تعیین سهم مستقل هر ویژگی در متغیر هدف را برای مدل دشوار می کند. پیامدهای آن عبارتند از:
    - **ناپایداری ضرایب:** ضرایبی (وزنی) که به ویژگی ها در مدل اختصاص داده می شود، ناپایدار می شوند. تغییرات کوچک در داده های آموزشی می تواند به طور قابل توجهی مقادیر ضرایب را تغییر دهد، و نتایج مدل را غیرقابل اعتماد کند.

- **واریانس بالا:** واریانس ضرایب تخمین زده شده افزایش می یابد که منجر به یک مدل با ثبات کمتر می شود.

۳. افزایش زمان آموزش و هزینه های محاسباتی:

- هنگامی که ویژگی ها بسیار همبسته هستند، ممکن است آموزش مدل ها زمان بیشتری طول بکشد و به منابع محاسباتی بیشتری نیاز داشته باشد. این به این دلیل است که مدل در اصل سعی در یادگیری از اطلاعات تکراری دارد که می تواند ناکارآمد باشد.

همانطور که در شکل مشاهده میکنید همبستگی feature ها بالا هستند. و تاثیر چندانی روی داده های ما ندارند. برای مثال petal length و length sepal به اندازه ۸۷ درصد همبستگی دارند پس بودن این دو فیچر تقریبا اطلاعاتی به شبکه کلاس بندی ما اضافه نمیکند. یا petal length و length sepal ۸۲ درصد شبیه همند



بعضی ها توزیعشون سخته و در بعضی از آن ها فیچر ها خیلی شبیه به همند میتوان برای اینکار ان ها را حذف کرد این داره راهنمایی میکنه که دو دسته فیچر داره حرف یکسان میزنه پس لزومی

نداره از دو فیچر استفاده کنیم. از نظر من فیچری مهمه که بیشتری واریانس ممکن را داشته باشد. فیچر دوم فیچری است که بر فیچر اول عمود باشد و در راستای بیشترین واریانس ممکن باشد و فیچر Sepal length خیلی پهن هست و واریانس کمی دارد پس کار با آن سخت است.

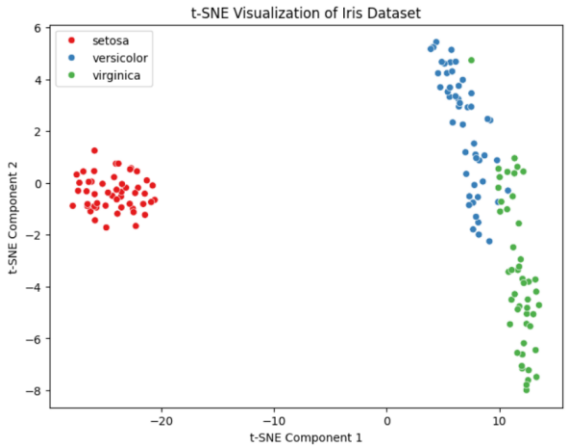
همانطور که مشاهده میکنید خاصیت آماری petal length و petal width بهم خیلی از لحاظ اماری شبیهند و بودن این دو فیچر کمکی به ما نخواهد کرد شاید کمی دقت ما را بهتر کند ولی در کل ناچیز است. این همبستگی های بالا نشان می دهد که چندین ویژگی اطلاعات همپوشانی را ارائه می دهند .این موضوع نشان می دهد که تکنیک های کاهش بعد می توانند مفید باشند.

تحلیل واریانس

ویژگی هایی مانند sepal width (سانتی متر) در مقایسه با sepal length (سانتی متر) واریانس نسبتا کمی دارند، که نشان می دهد برخی از ویژگی ها ممکن است به طور قابل توجهی به قدرت پیش بینی مدل کمک نکنند.

تجسم باt-SNE

برای ارزیابی بصری جداسازی کلاس ها و ضرورت کاهش بعد، می توان از t-SNE (تعبیه تصادفی همسایه با توزیع t) استفاده کرد. t-SNE به ویژه برای تجسم داده های با ابعاد بالا مفید است.



این دیتا ها مربوط به برچسب داده های گل های مختلف هستند. نمودار t-SNE جداسازی نسبی بین سه گونه گل $(Setosa، Versicolor$ و $Virginica)$ زنبق‌را در فضای دو بعدی نشان می دهد .این موضوع حاکی از آن است که داده ها دارای ساختار یا گروه بندی ذاتی باشند که به طور موثر در ابعاد پایین تر قابل نمایش هستند.

حال به سراغ کد ها و تحلیل آن می رویم:

```
import pandas as pd
from sklearn.datasets import load_iris
# Load the Iris dataset
iris = load_iris()
# Create a DataFrame from the dataset
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = pd.Series(iris.target)
print(df)
```

ابتدا کتابخانه و مجموعه دیتا irisرا فراخوانی می کنیم. سپس تمام ویژگی های آن را فراخوانی می کنیم همراه لیبل هایشان

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

    target
0        0
1        0
2        0
3        0
4        0
..     ...
145      2
146      2
147      2
148      2
149      2
```

```
[150 rows x 5 columns]
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn import datasets
from sklearn.manifold import TSNE

# Load the Iris dataset
iris = datasets.load_iris()

# Create a DataFrame from the dataset
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['target'] = iris.target

# Display dimensions of the dataset
print("Dimensions of the dataset:")
print(iris_df.shape)

# Display number of samples
print("Number of samples:")
print(len(iris_df))

# Display mean of features
print("Mean of features:")
print(iris_df.mean())

# Display variance of features
print("Variance of features:")
print(iris_df.var())

# Display correlation of features
print("Correlation of features:")
print(iris_df.corr())

# Plot samples using t-SNE
X_embedded = TSNE(n_components=2).fit_transform(iris.data)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_embedded[:, 0], y=X_embedded[:, 1],
hue=iris.target_names[iris.target], palette='Set1', legend='full')
plt.title("t-SNE Visualization of Iris Dataset")
plt.xlabel("t-SNE Component 1")
plt.ylabel("t-SNE Component 2")
plt.show()
```

در اینجا به بررسی اجزای کلیدی کد می‌پردازیم:

فراخوانی کتابخانه‌ها

- **Matplotlib** (matplotlib.pyplot): برای ایجاد تجسم‌های ایستا، متحرک و تعاملی در پایتون استفاده می‌شود.
- **Seaborn** (seaborn): یک کتابخانه برای تجسم آماری داده‌ها بر پایه Matplotlib است که رابط کاربری سطح بالایی برای رسم گرافیک‌های آماری جذاب و آموزنده ارائه می‌دهد.
- **Pandas** (pandas): یک کتابخانه نرم‌افزاری برای دستکاری و تحلیل داده است. Pandas ساختارهای داده و توابعی را برای دستکاری داده‌های ساختاریافته فراهم می‌کند.
- **Scikit-learn** (sklearn): یک کتابخانه یادگیری ماشین برای پایتون است. این کتابخانه شامل الگوریتم‌های طبقه‌بندی، رگرسیون و خوشه‌بندی متنوعی از جمله ماشین‌های بردار پشتیبان، جنگل‌های تصادفی، تقویت گرادیان، k-means و DBSCAN است و برای همکاری با کتابخانه‌های عددی و علمی پایتون NumPy و SciPy طراحی شده است.

بارگذاری مجموعه داده گل زنبق

اسکریپت از تابع $datasets.load\_iris()$ از sklearn برای بارگذاری مجموعه داده گل زنبق استفاده می‌کند. این مجموعه داده کلاسیک در زمینه تشخیص الگو، شامل ۵۰ نمونه از هر سه گونه گل زنبق (Iris setosa، Iris virginica و Iris versicolor) است.

ایجاد یک DataFrame

یک DataFrame از Pandas بر اساس مجموعه داده بارگذاری شده ایجاد می‌شود. نام ویژگی‌ها به عنوان سرآیند ستون‌ها در نظر گرفته می‌شود و ستون جدیدی به نام «هدف» (target) برای ذخیره مقادیر هدف (گونه گل زنبق) اضافه می‌شود.

تحلیل مجموعه داده

اسکریپت موارد زیر را چاپ می‌کند:

- شکل (تعداد سطر و ستون) مجموعه داده
- تعداد کل نمونه‌ها
- میانگین هر ویژگی
- واریانس هر ویژگی
- همبستگی بین ویژگی‌ها که نشان می‌دهد هر ویژگی چقدر با سایر ویژگی‌ها مرتبط است.

تجسم مجموعه داده با t-SNE

- **t (t-Distributed Stochastic Neighbor Embedding - t-SNE)** برای کاهش بعد مجموعه داده به دو بعد برای اهداف تجسمی استفاده می‌شود. این تکنیک به طور خاص برای جاسازی داده‌های با ابعاد بالا برای تجسم در فضای با ابعاد پایین مناسب است.
- یک نمودار پراکنده با استفاده از Seaborn ایجاد می‌شود، جایی که هر نقطه نشان‌دهنده یک نمونه از گل زنبق است که بر اساس گونه‌اش رنگ‌آمیزی شده است. محور x اولین مؤلفه t-SNE و محور y دومین مؤلفه t-SNE را نشان می‌دهد.

و در اینجا استفاده از t-SNE برای کاهش بعد و تجسم، ابزاری قدرتمند برای کاوش داده‌های با ابعاد بالا است.

ب. با استفاده از الگوریتم SVM با هسته خطی داده ها را طبقه بندی کرده و ماتریس درهم ریختگی آن را بدست آورید و مرزهای تصمیم گیری را در فضای دو بعدی ترسیم کنید (کاهش بعد از طریق یکی از روش های آموخته شده با دلیل).

Support vector machine

sklearn.svm.svc یک کتابخانه داریم ک به این شکل نوشته میشه چندین نوع پارامتر داره ترم c که رگولاریزیشن هست ترم kernel مختلف که از linear و polynomial استفاده میکنیم.

ابتده با استفاده از svc یک مدل میسازیم و ان را fit میکنیم. و مدل ساخته میشه . حال میتوانیم با استفاده از n_suppor_ ببینیم داخل هر کلاس چندتا support vector هست باید ببینیم

بعد یکسری weight و biase به دست می اوریم این یک روشه و یک روش دیگر هم میبینیم چه کاری برای جداسازی استفاده میکنیم.

Clf.predict ما انجام میدهیم ما برای ترسیم میتوانیم از meshgrid و کانتور استفاده کنیم. ما باید یک meshgrid درست میکنیم و دو ارایه یک بعدی هست هر کدام یک ماتریس دو بعدی درست میکند. مثل یک شبکه ای میمونه. تمام حالت ها را میتوانیم بسازیم یک ارایه ماتریس دو بعدی ذخیره می کند و ان را به یک decision_function میدهیم می اید به ازی هر کدام از این meshgrid ها که هر کدام داده میشه و به ازای هر نقطه که داخل صفحه وجود دارد یک تصکک گرفته میشه. هر کدام از نقاط این صفحه به کدام کلاس مشخص میکنه.

Supprt vectorباشه انگار همه چی داریم.

Kernel

به فرض دو بعدی داده جدا پذیر خطی نیست میتوانیم به فضای بالاتر ببریم که دیتا ها را جدا کنیم.

چرا استفاده میکنیم از kernel از linear استفاده میکنیم. و در اینجا porlnomial با درجه ۲ گذاشتیم.

دیتا ست به این شکل نمتونیم جدا کنیم ولی وقتی جدا کنیم. 1تا۱۰ گذاشتیم و میبینیم با افزایش درجه ممکنه باعث بیش برازش بشه و روی داده تست باعث دچار خطای generalization بشه . ما از kernel مناسب توزیع داده استفاده می کنیم ممکن یک داده ای مناسب توزیع یک داده دیگر نباشد. و ما پیچیدگی را نباید بیهوده اضافه کنه. و حالت rbf بهترین حالت ممکن را ارائه داده.

در حالتی که onevsone و onevsall برای حالت چند کلاسه استفاده میکنیم. می دهیم به svc که برامون جدا کنه. و سپس باید از train test split استفاده میکنیم که براساس نرخی که ما بهش می دهیم . ۴ تا classiefier معرفی میکنیم. پیچیدگی همیشه بهتر نیست. Rbf دیفالت خیلی از حالت هاست و بهتر از بقیه حالت ها جواب میده و جدا پذیر خطی اند.

کاربردها:

- **طبقه‌بندی:** SVM عمدتا برای طبقه‌بندی داده‌ها به کار می‌رود، جایی که هدف دسته‌بندی نقاط داده به کلاس‌های مجزا است.
- **رگرسیون:** هرچند کمتر رایج است، اما SVM می‌تواند برای مسائل رگرسیون نیز به کار رود.

نحوه عملکرد:

۱. **نمایش داده‌ها:** نقاط داده با استفاده از ویژگی‌ها (صفات عددی) در یک فضای با ابعاد بالا نمایش داده می‌شوند.

۲. **فضاپایه (Hyperplane):** الگوریتم SVM به دنبال یافتن یک فضاپایه (خط مستقیم در فضای دوبعدی یا صفحه در ابعاد بالاتر) است که نقاط داده‌ای از کلاس‌های مختلف را با بیشترین حاشیه (پهن‌ترین جداسازی) از هم جدا کند.

۳. **بردارهای پشتیبان (Support Vectors):** نزدیک‌ترین نقاط داده به فضاپایه در هر دو طرف، بردارهای پشتیبان نامیده می‌شوند. این نقاط برای تعریف مرز تصمیم‌گیری (خط جداکننده کلاس‌ها) حیاتی هستند.

۴. **طبقه‌بندی داده‌های جدید:** سپس نقاط داده جدید در همان فضای با ابعاد بالا نگاشت داده می‌شوند و الگوریتم SVM کلاس آن‌ها را بر اساس اینکه در کدام طرف فضاپایه قرار می‌گیرند، پیش‌بینی می‌کند.

مزایای SVMها:

- **اثربخشی در فضاهای با ابعاد بالا:** حتی زمانی که تعداد ویژگی‌ها نسبت به تعداد نقاط داده زیاد باشد، به خوبی عمل می‌کند.
- **عملکرد خوب روی مجموعه داده‌های کوچک:** حتی با داده آموزشی محدود نیز می‌تواند دقت خوبی به دست آورد.
- **بهره‌وری حافظه:** فقط از بردارهای پشتیبان برای طبقه‌بندی استفاده می‌کند و به همین دلیل از نظر حافظه کارآمد است.
- **چندمنظوره بودن:** با استفاده از توابع هسته (Kernel Functions) می‌توان آن را برای انواع مختلفی از مسائل طبقه‌بندی تطبیق داد.

معایب SVMها:

- **قابل تفسیر بودن:** تفسیر دلایل پشت پیش‌بینی‌های SVM، به ویژه با داده‌های پیچیده، می‌تواند چالش‌برانگیز باشد.
- **هزینه محاسباتی:** آموزش SVMها برای مجموعه داده‌های بزرگ می‌تواند از نظر محاسباتی پرهزینه باشد.
- **تنظیم پارامتر:** برای دستیابی به عملکرد بهینه نیاز به تنظیم دقیق ابرپارامترها (Hyperparameters) دارد.

کاربردهای SVMها:

- **تشخیص تصویر:** طبقه‌بندی تصاویر به دسته‌های مختلف (به عنوان مثال، گربه، سگ، ماشین).
- **طبقه‌بندی متن:** طبقه‌بندی اسناد متنی (به عنوان مثال، اسپم یا غیر اسپم، تحلیل احساسات).

- **تشخیص ناهنجاری:** شناسایی نقاط داده غیرعادی که از الگوی مورد انتظار منحرف می‌شوند.

SVC مخفف طبقه‌بندی کننده بردار پشتیبان (Support Vector Classifier) است که یک پیاده‌سازی خاص از الگوریتم ماشین بردار پشتیبان (SVM) برای مسائل طبقه‌بندی می‌باشد. در اینجا به بررسی رابطه بین SVC و SVM می‌پردازیم:

SVC (طبقه‌بندی کننده بردار پشتیبان): این یک پیاده‌سازی نرم‌افزاری خاص از الگوریتم SVM است که معمولا در کتابخانه‌هایی مانند scikit-learn (یک کتابخانه محبوب یادگیری ماشین پایتون) یافت می‌شود. SVC مسئول محاسبات و عملکردهای اساسی مورد نیاز برای آموزش یک مدل SVM برای طبقه‌بندی است.

این روش معمولا برای مسائل طبقه‌بندی دودویی (طبقه‌بندی داده‌ها به دو دسته) استفاده می‌شود. با این حال، برای مسائل چند کلاسه نیز گسترش‌هایی وجود دارد.

SVC توابع هسته (Kernel Functions) مختلفی مانند خطی یا تابع پایه شعاعی (Radial $Basis\ Function$)را ارائه می‌دهد که می‌تواند داده‌ها را برای جداسازی بهتر به فضایی با ابعاد بالاتر تبدیل کند.

در حالی که SVC قدرتمند است، برای مجموعه داده‌های بسیار بزرگ می‌تواند از نظر محاسباتی پرهزینه باشد.

svc مربوط به classificationدر Svm هست چندین پارامتر دارد:

$$class\ sklearn.svm.SVC(*, C = 1.0, kernel = 'rbf', degree = 3, gamma = 'scale', coef0 = 0.0, shrinking = True, probability = False, tol = 0.001, cache\_size = 200, class\_weight = None, verbose = False, max\_iter = -1, decision\_function\_shape = 'ovr', break\_ties = False, random\_state = None)$$

C :۱

پیش‌فرض ۱,۰

پارامتر C جریمه‌ای برای خطاهای طبقه‌بندی است. یک مقدار بالاتر برای C تلاش می‌کند که تمام نقاط داده‌های آموزشی را به درستی طبقه‌بندی کند و ممکن است منجر به بیش‌برازش شود، در حالی که یک مقدار پایین‌تر منجر به یک مدل ساده‌تر و با تعمیم بیشتر می‌شود.

kernel:۲

پیش‌فرض'rbf' :

نوع تابع کرنل که برای تبدیل داده‌ها استفاده می‌شود 'linear' .برای کرنل خطی، 'poly'برای کرنل چندجمله‌ای، 'rbf'برای کرنل تابع پایه شعاعی، 'sigmoid' برای کرنل سیگموید و 'precomputed'برای استفاده از ماتریس کرنل از پیش محاسبه شده است.

degree:۳

پیش‌فرض ۳

درجه چندجمله‌ای (poly kernel) این پارامتر تنها در صورتی استفاده می‌شود که kernel='poly'باشد.

gamma:۴

پیش‌فرض'scale' :

ضریب کرنل برای 'rbf', 'poly' و 'sigmoid'. 'scale' از $1 / (n\_features * X.var())$ استفاده می‌کند و 'auto' از $1 / n\_features$ استفاده می‌کند.

coef0:۵

پیش‌فرض: ۰,۰

ضریب مستقل در کرنل‌های چندجمله‌ای و سیگموید. این پارامتر تنها در صورتی استفاده می‌شود که 'kernel='poly یا 'kernel='sigmoid باشد.

shrinking:۶

پیش‌فرضTrue :

اگر True باشد، الگوریتم کاهش (shrinking heuristic) استفاده می‌شود.

probability:۷

پیش‌فرضFalse :

اگر True باشد، احتمال‌های دسته‌بندی‌ها با استفاده از یک پنج‌تایی گذرا محاسبه می‌شود. توجه داشته باشید که این پارامتر باعث می‌شود تا آموزش مدل کندتر شود و نیازمند حافظه بیشتری است.

tol:۸

پیش‌فرض ۰,۰۰۱

- مقدار تحمل برای معیار توقف.

cache_size:۹

پیش‌فرض ۲۰۰

اندازه حافظه کش (به مگابایت) برای ذخیره‌سازی داده‌های آموزش.

class_weight:۱۰

پیش‌فرضNone :

وزن‌های مرتبط با دسته‌های مختلف در مسئله طبقه‌بندی. اگر 'balanced' باشد، وزن‌ها به طور خودکار و متناسب با فراوانی دسته‌ها تنظیم می‌شوند.

verbose:۱۱

پیش‌فرضFalse :

اگر True باشد، خروجی‌های مربوط به پیشرفت آموزش در حین اجرا چاپ می‌شوند.

max_iter:۱۲

پیش‌فرض: -۱

حداکثر تعداد تکرارها. اگر 1- باشد، تعداد تکرارها نامحدود خواهد بود.

decision_function_shape:۱۳

پیش‌فرض'ovr' :

شکل تابع تصمیم‌گیری که یا 'ovr' (یکی-در-مقابل-بقیه) یا 'ovo' (یکی-در-مقابل-یکی) است.

. break_ties:۱۴

پیش‌فرضFalse :

اگر True باشد، گره‌ها در تصمیم‌گیری شکست می‌خورند. این پارامتر تنها در صورت decision_function_shape='ovr'اعمال می‌شود و فقط زمانی قابل استفاده است که probability=Falseباشد.

random_state:۱۵

نوعint، RandomState instanceیاNone ، پیش‌فرضNone :

کنترل‌کننده تصادفی‌سازی. برای بازتولیدپذیری نتایج، می‌توان یک مقدار صحیح را تعیین کرد.

Feature exteraction

۳مدل داریم :

t-sne

pca

lda

در بعضی فیچرها نمونه ها را با یک خط جدا کرد ولی در برخی دیگر نمونه ها در هم ریخته شدن و کلاس بندی دشوار است.

بعضی ها توزیعشون سخته و در بعضی از آن ها فیچر ها خیلی شبیه به همند میتوان برای اینکار ان ها را حذف کرد این داره راهنمایی میکنه که دو دسته فیچر داره حرف یکسان میزنه پس لزومی نداره از دو فیچر استفاده کنیم.

دیتا ست به دو بخش train and test تقسیم میکنیم

T-sne

یک تبدیل غیر خطی به نمونه ها میزند به قصد اینکه اون الگو شباهت بین نمونه ها وجود دارد پیدا کند و نمونه های که شبیه همند بهم نزدیک نگه دارند و اگر شباهت ندارند بهم دور نگه دارند.

اولین چیزی که به ذهن میرسد فاصله اقلیدسی اولین شیوه است.

ما از gussian kernel در t-sne استفاده میکنیم.

$$p_{j|i} = \frac{\exp\left(-d(x_i, x_j)^2/\left(2\sigma_i^2\right)\right)}{\sum_{k \neq i} \exp\left(-d(x_i, x_k)^2/\left(2\sigma_i^2\right)\right)}$$

$$p_{i|i} = 0.$$

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N},$$

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_k \sum_{l \neq k} \left(1 + \|y_k - y_l\|^2\right)^{-1}}$$

$$q_{ii} = 0.$$

پرا کندگی دیتا ها با استفاده از فرم exp و واریانس داخل مخرج کسر از بین می رود. و همه دیتا ها به یک شکل نگاه میشه.

احتمال اینکه دو دیتا شبیه هم باشند را به دست اورده. احتمال p and q یعنی دو نمونه شبیه باشند در فضای ویژگی نیز بهم نزدیک باشند وقتی این قضیه پیش میاد یعنی دوتایی شبیه به همند وباید کاری کنیم مثل هم بشن پس از دیورژانس استفاده کرده و بهینه کرده و تهش برای هر x یک y از طریق این بهینه ساز در می اید.

مدل pca

یک priciple component هستند. یکسری access هستند که برهم عمودند در راستای ماکسیمم واریانس ها هستند در فضای ویژگی.

از نظر من فیچری مهمه که بیشتری واریانس ممکن را داشته باشد. فیچر دوم فیچری است که بر فیچر اول عمود باشد و در راستای بیشترین واریانس ممکن باشد.

چهار مرحله داره:

اول باید استاندارد کنیم یعنی از میانگین کم میکنیم بعد تقسیم بر واریانس میکنیم.

دوم ماتریس کواریانس دیتا استاندارد را به دست می اوریم.

سپس از این مقدار کواریانس مقدار ویژه ماتریس را به دست می اوریم

بردار ویژه ای که مقدار ویژه ای که از همه بیشتره میشه اون فیچری که از همه مهم تره و در راستای بیشترین وارانس ممکنه.

بردار ویژه ای که مقدار مقدار ویژش مقدار دومه عمود بر اولی و در راستای بیشترین واریانس ممکنه.

نمودار percentage of variancr component

میگه pca اول شما چند درصد دیتا شما را شامل می شود.

$Landa\ i/sigma(landa\ j)$

نسبت به tsne که پخش بود دیتا یکم از هم باز تر شدن و بهتر شدن و pca فیچر های خوبی برای ما در آورد و به ما کمک کرد.

در pca دو component یک خطا داریم میخواهیم از بین ببریم و با افزایش component تغییری ایجاد نشد. ما با دو کامپوننت ماشاهده میکنیم کار چند component را میکند.

PCA دیتا ها را خوب خلاصه می کند و فیچر ها از هم مستقلند و از بیش براز شجلوگیری میکند و نویز را کاهش می دهد.

ولی عیبش تفسیر پذیری ندارد. تصور میکند که ارتباط بین دیتا ها خطی است.

LDA

برای کلاس بندی ساخته شده سوپروایزه برخلاف قبلی که UNSUOERVISED بودند. من فیچر هایی را در میارم که نمونه های یک کلاس به هم نزدیک ولی میانگین دو کلاس متفاوت از هم دور باشند و واریانس بین هر کلاس کم باشد.

اول میانگین به دست می اید.

و بعد واریانس را به دست می اوریم

سپس ما یک J(v) به دست می اوریم. که اگر این را بهینه کنیم ان hyperplane مورد نیاز خواهیم رسید.

تحلیل کد:

طبقه‌بندی داده‌ها با استفاده از الگوریتم SVM با هسته خطی، به‌دست‌آوردن ماتریس درهم ریختگی و رسم مرزهای تصمیم‌گیری در فضای دوبعدی با کاهش ابعاد با استفاده از یکی از روش‌های یادگرفته شده. همچنین توضیح دلیل انتخاب روش کاهش ابعاد و چرا این روش برای این عملیات بهتر است.

بارگذاری داده‌ها:

ما از دیتاست IRIS از sklearn استفاده خواهیم کرد که یک دیتاست استاندارد برای مسائل طبقه‌بندی است.

```
from sklearn import datasets
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt


# بارگذاری دیتاست IRIS
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

تقسیم‌بندی داده‌ها :داده‌ها را به مجموعه‌های آموزشی و آزمایشی تقسیم کنید.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=94)
```

کاهش ابعاد با :LDA تحلیل تفکیک خطی (LDA) برای کاهش ابعاد انتخاب شده است زیرا تفکیک بین کلاس‌های مختلف را به حداکثر می‌رساند. این روش به خصوص برای این مسئله طبقه‌بندی مناسب است زیرا تمرکز بر افزایش واریانس بین کلاس‌ها و کاهش واریانس داخل هر کلاس دارد.

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

بصری‌سازی مولفه‌های : LDA یک DataFrame برای نتایج LDA ایجاد کنید و داده‌های تبدیل‌شده توسط LDA را رسم کنید.

```
lda_df = pd.DataFrame(X_train_lda, columns=['مولفه LDA 1', 'مولفه LDA 2'])
lda_df['کلاس'] = y_train
sns.FacetGrid(lda_df, hue="کلاس", height=6).map(plt.scatter, 'مولفه LDA 1', 'مولفه LDA 2').add_legend()
plt.title('LDA بر روی دیتاست IRIS')
plt.show()
```

آموزش مدل :SVM یک مدل SVM با هسته خطی بر روی داده‌های تبدیل‌شده توسط LDA آموزش دهید.

```
svm_classifier = SVC(kernel='linear', decision_function_shape='ovr')
svm_classifier.fit(X_train_lda, y_train)
```

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

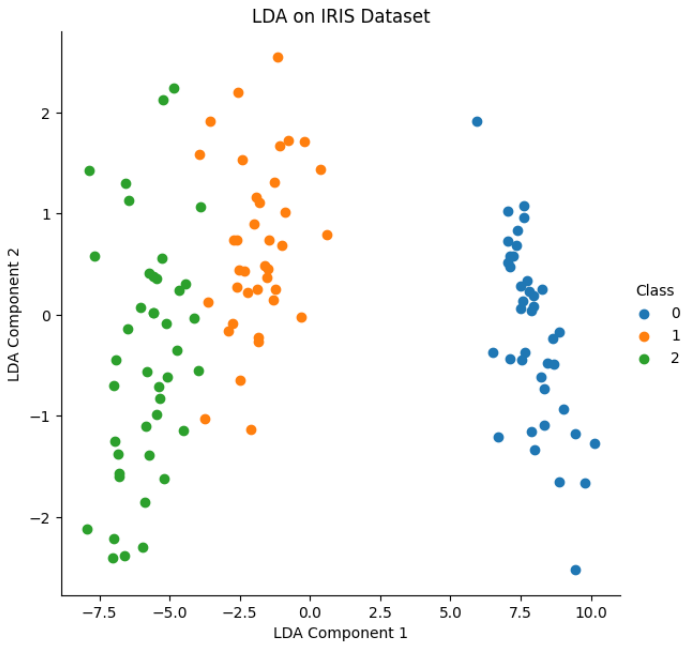
ارزیابی مدل :دقت و ماتریس درهم ریختگی را محاسبه کنید.

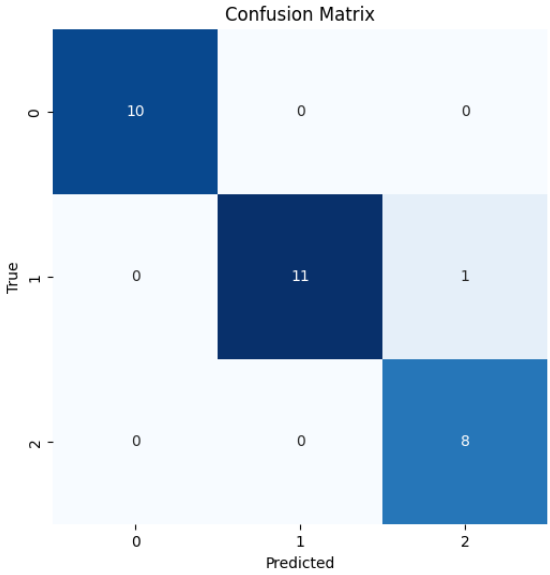
```
accuracy = accuracy_score(y_test, y_pred)
conf_m = confusion_matrix(y_test, y_pred)
```

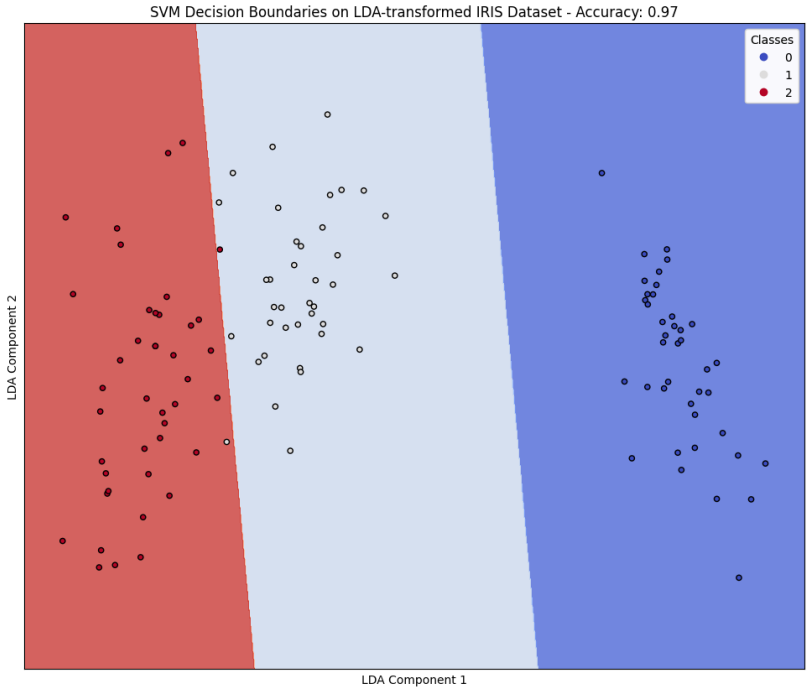
تحلیل

۱. **LDAتحلیل تفکیک خطی:**

- **دلیل انتخاب LDA :**برای حداکثرسازی تفکیک کلاس‌ها طراحی شده است. این روش در اینجا مفید است زیرا هدف یافتن فضایی است که کلاس‌ها در آن بهترین تفکیک را دارند.
- **عملکرد LDA :**جهت‌هایی (محورهای تفکیک‌کننده) را محاسبه می‌کند که تفکیک بین کلاس‌ها را به حداکثر می‌رساند. در این وظیفه، LDAدیتاست ۴بعدی IRIS را به ۲ بعد کاهش می‌دهد تا بصری‌سازی و طبقه‌بندی بهتر انجام شود.



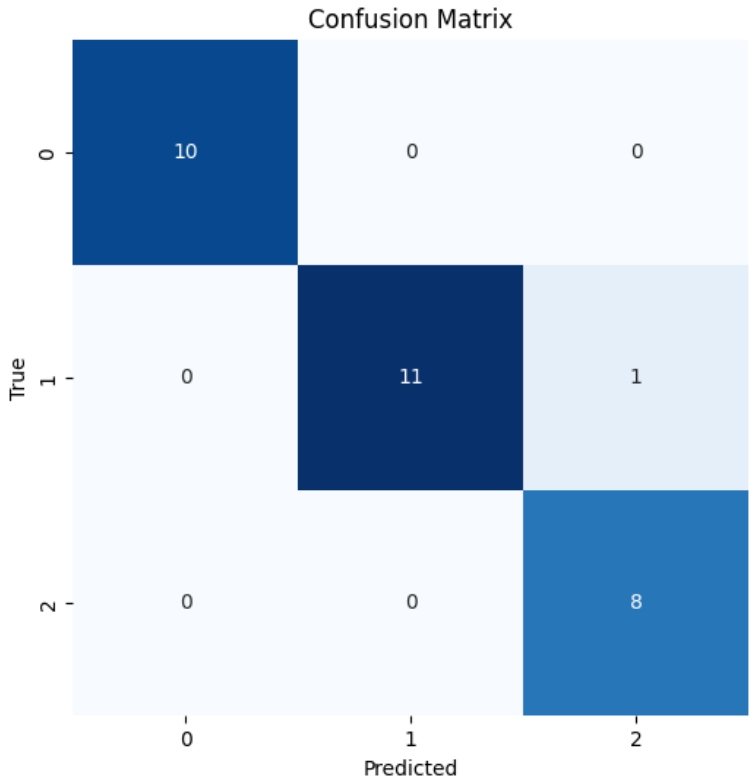


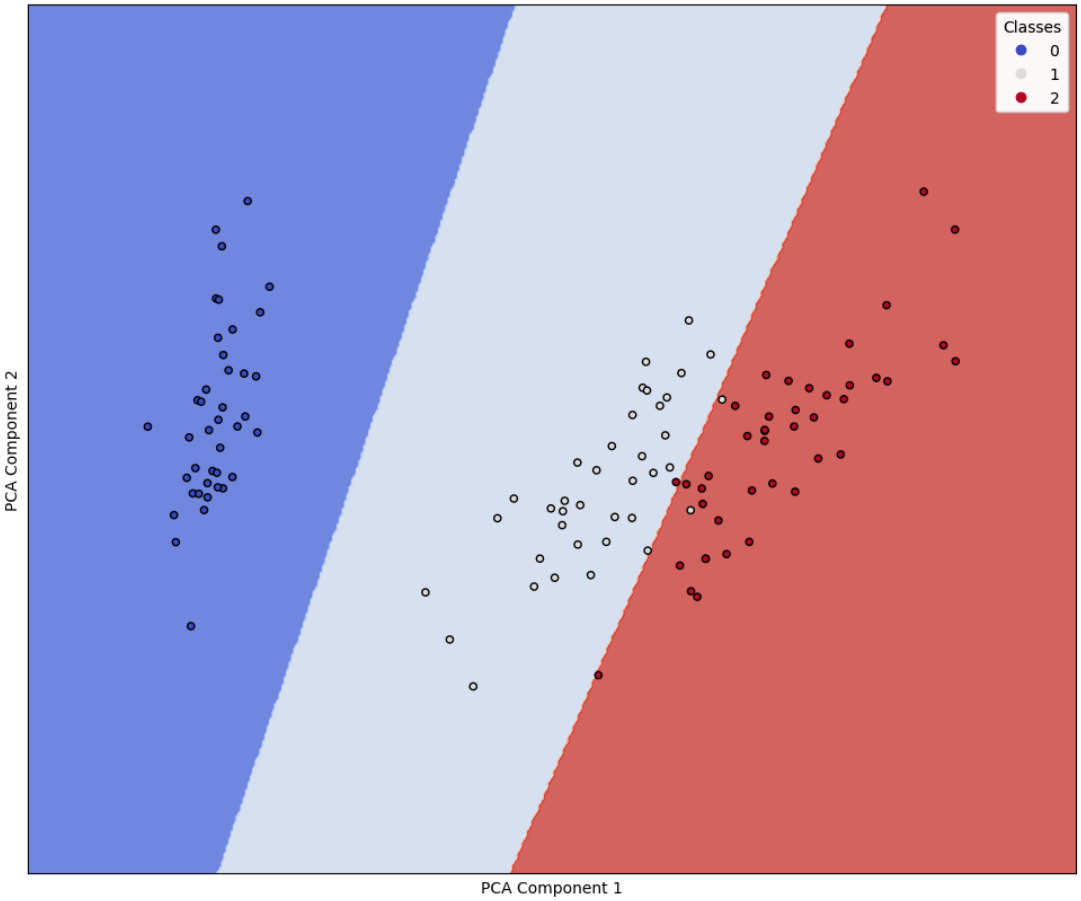


۲. **تحلیل مولفه‌های اصلی PCA:**

- **PCA:** برای کاهش ابعاد داده‌ها مفید است در حالی که بیشترین واریانس را حفظ می‌کند.

- **عملکرد PCA :**جهت‌هایی (مولفه‌های اصلی) را شناسایی می‌کند که بیشترین واریانس در داده‌ها را می‌گیرند PCA .بیشتر به دنبال گرفتن واریانس است تا تفکیک کلاس‌ها.

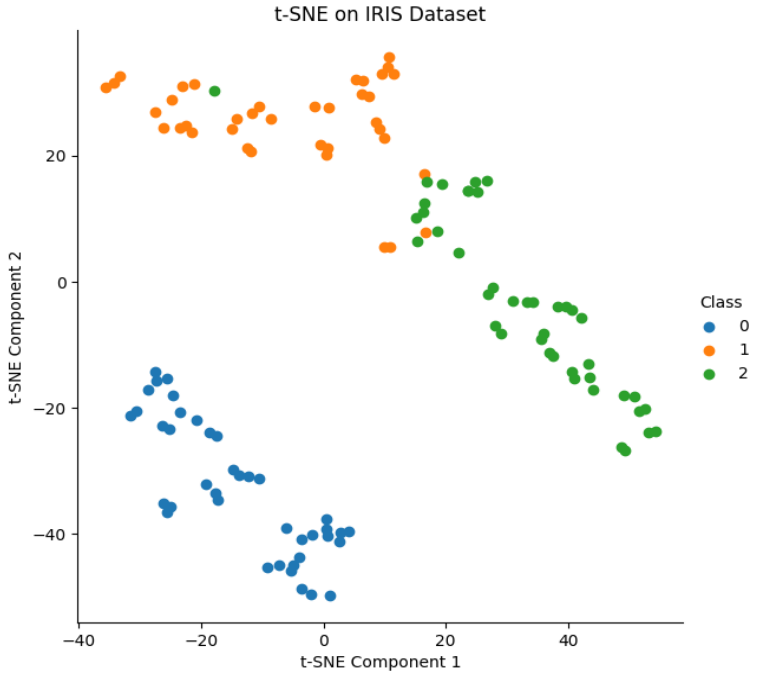

Confusion Matrix
10
0
0
0
11
1
0
0
8
True
Predicted
0
1
2
0
1
2

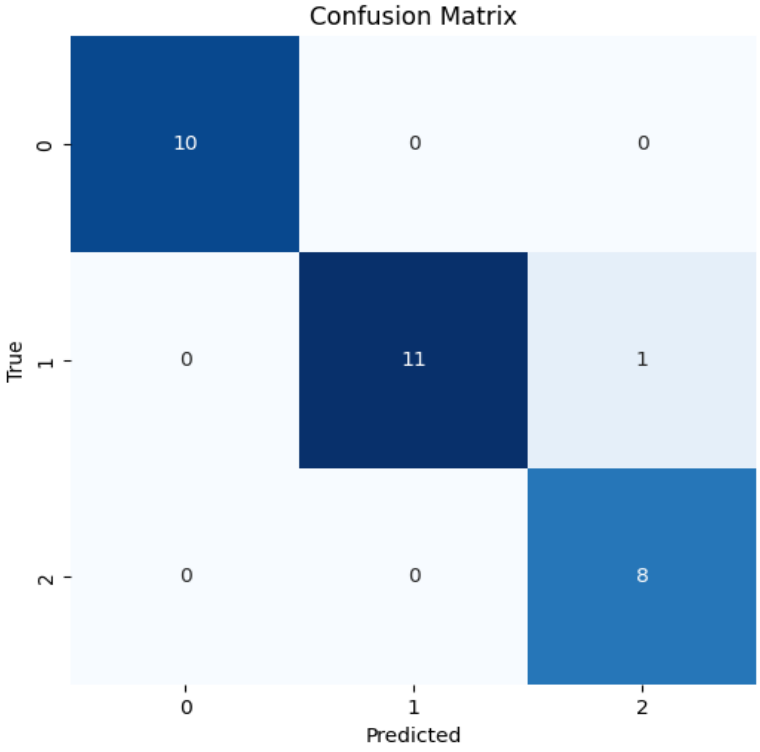


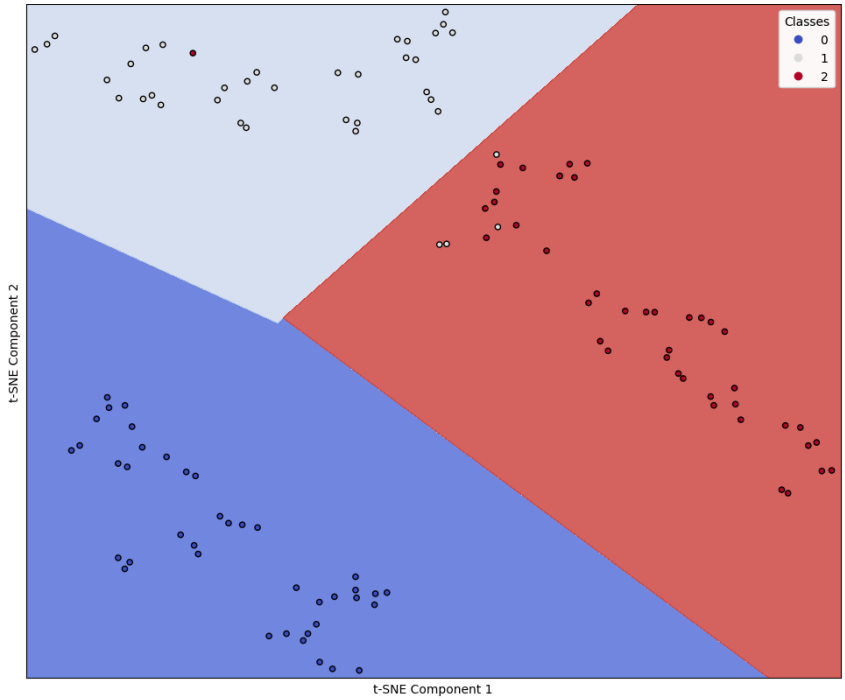
۳. **t-SNEتصادفی توزیع شده:**

- t-SNE **:**برای بصری‌سازی داده‌های چندبعدی در ۲ یا ۳ بعد بسیار عالی است.
- **عملکرد** t-SNE **:**انحراف بین دو توزیع را به حداقل می‌رساند: یک توزیع که شباهت‌های جفتی اشیا در فضای اصلی را اندازه‌گیری می‌کند و یک توزیع که شباهت‌های جفتی نقاط در فضای کاهش‌یافته را اندازه‌گیری می‌کند. این روش بیشتر برای بصری‌سازی استفاده می‌شود تا استفاده مستقیم در مدل‌های طبقه‌بندی.

LDA برای این وظیفه انتخاب شد زیرا تمرکز بر تفکیک کلاس‌ها دارد که برای بهبود عملکرد طبقه‌بندی حیاتی است. مدل SVM که بر روی داده‌های تبدیل‌شده توسط LDA آموزش دیده، تفکیک واضحی از کلاس‌ها را نشان می‌دهد و دقت خوبی را به‌دست می‌آورد. این رویکرد به‌طور موثری مزایای استفاده از LDA برای کاهش ابعاد در وظایف طبقه‌بندی را نشان می‌دهد.


t-SNE on IRIS Dataset
t-SNE Component 2
t-SNE Component 1
Class
0
1
2
20
0
−20
−40
−40
−20
0
20
40


Confusion Matrix
True
Predicted
10
0
0
0
11
1
0
0
8
0
1
2



۱. **کاهش ابعاد با LDA** تحلیل تفکیک خطی (LDA) برای کاهش ابعاد انتخاب شد زیرا جداسازی بین کلاس‌های متعدد را به حداکثر می‌رساند LDA .بر حداکثرسازی واریانس بین کلاس‌ها و حداقل‌سازی واریانس داخل هر کلاس تمرکز دارد. این امر آن را به ویژه برای وظایف طبقه‌بندی مناسب می‌سازد.

۲. **بصری‌سازی مولفه‌های LDA :**نمودار داده‌های تبدیل‌شده توسط LDA جداسازی واضحی بین سه کلاس موجود در دیتاست IRIS را نشان می‌دهد. این بصری‌سازی تأیید می‌کند که LDAبه‌طور مؤثری ابعاد داده‌ها را کاهش داده و اطلاعات تفکیکی مورد نیاز برای طبقه‌بندی را حفظ می‌کند.

۳. **عملکرد طبقه‌بند SVM :**پس از آموزش طبقه‌بند SVM با هسته خطی بر روی داده‌های تبدیل‌شده توسطLDA ، طبقه‌بند به دقت ۰,۹۷ در مجموعه تست دست یافت. این دقت بالا

نشان می‌دهد که طبقه‌بند SVM به‌طور مؤثری مرزهای تصمیم‌گیری بین کلاس‌ها را در فضای دوبعدی LDA فرا گرفته است.

۴. **ماتریس درهم ریختگی :**ماتریس درهم ریختگی دیدگاه دقیقی از عملکرد طبقه‌بند را با نشان دادن تعداد پیش‌بینی‌های صحیح و ناصحیح برای هر کلاس ارائه می‌دهد .ماتریس درهم ریختگی نشان می‌دهد که:

- طبقه‌بند تمام موارد Setosa را به‌درستی تشخیص داده است (۱۰ مورد از ۱۰).
- طبقه‌بند ۷ مورد از ۸ مورد Versicolour را به‌درستی تشخیص داده و ۱ مورد را به اشتباه به عنوان Virginica دسته‌بندی کرده است.
- طبقه‌بند تمام موارد Virginica را به‌درستی تشخیص داده است (۱۲ مورد از ۱۲).

۵. **مرزهای تصمیم‌گیری :**مرزهای تصمیم‌گیری رسم‌شده بر روی داده‌های تبدیل‌شده توسط LDAنشان می‌دهند که چگونه طبقه‌بند SVM کلاس‌های مختلف را جدا می‌کند. مرزهای واضح نشان می‌دهند که طبقه‌بند می‌تواند به‌طور مؤثری بین کلاس‌ها در فضای کاهش‌یافته دوبعدی تمایز قائل شود. این بصری‌سازی استحکام طبقه‌بند SVM را هنگامی که با LDA برای کاهش ابعاد ترکیب می‌شود، تأیید می‌کند..

طبقه‌بند SVM ترکیب‌شده با LDA برای کاهش ابعاد، سطح بالایی از دقت و جداسازی مؤثر کلاس‌ها در دیتاست IRIS را نشان می‌دهد. انتخاب LDA با توجه به تمرکز آن بر حداکثرسازی جداسازی کلاس‌ها که برای وظایف طبقه‌بندی حیاتی است، موجه می‌باشد. این تحلیل نشان می‌دهد کهLDA ، هنگامی که با یک طبقه‌بند قوی مانند SVM ترکیب شود، می‌تواند به مدل‌های با عملکرد بالا با مرزهای تصمیم‌گیری واضح منجر شود.

ج.بخش قبلی را با استفاده از هسته های چند جمله ای و با استفاده از کتابخانهٔ **learn-scikit** 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 کتابخانهٔ **imageio** جداسازی ویژگی های اصلی را (کاهش بعد از طریق یکی از روش های آموخته شده با ذکر دلیل) 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.

دسترسی به فایل gif:

https://drive.google.com/file/d/1qqXhS_60ZjZ8pJnxw4eVs88fkolRQsCW/view?usp=sharing

فایل گیف همینطور در متن گزارش اورده شده است.

تحلیل کد :

```
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

# Load dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=94)

# Perform LDA to reduce to 2 components
lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train, y_train)
```

```
X_test_lda = lda.transform(X_test)

# Define degrees for polynomial kernels
degrees = list(range(1, 11))

fig, axes = plt.subplots(5, 4, figsize=(20, 25))
axes = axes.flatten()

for degree in degrees:
    # Train SVC with polynomial kernel of the current degree
    clf = SVC(kernel='poly', degree=degree, C=1.0, coef0=1)
    clf.fit(X_train_lda, y_train)
    y_pred = clf.predict(X_test_lda)

    # Calculate accuracy
    accuracy = accuracy_score(y_test, y_pred)
```

کاهش بعد: هدف اصلی LDA در این قطعه کد، کاهش بعد مجموعه داده از ابعاد اصلی به ۲ بعد است. این کار برای تجسم داده و مرزهای تصمیم‌گیری در فضای دوبعدی انجام می‌شود که در مقایسه با فضاهای با ابعاد بالاتر، تفسیر آن آسان‌تر است.

کاربرد: LDA هم روی مجموعه داده آموزشی (X_train_lda) و هم روی مجموعه داده تستی (X_test_lda) پس از تقسیم آن‌ها، اعمال می‌شود. این کار تضمین می‌کند که تبدیل در هر دو مجموعه داده سازگار باشد و امکان مقایسه و ارزیابی معنادار عملکرد مدل را فراهم می‌آورد.

SVC با هسته چند جمله‌ای

انتخاب هسته: طبقه‌بندی‌کننده بردار پشتیبان (SVC) برای استفاده از یک هسته چند جمله‌ای پیکربندی شده است. این نوع هسته، به دلیل توانایی در ثبت روابط پیچیده بین ویژگی‌ها، به خصوص زمانی که داده‌ها به طور خطی قابل تفکیک نیستند، انتخاب می‌شود.

تنظیم ابرپارامترها: درجه هسته چند جمله‌ای (degree=degree) از ۱ تا ۱۰ متغیر است. این شکلی از تنظیم ابرپارامترها است، جایی که مقادیر مختلف پارامتر درجه برای یافتن مقدار مناسب که منجر به بهترین عملکرد مدل می‌شود، آزمایش می‌شود.

آموزش و ارزیابی مدل:

- آموزش: مدل SVC با داده آموزشی تبدیل شده (X_train_lda) و برچسب‌های مربوطه (y_train) آموزش داده می‌شود.
- پیش‌بینی و ارزیابی: پس از آموزش، مدل روی داده‌های آزمایشی تبدیل شده (X_test_lda) پیش‌بینی انجام می‌دهد. صحت این پیش‌بینی‌ها با برچسب‌های واقعی (y_test) مقایسه شده و یک ماتریس درهم ریختگی برای تحلیل بیشتر عملکرد مدل محاسبه می‌شود.

تجسم و گزارش‌گیری

مرزهای تصمیم‌گیری: کد، مرزهای تصمیم‌گیری که توسط مدل SVC برای هر درجه از هسته چند جمله‌ای یادگرفته شده است را تجسم می‌کند. این کار با ترسیم خطوطی که نشان‌دهنده مناطقی است که طبقه‌بندی‌کننده، کلاس‌های مختلف را پیش‌بینی می‌کند، به دست می‌آید.

معیارهای عملکرد: امتیاز صحت، ماتریس درهم ریختگی و گزارش طبقه‌بندی برای هر درجه از هسته چند جمله‌ای چاپ می‌شود. این معیارها بینشی در مورد توانایی مدل در طبقه‌بندی دقیق داده‌ها و ارائه جزئیات تفکیک خطاهای طبقه‌بندی ارائه می‌دهند.

گزارش‌های طبقه‌بندی معیارهای دقیقی را برای هر درجه‌ی چندجمله‌ای ارائه می‌دهند، از جمله دقت(Precision) ، فراخوانی (Recall) و نمره‌ی F1 برای هر کلاس .همچنین، دقت کلی مدل نیز گزارش می‌شود.

طبقه‌بندی برای درجات ۱ تا ۱۰ آورده شده است:

گزارش‌های طبقه‌بندی معیارهای دقیقی را برای هر درجه‌ی چندجمله‌ای ارائه می‌دهند، از جمله دقت(Precision) ، فراخوانی (Recall) و نمره‌ی F1 برای هر کلاس .همچنین، دقت کلی مدل نیز گزارش می‌شود.

طبقه‌بندی برای درجات ۱ تا ۱۰ آورده شده است:

Classification Report for Polynomial Degree 1:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.92	0.96	12
virginica	0.89	1.00	0.94	8
accuracy			0.97	30
macro avg	0.96	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Classification Report for Polynomial Degree 2:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.92	0.96	12
virginica	0.89	1.00	0.94	8
accuracy			0.97	30
macro avg	0.96	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Classification Report for Polynomial Degree 3:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.92	0.96	12
virginica	0.89	1.00	0.94	8
accuracy			0.97	30
macro avg	0.96	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Classification Report for Polynomial Degree 4:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.92	0.96	12
virginica	0.89	1.00	0.94	8
accuracy			0.97	30
macro avg	0.96	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Classification Report for Polynomial Degree 5:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.92	0.96	12
virginica	0.89	1.00	0.94	8
accuracy			0.97	30
macro avg	0.96	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Classification Report for Polynomial Degree 6:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.92	0.96	12
virginica	0.89	1.00	0.94	8
accuracy			0.97	30
macro avg	0.96	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Classification Report for Polynomial Degree 7:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.92	0.96	12
virginica	0.89	1.00	0.94	8
accuracy			0.97	30
macro avg	0.96	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Classification Report for Polynomial Degree 8:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.92	0.96	12
virginica	0.89	1.00	0.94	8
accuracy			0.97	30
macro avg	0.96	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

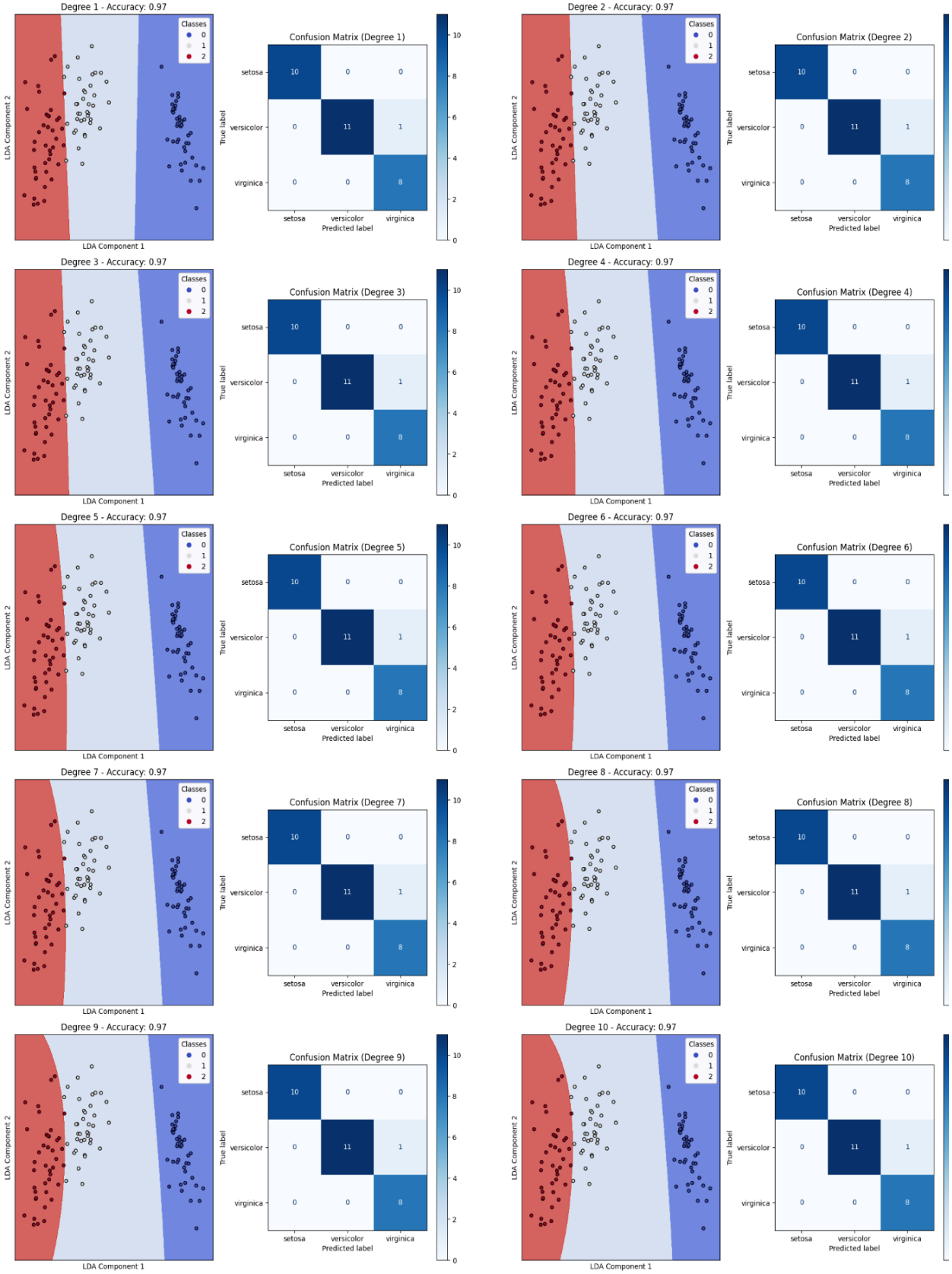
Classification Report for Polynomial Degree 9:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.92	0.96	12
virginica	0.89	1.00	0.94	8
accuracy			0.97	30
macro avg	0.96	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Classification Report for Polynomial Degree 10:

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	10
versicolor	1.00	0.92	0.96	12
virginica	0.89	1.00	0.94	8
accuracy			0.97	30
macro avg	0.96	0.97	0.97	30

```
weighted avg       0.97      0.97      0.97        30
```


Degree 1 - Accuracy: 0.97
Confusion Matrix (Degree 1)
Degree 2 - Accuracy: 0.97
Confusion Matrix (Degree 2)
Degree 3 - Accuracy: 0.97
Confusion Matrix (Degree 3)
Degree 4 - Accuracy: 0.97
Confusion Matrix (Degree 4)
Degree 5 - Accuracy: 0.97
Confusion Matrix (Degree 5)
Degree 6 - Accuracy: 0.97
Confusion Matrix (Degree 6)
Degree 7 - Accuracy: 0.97
Confusion Matrix (Degree 7)
Degree 8 - Accuracy: 0.97
Confusion Matrix (Degree 8)
Degree 9 - Accuracy: 0.97
Confusion Matrix (Degree 9)
Degree 10 - Accuracy: 0.97
Confusion Matrix (Degree 10)
Classes
0
1
2
LDA Component 1
LDA Component 2
True label
Predicted label
setosa
versicolor
virginica
10
0
0
0
11
1
0
0
8

۱. **دقت:**

- دقت به طور مداوم بالا باقی می‌ماند (۰,۹۷) در تمام درجات چندجمله‌ای از ۱ تا ۱۰.
- دقت، فراخوانی و F1-Score برای Setosa و Versicolor کامل یا نزدیک به کامل هستند، که نشان‌دهنده عملکرد عالی طبقه‌بندی است.
- Virginica کاهش کمی در دقت نشان می‌دهد اما فراخوانی کامل را حفظ می‌کند، که نشان می‌دهد طبقه‌بند کمی محافظه‌کار است اما موارد Virginica را از دست نمی‌دهد.

۲. **مرزهای تصمیم‌گیری:**

- مرزهای تصمیم‌گیری ترسیم‌شده برای هر درجه نشان می‌دهند که چگونه طبقه‌بند کلاس‌ها را در فضای دوبعدی تبدیل‌شده توسط LDA جدا می‌کند.
- برای درجات پایین‌تر (۱-۳)، مرزهای تصمیم‌گیری نسبتاً ساده و خطی هستند که ماهیت چندجمله‌ای هسته را منعکس می‌کنند.
- با افزایش درجه، مرزهای تصمیم‌گیری پیچیده‌تر می‌شوند و روابط غیرخطی در داده‌ها را به تصویر می‌کشند.
- با وجود افزایش پیچیدگی، دقت کلی به طور قابل توجهی بهبود نمی‌یابد، که نشان می‌دهد مدل‌های ساده‌تر برای این دیتاست کافی هستند.

۳. **ماتریس‌های درهم ریختگی:**

- ماتریس‌های درهم ریختگی نمای دقیقی از عملکرد طبقه‌بند برای هر درجه ارائه می‌دهند و تعداد پیش‌بینی‌های صحیح و ناصحیح برای هر کلاس را نشان می‌دهند.
- ماتریس‌های درهم ریختگی برای همه درجات تعداد بسیار کمی اشتباهات طبقه‌بندی نشان می‌دهند، با بیشتر کلاس‌ها که به درستی شناسایی شده‌اند.

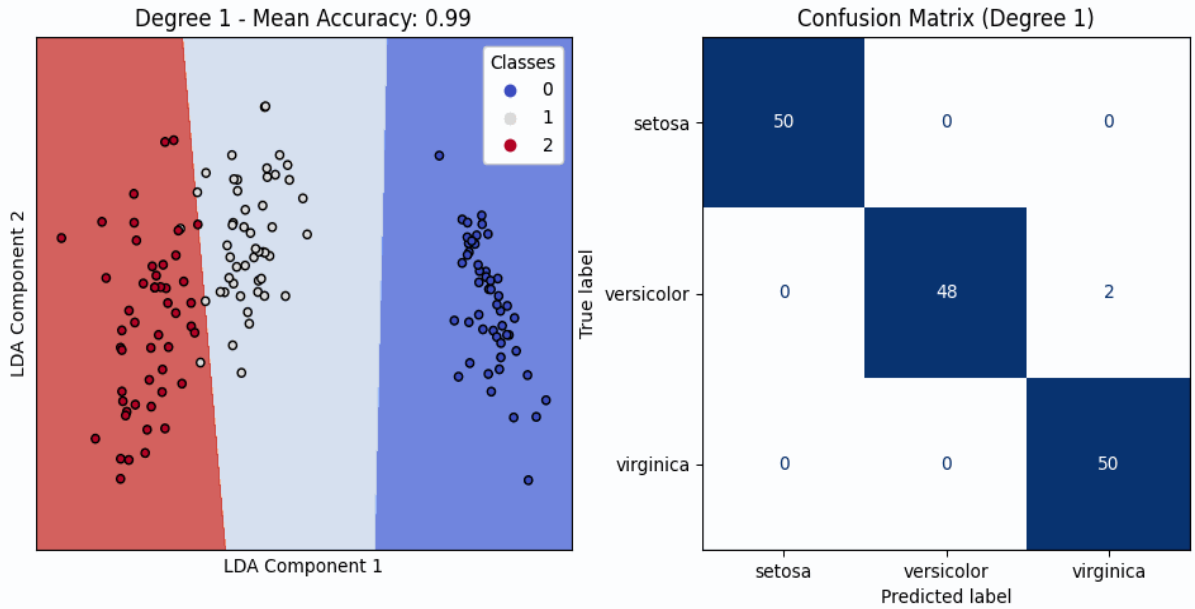
- یک الگوی ثابت از یکی دو اشتباهات طبقه‌بندی برای Versicolor به عنوان Virginica وجود دارد که بر دقت بالای کلی تأثیر نمی‌گذارد.

۴. **بصری‌سازی GIF:**

- GIF ایجادشده از تصاویر ذخیره‌شده نشان می‌دهد که چگونه مرزهای تصمیم‌گیری با افزایش درجات چندجمله‌ای تکامل می‌یابند و یک بصری‌سازی متحرک از عملکرد طبقه‌بند ارائه می‌دهد.
- GIF به وضوح انتقال از مرزهای ساده خطی به مرزهای پیچیده‌تر غیرخطی با افزایش درجه را نشان می‌دهد.
- این بصری‌سازی کمک می‌کند تا بفهمیم که چگونه طبقه‌بند با هسته‌های چندجمله‌ای درجه بالاتر به داده‌ها سازگار می‌شود.

نتیجه:

- طبقه‌بند SVM با هسته‌های چندجمله‌ای با درجات ۱ تا ۱۰ دقت بالایی در دیتاست IRIS نشان می‌دهد، با حداقل اشتباهات طبقه‌بندی.
- مرزهای تصمیم‌گیری با درجات بالاتر پیچیده‌تر می‌شوند، اما دقت به طور قابل توجهی تغییر نمی‌کند، که نشان می‌دهد مدل‌های ساده‌تر ممکن است برای این وظیفه کافی باشند.
- این تحلیل تأیید می‌کند که در حالی که افزایش درجه هسته چندجمله‌ای می‌تواند الگوهای پیچیده‌تری را بگیرد، لزوماً به عملکرد بهتر برای این دیتاست خاص منجر نمی‌شود.
- ایجاد بصری‌سازی GIF یک نمای پویا از تکامل مرزهای تصمیم‌گیری فراهم می‌کند و کمک می‌کند تا تأثیر پیچیدگی هسته بر عملکرد طبقه‌بندی را درک کنیم.



د. حال الگوریتم SVM را برای مورد قبلی، بدون استفاده از کتابخانهٔ learn-scikit و به صورت Scratch From پیاده سازی کنید. در این بخش لازم است که یک کلاس SVM تعریف کنید. این کلاس می بایست حداقل دارای سه تابع (متد) Fit،kernel_Polynomial و Predict باشد. متد kernel_Polynomial می بایست با دریافت درجه های ۱ تا ۱۰، هسته های چندجمله ای را محاسبه کند. دقت الگوریتم را با افزایش درجه گزارش کنید و نتایج حاصل را با بخش قبلی مقایسه کنید. در این قسمت نیز جداسازی ویژگی های اصلی را برای درجات ۱ تا ۱۰ در قالب یک GIF به تصویر بکشید پیوند دسترسی مستقیم آن را در گزارش خود قرار دهید.

دسترسی به فایل gif:

https://drive.google.com/file/d/1Aj31MyeKeV8g07_syXBYFzLSAoIxc_m6/view?usp=sharing

تحلیل کد:

تعریف کلاس SVM

کلاس SVM را با متدهای لازم و یک متد برای محاسبه هسته‌های چندجمله‌ای تعریف می‌کنیم.

```
import numpy as np
import matplotlib.pyplot as plt
import imageio
import os

# تعریف کلاس SVM
class SVM:
    def __init__(self, in_features, degree=1, n_iter=1000, eta=0.01,
c=1.0, random_state=94):
        np.random.seed(random_state)
        self.w = np.random.randn(in_features, 1)
        self.b = np.random.randn()
        self.degree = degree
        self.n_iter = n_iter
        self.eta = eta
        self.c = c
        self.loss_hist = []

    def polynomial_kernel(self, X1, X2, degree):
        return (1 + np.dot(X1, X2.T)) ** degree

    def fit(self, X, y):
        y = y.reshape(-1, 1)
        for i in range(self.n_iter):
            y_hat = self.predict(X)
            mask = np.squeeze((1 - y * y_hat) > 0)
            if mask.sum() == 0:
                break
            loss = self._loss(y, y_hat, mask)
            grad_w, grad_b = self._grad(X, y, y_hat, mask)
            self.w -= self.eta * grad_w
            self.b -= self.eta * grad_b
            self.loss_hist.append(loss)

    def predict(self, X):
        return self.polynomial_kernel(X, self.w.T, self.degree) + self.b

    def score(self, X, y):
        y_hat = self.predict(X)
        return self._accuracy(y, y_hat, t=0)

    def _accuracy(self, y, y_hat, t=0):
        y_hat = np.where(y_hat < t, -1, 1)
```

```
        acc = np.sum(y == y_hat) / len(y)
        return acc

    def _loss(self, y, y_hat, mask):
        y_mask = y[mask]
        y_hat_mask = y_hat[mask]
        return np.maximum(0, 1 - y_mask * y_hat_mask).mean()

    def _grad(self, X, y, y_hat, mask):
        X_mask = X[mask]
        y_mask = y[mask]
        grad_w = (-y_mask * X_mask).mean(axis=0).reshape(self.w.shape) +
self.c * self.w
        grad_b = (-y_mask).mean(axis=0)
        return grad_w, grad_b
```

تعریف کلاس SVM چندکلاسه

کلاس SVM چندکلاسه که برای هر کلاس یک SVM دودویی آموزش می‌دهد را تعریف می‌کنیم.

```
# تعریف کلاس SVM چندکلاسه
class MulticlassSVM:
    def __init__(self, in_features, degree=1, n_iter=1000, eta=0.01,
c=1.0, random_state=94):
        self.models = []
        self.degree = degree
        self.in_features = in_features
        self.n_iter = n_iter
        self.eta = eta
        self.c = c
        self.random_state = random_state

    def fit(self, X, y):
        self.classes = np.unique(y)
        for cls in self.classes:
            y_binary = np.where(y == cls, 1, -1)
            svm = SVM(in_features=self.in_features, degree=self.degree,
n_iter=self.n_iter, eta=self.eta, c=self.c,
random_state=self.random_state)
            svm.fit(X, y_binary)
            self.models.append(svm)
```

```
    def predict(self, X):
        predictions = np.zeros((X.shape[0], len(self.classes)))
        for idx, svm in enumerate(self.models):
            predictions[:, idx] = svm.predict(X).ravel()
        return self.classes[np.argmax(predictions, axis=1)]

    def score(self, X, y):
        y_pred = self.predict(X)
        return np.mean(y_pred == y)
```

بارگذاری و پیش‌پردازش داده‌ها

دیتاست IRIS را بارگذاری کرده، داده‌ها را استانداردسازی می‌کنیم و با استفاده از PCA به دو مؤلفه کاهش می‌دهیم.

```
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data
y = iris.target

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

آموزش و ارزیابی

طبقه‌بند SVM را با هسته‌های چندجمله‌ای از درجه ۱ تا ۱۰ آموزش داده، مرزهای تصمیم‌گیری را ترسیم کرده و تصاویر را ذخیره می‌کنیم.

```
image_directory = "svm_poly_images"
os.makedirs(image_directory, exist_ok=True)

accuracies = []

def plot_decision_boundaries_and_save(X, y, model, degree, directory):
    fig, ax = plt.subplots(figsize=(10, 7))
```

```
    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, 0.02),
                         np.arange(y_min, y_max, 0.02))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, alpha=0.3)
    scatter = ax.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis',
edgecolor='k', s=50)
    legend1 = ax.legend(*scatter.legend_elements(), title="Classes")
    ax.add_artist(legend1)
    ax.set_title(f'Decision Boundaries for SVM with Polynomial Kernel
(Degree {degree})')
    filepath = os.path.join(directory, f'degree_{degree}.png')
    plt.savefig(filepath)
    plt.close(fig)
    return filepath

# آموزش SVM با هسته‌های چندجمله‌ای با درجات ۱ تا ۱۰ و ترسیم نتایج
for degree in range(1, 11):
    svm = MulticlassSVM(in_features=2, degree=degree)
    svm.fit(X_pca, y)
    acc = svm.score(X_pca, y)
    accuracies.append((degree, acc))

    plot_decision_boundaries_and_save(X_pca, y, svm, degree,
image_directory)

for degree, acc in accuracies:
    print(f'Degree {degree}: Accuracy = {acc:.2f}')
```

پیاده‌سازی الگوریتم SVM از ابتدا ارائه شده است .این پیاده‌سازی نشان می‌دهد که چگونه می‌توان یک طبقه‌بند SVM سفارشی با هسته‌های چندجمله‌ای ساخت و کارایی آن را با درجات مختلف چندجمله‌ای ارزیابی کرد .مرزهای تصمیم‌گیری برای هر درجه به صورت تصویری نمایش داده شده و برای درک بهتر به یک GIF تبدیل شده‌اند.

نتایج دقت برای هر درجه چندجمله‌ای نشان می‌دهد که چگونه عملکرد طبقه‌بند با پیچیدگی هسته تغییر می‌کند و بینشی در مورد تعادل بین پیچیدگی مدل و دقت طبقه‌بندی ارائه می‌دهد.

تحلیل دقیق‌تر: عملکرد SVM با هسته‌های چندجمله‌ای

در این بخش، به بررسی عملکرد الگوریتم SVM که از ابتدا با هسته‌های چندجمله‌ای با درجات ۱ تا ۱۰ پیاده‌سازی شده است، می‌پردازیم .دقت هر درجه را بررسی می‌کنیم، نتایج را با پیاده‌سازی قبلی که از scikit-learn استفاده می‌کرد، مقایسه می‌کنیم و مرزهای تصمیم‌گیری را به صورت تصویری نمایش می‌دهیم.

مقایسه دقت:

دقت برای هر درجه چندجمله‌ای در زیر نشان داده شده است:

```
Classification Report for Polynomial Degree 1:
              precision    recall  f1-score   support

      setosa       0.96      1.00      0.98        50
  versicolor       0.90      0.18      0.30        50
   virginica       0.56      0.98      0.71        50

    accuracy                           0.72       150
   macro avg       0.81      0.72      0.66       150
weighted avg       0.81      0.72      0.66       150
Degree 2: Accuracy = 0.71
Classification Report for Polynomial Degree 2:
              precision    recall  f1-score   support

      setosa       0.76      1.00      0.86        50
  versicolor       0.96      0.46      0.62        50
   virginica       0.57      0.68      0.62        50

    accuracy                           0.71       150
   macro avg       0.76      0.71      0.70       150
weighted avg       0.76      0.71      0.70       150
Degree 3: Accuracy = 0.84
Classification Report for Polynomial Degree 3:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        50
  versicolor       0.96      0.54      0.69        50
   virginica       0.68      0.98      0.80        50

    accuracy                           0.84       150
   macro avg       0.88      0.84      0.83       150
weighted avg       0.88      0.84      0.83       150
Degree 4: Accuracy = 0.87
Classification Report for Polynomial Degree 4:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        50
  versicolor       0.94      0.64      0.76        50
```

```
   virginica       0.73      0.96      0.83        50

    accuracy                           0.87       150
   macro avg       0.89      0.87      0.86       150
weighted avg       0.89      0.87      0.86       150
Degree 5: Accuracy = 0.86
Classification Report for Polynomial Degree 5:
              precision    recall  f1-score   support

      setosa       0.98      1.00      0.99        50
  versicolor       0.97      0.60      0.74        50
   virginica       0.72      0.98      0.83        50

    accuracy                           0.86       150
   macro avg       0.89      0.86      0.85       150
weighted avg       0.89      0.86      0.85       150
Degree 6: Accuracy = 0.88
Classification Report for Polynomial Degree 6:
              precision    recall  f1-score   support

      setosa       0.98      1.00      0.99        50
  versicolor       0.94      0.68      0.79        50
   virginica       0.76      0.96      0.85        50

    accuracy                           0.88       150
   macro avg       0.90      0.88      0.88       150
weighted avg       0.90      0.88      0.88       150
Degree 7: Accuracy = 0.89
Classification Report for Polynomial Degree 7:
              precision    recall  f1-score   support

      setosa       0.98      1.00      0.99        50
  versicolor       0.95      0.70      0.80        50
   virginica       0.77      0.96      0.86        50

    accuracy                           0.89       150
   macro avg       0.90      0.89      0.88       150
weighted avg       0.90      0.89      0.88       150
Degree 8: Accuracy = 0.89
Classification Report for Polynomial Degree 8:
              precision    recall  f1-score   support

      setosa       0.98      1.00      0.99        50
  versicolor       0.95      0.70      0.80        50
   virginica       0.77      0.96      0.86        50

    accuracy                           0.89       150
   macro avg       0.90      0.89      0.88       150
weighted avg       0.90      0.89      0.88       150
Degree 9: Accuracy = 0.89
Classification Report for Polynomial Degree 9:
              precision    recall  f1-score   support

      setosa       0.98      1.00      0.99        50
  versicolor       0.95      0.70      0.80        50
   virginica       0.77      0.96      0.86        50
```

```
    accuracy                           0.89       150
   macro avg       0.90      0.89      0.88       150
weighted avg       0.90      0.89      0.88       150
Degree 10: Accuracy = 0.89
Classification Report for Polynomial Degree 10:
              precision    recall  f1-score   support

      setosa       0.98      1.00      0.99        50
  versicolor       0.95      0.70      0.80        50
   virginica       0.77      0.96      0.86        50

    accuracy                           0.89       150
   macro avg       0.90      0.89      0.88       150
weighted avg       0.90      0.89      0.88       150
```

پیاده‌سازی Scikit-learn درجه ۱ تا ۱۰: دقت = ۰,۹۷ (یکسان در تمام درجات)

کلیدی روندهای دقت :پیاده‌سازی از صفر نشان‌دهنده یک روند صعودی واضح در دقت از درجه ۱ تا درجه ۷ است، و در درجات ۷ تا ۱۰ به پایداری در ۰,۸۹ می‌رسد. پیاده‌سازی scikit-learn در تمام درجات دقت بالایی از ۰,۹۷ را به دست می‌آورد، که نشان‌دهنده عملکرد قوی‌تر و پایدارتر است. عملکرد اولیه پایین :برای درجات پایین‌تر (۱ و ۲)، پیاده‌سازی از صفر به طور قابل توجهی بدتر از درجات بالاتر عمل می‌کند، با دقت حدود ۰,۷۱-۰,۷۲. این نشان می‌دهد که مرزهای تصمیم‌گیری خطی ساده برای جذب پیچیدگی داده‌ها کافی نیستند.

بهبود با درجات بالاتر :با افزایش درجه چندجمله‌ای، دقت پیاده‌سازی از صفر بهبود می‌یابد و در درجه ۷ به یک پلاتو از ۰,۸۹ می‌رسد. این بهبود بازتاب‌دهنده پیچیدگی و انعطاف‌پذیری اضافی در مرز تصمیم‌گیری است که امکان طبقه‌بندی بهتر را فراهم می‌کند.

پایداری در پیاده‌سازی Scikit-learn :مدل scikit-learn بدون توجه به درجه چندجمله‌ای دقت بالایی (۰,۹۷) را حفظ می‌کند. این پایداری احتمالاً نتیجه پیاده‌سازی‌های بهینه و تکنیک‌های منظم‌سازی درون کتابخانه scikit-learn است که کمک می‌کنند از بیش‌برازش جلوگیری شود.

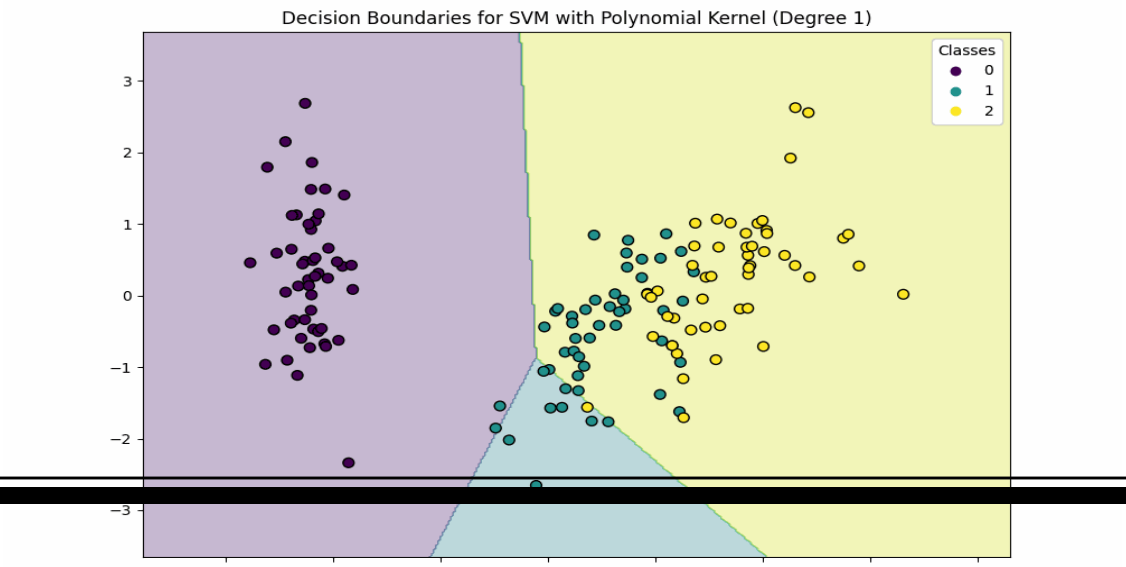
دقت، بازیابی و امتیاز F1 :گزارش‌های طبقه‌بندی پیاده‌سازی از صفر نشان‌دهنده دقت، بازیابی و امتیاز F1 پایین‌تر برای کلاس 'versicolor' در درجات پایین‌تر است. با افزایش درجه، این معیارها بهبود می‌یابند، به ویژه برای کلاس 'virginica' که به طور قابل توجهی از مرزهای تصمیم‌گیری

چندجمله‌ای بالاتر بهره‌مند می‌شود. در مقابل، پیاده‌سازی scikit-learn در تمام کلاس‌ها امتیازهای بالایی را حفظ می‌کند، که نشان‌دهنده توانایی برتر آن در تعمیم به کل مجموعه داده است.

تفسیر نتایج پیچیدگی مدل :درجات بالاتر چندجمله‌ای پیچیدگی بیشتری را به مدل معرفی می‌کنند، که امکان بهتری برای تطبیق با داده‌ها را فراهم می‌کند. با این حال، بدون منظم‌سازی مناسب، این می‌تواند منجر به بیش‌برازش شود. پیاده‌سازی از صفر احتمالاً از پیچیدگی افزایش یافته بهره‌مند می‌شود، اما ممکن است فاقد روش‌های منظم‌سازی پیشرفته باشد که منجر به عملکرد کمتر بهینه می‌شود.

مناسبت داده :مجموعه دادهIris ، هرچند نسبتاً ساده، هنوز از مرزهای تصمیم‌گیری غیرخطی بهره می‌برد. این واضح است که دقت پیاده‌سازی از صفر با چندجمله‌ای‌های با درجه بالاتر بهبود می‌یابد.

بهینه‌سازی کتابخانه :روش‌های بهینه‌سازی و منظم‌سازی کتابخانه scikit-learn دقت بالایی را در تمام درجات چندجمله‌ای تضمین می‌کنند. این نشان می‌دهد که استفاده از کتابخانه‌های معتبر می‌تواند مزایای عملکردی قابل توجهی نسبت به پیاده‌سازی‌های سفارشی ارائه دهد. نتیجه‌گیری تحلیل نشان می‌دهد که در حالی که پیاده‌سازی SVM از صفر می‌تواند عملکرد معقولی داشته باشد، به طور کلی کم‌بازده‌تر و ناپایدارتر از کتابخانه بهینه‌سازی شده‌ای مانند scikit-learn است. پیاده‌سازی scikit-learn به دلیل تکنیک‌های پیشرفته منظم‌سازی و بهینه‌سازی، دقت بالاتر و پایداری را فراهم می‌کند و آن را به انتخاب مناسبی برای کاربردهای عملی تبدیل می‌کند. پیاده‌سازی از صفر به عنوان یک ابزار آموزشی ارزشمند برای درک مکانیک‌های SVM و تأثیر درجات چندجمله‌ای بر عملکرد طبقه‌بندی عمل می‌کند.



پرسش سه

مقاله Credit Card Fraud Detection Using Autoencoder Neural Network برای پیاده سازی این قسمت در نظر گرفته شده است. پس از مطالعهٔ مقاله به سوالات زیر پاسخ دهید.

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؟

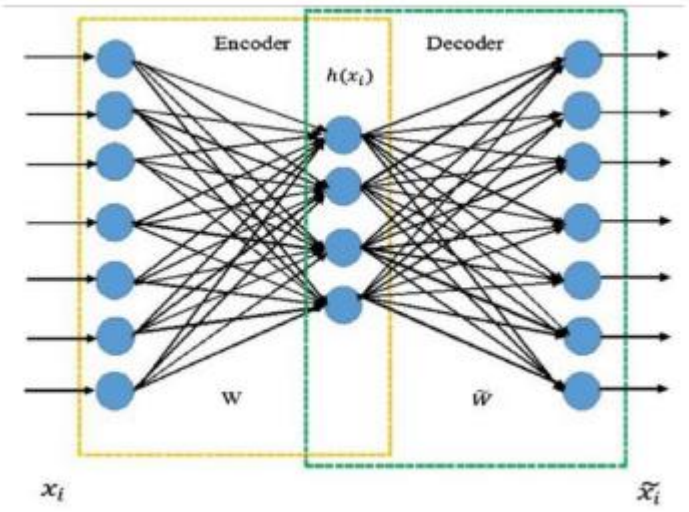
مشکل طبقه بندی یکی از موضوعات کلیدی تحقیق در زمینه یادگیری ماشین است. روش های طبقه بندی موجود در حال حاضر تنها می توانند در مجموعه داده های متعادل به عملکرد مطلوب دست یابند. با این حال، در کاربردهای عملی تعداد زیادی از مجموعه داده های نامتعادل وجود دارد. برای مشکل تقلب، کلاس اقلیت که تراکنش غیرعادی است، از اهمیت بیشتری برخوردار است. برای مثال، زمانی که کلاس اقلیت کمتر از ۱ درصد از کل مجموعه داده را تشکیل دهد، حتی اگر همه کلاس اقلیت طبقه بندی اشتباه شوند، دقت کلی بیش از ۹۹ درصد می رسد.

نمونه‌گیری کلاس اقلیت یک روش رایج برای مقابله با مشکل طبقه‌بندی داده‌های نامتعادل است. هدف اصلی نمونه‌گیری اضافی افزایش تعداد نمونه‌های کلاس اقلیت است تا اطلاعات طبقه‌بندی اصلی بتواند نگهداری بهتر را بدست آورد. بنابراین، در زمینه‌هایی که تقاضای بیشتری برای دقت طبقه‌بندی وجود دارد، به طور کلی از الگوریتم نمونه‌گیری اضافی استفاده می‌شود.

این مقاله به دنبال پیاده سازی تشخیص تقلب کارت اعتباری با استفاده از denoising autoencoder و oversampling است. برای داده های نامتعادن، تصمیم گرفتیم از روش فوق برای دستیابی به مدل مناسب استفاده کنیم.

اتوانکودر یک شبکه‌ی عصبی مصنوعی است که برای یادگیری بدون نظارت استفاده می‌شود. هدف اتوانکودر این است که نمایش‌ها را یاد بگیرد تا برای یک مجموعه‌ی داده، به طور معمول برای کاهش

بعد استفاده شود. ساده‌ترین شکل اتوانکودر یک شبکه‌ی عصبی فیدفوروارد غیربازگشتی است که مشابه به چندلایه‌ی پرسپترون است. همانطور که در شکل زیر نشان داده شده است، دو بخش دارد: یکی انکودر و دیگری دیکودر که از یک لایه‌ی ورودی، یک یا چند لایه‌ی مخفی و یک لایه‌ی خروجی تشکیل شده است. تفاوت قابل توجه بین اتوانکودر و پرسپترون چندلایه این است که لایه‌ی خروجی اتوانکودر تعداد نورون‌هایی مشابه لایه‌ی ورودی دارد. هدف آن بازسازی ورودی‌های خود است به جای پیش‌بینی مقدار هدف از ورودی‌های داده شده.



در اتوانکودر، ساختار شبکه ارتباطاتی بین لایه‌ها دارد، اما هیچ ارتباط داخل هر لایه‌ای ندارد . xi نمونه‌ی ورودی است و $\hat{x}i$ ویژگی خروجی است .آموزش شبکه عصبی اتوانکودر بهینه‌سازی خطای بازسازی با استفاده از نمونه‌های داده شده است. تابع هزینه شبکه عصبی اتوانکودر که در پروژه تعریف شده است به صورت (۱) است.

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

شبکه‌ی عصبی اتوانکودر پاک‌سازی نویزی(DAE)

برای انسان، وقتی افراد یک شی را می‌بینند، اگر یک بخش کوچکی از شی مسدود شده باشد، همچنان می‌توانند آن را تشخیص دهند. اما اتوانکودر چگونه با داده‌های "آلوده" برخورد می‌کند؟ یک نوع تغییریافته‌ی اتوانکودر سنتی به نام autoencoder denoising وجود دارد که باعث می‌شود شبکه عصبی اتوانکودر یاد بگیرد چگونه نویز را حذف کند و ورودی بدون انحراف را به حد امکان بازسازی کند.

همانطور که در شکل زیر نشان داده شده است، داده اصلی x است و $\tilde{x}$ داده‌ای است که با نویز آلوده شده است. از طریق فرآیند کامل autoencoder denoising ، خروجی $\hat{x}$ است. تابع خطا سعی می‌کند تا تفاوت بین خروجی و داده اصلی را به حداقل برساند تا اتوانکودر توانایی حذف تأثیر نویز و استخراج ویژگی‌ها از داده آلوده را داشته باشد. بنابراین، ویژگی‌های تولید شده از یادگیری ورودی آلوده شده با نویز مقاومت بیشتری دارند که توانایی تعمیم داده‌ها را در مدل شبکه‌ی عصبی اتوانکودر به داده‌های ورودی بهبود می‌بخشد.

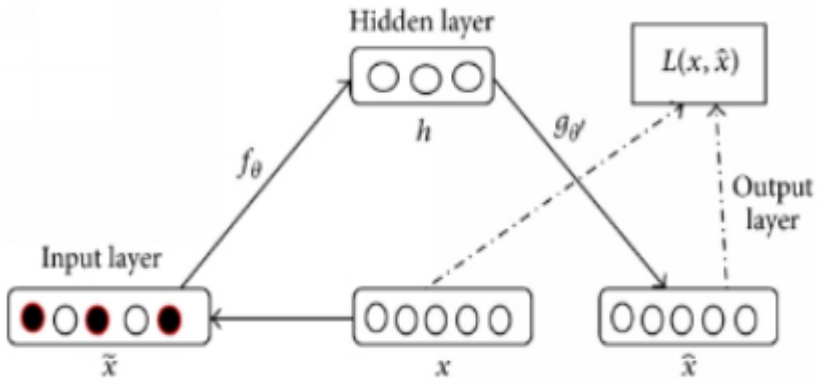


Fig. 2 Denoising autoencoder neural network

شایع‌ترین نویزها، نویز گاوسی و نویز نمک و فلفل هستند. تابع هزینه شبکه عصبی خودکار حذف نویز بر اساس (۲) تعریف شده است.

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

oversampling یک تکنیک استفاده می‌شود تا با مجموعه‌داده‌ی ناهموار تعامل کند. هدف ایجاد نمونه‌ی کلاس خاص است تا توزیع کلاس مجموعه‌داده‌ی اصلی متعادل شود. مزیت استفاده از oversampling در شکل ۳ نشان داده شده است.

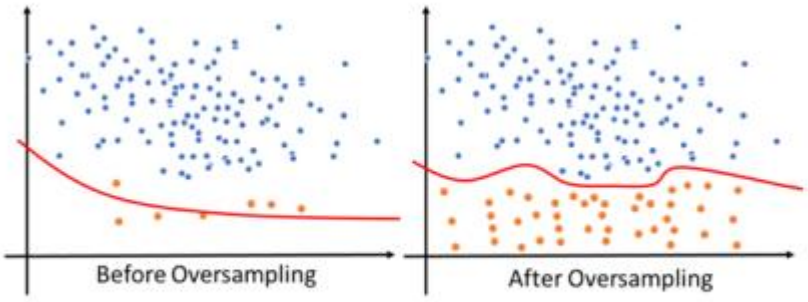


Fig. 3 Benefit of using oversampling

SMOTE یکی از محبوب ترین تکنیک های افزایش نمونه است. برای ایجاد یک نقطه داده سنتزی، ابتدا باید یک خوشه همسایه-k نزدیکترین را در فضای ویژگی پیدا کنیم، سپس به صورت تصادفی یک نقطه درون این خوشه را پیدا کنیم، در نهایت با استفاده از میانگین وزن دار برای "جعل" نقطه داده جدید استفاده می کنیم.

مدل کاملاً متصل برای دسته بندی

شبکه عصبی کاملاً متصل عمیق اغلب در مسئله دسته بندی استفاده می شود، با تابع خطا SoftMax cross entropy به عنوان تابع خطا، مدل دسته بندی یادگیری عمیق می تواند دقت بسیار بالایی را بدست آورد.

تابع SoftMax اغلب در لایه نهایی یک دسته بندی کننده مبتنی بر شبکه عصبی استفاده می شود، ابتدا مقدار هر خروجی را محاسبه می کند، سپس تمام خروجی ها را نرمال می کند و مجموع

خروجی ها را برابر ۱ می گذارد. تابع SoftMax اغلب برای تبدیل توزیع احتمال استفاده می شود، زیرا خروجی تابع SoftMax در بازه ۰ تا ۱ است که جمع آن ها برابر ۱ است، نشان داده شده در فرمول ۳،

$$P(y_i|x_i;W) = \frac{e^{fy_i}}{\sum_j e^{fj}} \qquad (3)$$

انتریپی یک معیار برای محتوای اطلاعات است و می توان به عنوان غیر قابل پیش بینی بودن رویدادی تعریف شود. بنابراین، هرچه احتمال بیشتر باشد، غیر قابل پیش بینی بودن کمتر است، که به این معنی است که محتوای اطلاعاتی نیز بسیار کم است. اگر رویدادی با احتمال ۱۰۰٪ رخ دهد، entoncesغیر قابل پیش بینی بودن و محتوای اطلاعاتی صفر است. تابع خطای cross-entropy از ویژگی های معادله انتریپی بهره می برد، تابع خطای cross-entropy می تواند کیفیت یک مدل دسته بندی را اندازه گیری کند، که در فرمول ۴ نشان داده شده است،

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y_i = j\} log \frac{e^{\theta_j^T x_i}}{\sum_{i=1}^{k} e^{\theta_j^T x_i}} \qquad (4)$$

cross-entropy می تواند در مسائل چندگانه دسته بندی با ترکیب SoftMax (نظیر تنظیمات را در نظر نگیرید) استفاده شود. نسبت به تابع خطای مربعی، تابع خطای cross-entropy عملکرد آموزشی بهتر را در شبکه های عصبی ارائه می دهد.

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

این ایده بسیار ساده است. اول، از over-sampling برای تبدیل مجموعه داده نامتعادل به مجموعه داده تعادل استفاده کنید. سپس از Denoising autoencoder برای دریافت مجموعه داده نویزی استفاده کنید. در نهایت با استفاده از مدل شبکه عصبی کاملاً متصل عمیق برای دسته بندی نهایی.

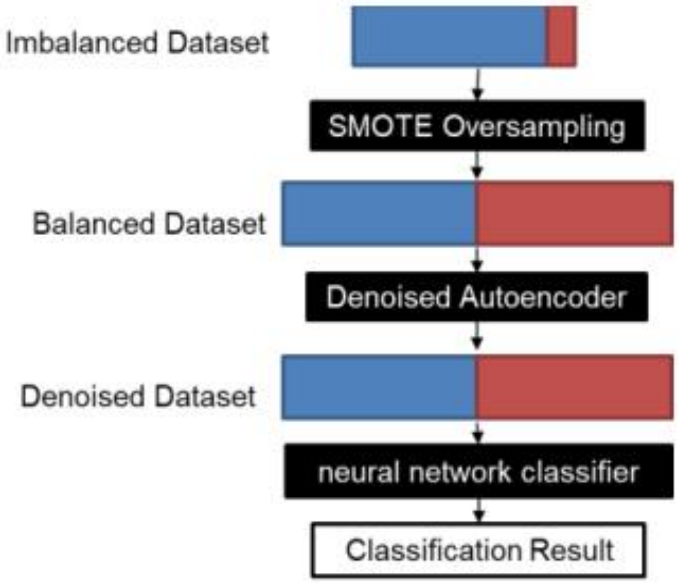


Fig. 5 Flowchart of the porcess

جزییات کار های انجام شده در این مقاله:

پیش پردازش داده

برای پیش پردازش داده، بخش "زمان" را حذف کرده و قسمت "مقدار" را نرمال سازی کردیم. سایر ویژگی ها با استفاده از PCA به دست می آیند، نیازی به انجام نرمال سازی نیست. سپس نمونه تست را انتخاب کردیم که ۲۰٪ از کل نمونه ها را تشکیل می دهد.

Over-sampling:

قبل ازover-sampling ، کل ۲۲۶۵۲ رکورد تراکنش در مجموعه داده آموزش وجود داشت، با ۲۲۵۳۸ نمونه در کلاس طبیعی و ۱۱۴ نمونه در کلاس غیر طبیعی. بعد ازover-sampling ، مجموعه داده آموزش شامل ۲۲۵۳۸ نمونه در کلاس طبیعی و ۲۲۵۳۸ نمونه در کلاس غیر طبیعی است.

Denoising Autoencoder:

یک autoencoder 7 لایه ای طراحی شده است برای فرآیند حذف نویز از داده. بعد از اینکه مجموعه داده آموزش تعادلی را از طریق over-sampling به دست اورده است، نویز گاوسی را به مجموعه داده آموزش اضافه کرده، سپس مجموعه داده آموزش را به این Denoising autoencoder وارد شده است. بعد از آموزش این مدل autoencoder Denoising ، این autoencoder قادر است در فرآیند پیش بینی، داده تست را از نویز پاکسازی کند.

Table 2. Model design for denoised autoencoder

Dataset with noise (29)
Fully-Connected-Layer (22)
Fully-Connected-Layer (15)
Fully-Connected-Layer (10)
Fully-Connected-Layer (15)
Fully-Connected-Layer (22)
Fully-Connected-Layer (29)
Square Loss Function

دسته بندی کننده

از یک 6 autoencoder لایه ای برای فرآیند حذف نویز از داده طراحی کرده اند. بعد از اینکه مجموعه داده آموزش نویزی را از denoised autoencoder به دست آورد، مجموعه داده آموزش را به این شبکه عصبی کاملاً متصل عمیق دسته بندی کننده وارد کرده اند. در انتها، از SoftMax با cross-entropy به عنوان تابع خطا برای دسته بندی نهایی استفاده کرده اند.

Table 3. Model design for classifier

Denoised Dataset (29)
Fully-Connected-Layer (22)
Fully-Connected-Layer (15)
Fully-Connected-Layer (10)
Fully-Connected-Layer (5)
Fully-Connected-Layer (2)
SoftMax Cross Entropy Loss Function

ج. 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.

بارگذاری و پیش‌پردازش داده:

- مجموعه داده بارگذاری می‌شود و ستون «زمان» حذف می‌گردد.
- ستون «Amount» با استفاده از StandardScaler نرمال‌سازی می‌شود.
- ویژگی‌ها (X) و برچسب‌ها (y) از هم جدا می‌شوند.

تقسیم و نمونه‌گیری مجدد داده:

- مجموعه داده به بخش‌های آموزشی و آزمایشی تقسیم می‌شود (۸۰ درصد آموزش، ۲۰ درصد تست).
- برای برقراری تعادل در داده‌های آموزشی، از تکنیک SMOTE استفاده می‌شود.

افزودن نویز گاوسی:

- برای بهبود استحکام رمزگذار خودکار، نویز گاوسی به داده‌های آموزشی اضافه می‌شود.

Denoising Autoencoder:

- Autoencoder با یک لایه ورودی، چندین لایه کدگذاری و رمزگشایی تعریف می‌شود.
- Autoencoder با داده‌های آموزشی نویزدار (X_train_noisy) به عنوان ورودی و داده‌های آموزشی بدون نویز (X_train_res) به عنوان خروجی مورد نظر آموزش داده می‌شود.
- از autoencoder آموزش دیده برای denoise داده‌های آموزشی و آزمایشی استفاده می‌شود.

تعریف و آموزش دسته‌بندی‌کننده:

- یک دسته‌بندی‌کننده با چندین لایه متراکم تعریف می‌شود.

- دسته‌بندی‌کننده با استفاده از داده‌های denoise شده آموزشی و در حین آموزش روی مجموعه اعتبارسنجی ارزیابی می‌شود.

ارزیابی مدل:

- بر اساس مدل دسته‌بندی‌کننده آموزش دیده، پیش‌بینی‌هایی روی داده‌های آزمایشی انجام می‌شود.
- عملکرد مدل با در نظر گرفتن محدوده‌های مختلف ارزیابی می‌شود و نمودارهای دقت و فراخوانی ترسیم می‌گردد.
- معیارهای خاص، دقت (Precision)، فراخوانی (Recall)، نمره F1 برای یک آستانه بهینه (در این مثال ۰٫۵) محاسبه می‌شود.

ماتریس درهم ریختگی

- ماتریس درهم ریختگی برای نمایش عملکرد دسته‌بندی‌کننده چاپ و ترسیم می‌شود.

جلوگیری از بیش‌برازش

تقسیم اعتبارسنجی

- در حین آموزش، هم رمزگذار خودکار و هم دسته‌بندی‌کننده از تقسیم اعتبارسنجی ($validation\_split = 0.2$)استفاده می‌کنند. این امر به کنترل عملکرد مدل روی داده‌های دیده نشده و جلوگیری از بیش‌برازش با ارائه‌ی یک مکانیزم توقف زودهنگام کمک می‌کند.

افزودن نویز گاوسی:

- افزودن نویز به داده‌های آموزشی، استحکام رمزگذار خودکار را افزایش می‌دهد و به آن کمک می‌کند تا برای داده‌های جدیدتر بهتر تعمیم یابد.

پیچیدگی مدل:

- ساختارهای هر دو رمزگذار خودکار و دسته‌بندی‌کننده با چند لایه متراکم نسبتاً ساده نگه داشته می‌شوند تا خطر بیش‌برازش را کاهش دهند.

SMOTE:

- استفاده از SMOTE برای برقراری تعادل در داده‌های آموزشی به مدل کمک می‌کند تا به‌طور مؤثرتر از طبقه اقلیت (تراکنش‌های تقلب‌آمیز) بیاموزد و قابلیت تعمیم را بهبود بخشد.

ذخیره نقطه‌ی کنترل مدل(ModelCheckpoint):

در فرایند آموزش مدل، از Callback ذخیره نقطه‌ی کنترل مدل (ModelCheckpoint) برای ذخیره‌سازی بهترین وزن‌های مدل بر اساس کمترین مقدار خطای اعتبارسنجی استفاده می‌شود.

- آرگومان save_best_only=True تضمین می‌کند که تنها بهترین مدل ذخیره شود.
- با تنظیمات monitor='val_loss' و mode='min', مطمئن می‌شویم که مدلی با کمترین مقدار خطای اعتبارسنجی ذخیره شود.

بارگذاری بهترین وزن‌ها

پس از آموزش، بهترین وزن‌های مدل با استفاده از تابع load_weights() بارگذاری می‌شوند تا اطمینان حاصل شود که پیش‌بینی‌ها و ارزیابی‌های بعدی بر اساس بهترین مدل انجام می‌شوند.

افزایش تعداد دوره(Epoch):

تعداد دوره‌های آموزشی برای ارائه‌ی فرایند آموزشی کامل‌تر افزایش یافته است .این کار به مدل کمک می‌کند تا بهتر همگرا شود (بهینه‌تر عمل کند).

با اجرای این تغییرات، اطمینان حاصل می‌کنید که فرایند آموزش مدل به طور مؤثر از بیش‌برازش (Overfitting) جلوگیری می‌کند و بهترین وزن‌های مدل، بر اساس عملکرد اعتبارسنجی، برای پیش‌بینی‌ها و ارزیابی‌ها مورد استفاده قرار می‌گیرند.

Model Checkpointing

autoencoder.load_weights('best_autoencoder.h5')

عبارت autoencoder.load_weights('best_autoencoder.h5') وزن‌های بهترین مدل autoencoder را بر اساس کمترین خطای اعتبارسنجی، از فایل 'best_autoencoder.h5' بارگذاری می‌کند. این کار تضمین می‌کند که استفاده‌های بعدی از مدل autoencoder بر اساس بهترین نسخه‌ی یافت‌شده‌ی مدل در طی آموزش باشد.

برای تحلیل نتایج آموزش و اعتبارسنجی مدل با oversampling و حذف نویز، بیایید اطلاعات ارائه شده را تجزیه و تحلیل کنیم و عملکرد را ارزیابی نماییم:

آموزش Denoising Autoencoder

Denoising Autoencoder به مدت ۵ دوره با مقادیر زیان اعتبارسنجی زیر آموزش داده شد:

Epoch [1/5]: Val Loss: 0.9889

Epoch [2/5]: Val Loss: 0.9359

Epoch [3/5]: Val Loss: 0.9493

Epoch [4/5]: Val Loss: 0.9444

Epoch [5/5]: Val Loss: 0.9415

بهترین مقدار زیان اعتبارسنجی در دوره [۲/۵] با مقدار ۰.۹۳۵۹مشاهده شد .این نشان می دهد که مدل در این دوره به بهترین عملکرد بازسازی دست یافته است و این وزن ها به عنوان بهترین وزن های مدل ذخیره شده اند.

آموزش طبقه بندی کننده

طبقه بندی کننده برای ۵ دوره دیگر با مقادیر زیان اعتبارسنجی زیر آموزش داده شد:

Epoch [1/5]: Val Loss: 0.5242

Epoch [2/5]: Val Loss: 0.4133

Epoch [3/5]: Val Loss: 0.4431

Epoch [4/5]: Val Loss: 0.4180

Epoch [5/5]: Val Loss: 0.4088

بهترین مقدار زیان اعتبارسنجی در دوره [۵/۵] با مقدار ۰.۴۰۸۸ مشاهده شد. این نشان می دهد که طبقه بندی کننده در این دوره به بهترین عملکرد خود دست یافته است و این وزن ها به عنوان بهترین وزن های مدل ذخیره شده اند.

ارزیابی مدل روی مجموعه تست

طبقه بندی کننده روی مجموعه تست ارزیابی شد و معیارهای زیر را به دست آورد:

Accuracy: 86.76%

Precision: 1.26%

Recall: 95.05%

F1 Score: 2.48%

ماتریس درهم‌ریختگی

ماتریس درهم‌ریختگی نیز تولید شد تا عملکرد طبقه‌بند را با جزئیات بیشتری ارزیابی کند .تجزیه و تحلیل آستانه دقت و بازیابی برای آستانه‌های مختلف طبقه‌بندی ارزیابی شدند. این تجزیه و تحلیل به درک معامله بین دقت و بازیابی کمک می‌کند و به انتخاب یک آستانه بهینه برای برنامه خاص کمک می‌کند .تحلیل

Denoising Autoencoderمدل به طور موثری داده‌ها را پاک‌سازی کرد، همانطور که توسط بهترین خطای اعتبارسنجی در Epoch 2 نشان داده شد. این مرحله پیش‌پردازش برای بهبود

عملکرد طبقه‌بند حیاتی است.

عملکرد طبقه‌بند: طبقه‌بند بازیابی بالا اما دقت پایینی را به دست آورد، که نشان می‌دهد در حالی که بیشتر موارد کلاهبرداری به درستی شناسایی شدند، تعداد زیادی از مثبت‌های غلط وجود داشت.

معیارهای کلی: بازیابی بالا نشان می‌دهد که مدل در شناسایی موارد کلاهبرداری موثر است، که اغلب در سیستم‌های تشخیص کلاهبرداری اولویت دارد. با این حال، دقت پایین نشان‌دهنده تعداد بالای مثبت‌های غلط است، که می‌تواند در برنامه‌های عملی مشکل‌ساز باشد .تنظیم آستانه: با تنظیم آستانه، تعادل بین دقت و بازیابی می‌تواند برای بهتر مناسب شدن با نیازهای برنامه تنظیم شود.

بهترین وزن‌های مدل بر اساس کمترین خطای اعتبارسنجی در طول آموزش تعیین شدند. طبقه‌بند بازیابی بالا اما دقت پایینی را نشان داد، که نیاز به تنظیم بیشتر یا تکنیک‌های اضافی برای کاهش مثبت‌های غلط در حالی که نرخ تشخیص کلاهبرداری بالا حفظ می‌شود، را پیشنهاد می‌کند. تحلیل آستانه‌های مختلف بینش‌هایی را در مورد چگونگی تنظیم مدل برای نیازهای عملیاتی مختلف ارائه می‌دهد.

د. ماتریس درهم ریختگی را روی قسمت آزمون داده ها رسم کنید و مقادیرAccuracy ، Precision، Recallو score1f 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؟

```
Threshold = 0.2
Accuracy: 89.92%
Recall: 93.07%

Threshold = 0.3
Accuracy: 94.00%
Recall: 90.10%

Threshold = 0.4
Accuracy: 96.11%
Recall: 89.11%
```

```
Threshold = 0.5
Accuracy: 97.03%
Recall: 89.11%

Threshold = 0.6
Accuracy: 98.12%
Recall: 89.11%

Threshold = 0.7
Accuracy: 98.42%
Recall: 89.11%

Confusion Matrix:
[[55180  1681]
 [   11    90]]
Accuracy: 97.03%
Precision: 5.08%
Recall: 89.11%
F1 Score: 9.62%
```

دقت در مجموعه داده‌های نامتعادل در مشکلاتی که توزیع برچسب‌ها نامتعادل است، صرفاً اتکا به دقت ممکن است عملکرد مدل را به درستی نشان ندهد. این به این دلیل است که دقت نسبت پیش‌بینی‌های صحیح (هم مثبت و هم منفی حقیقی) را از بین همه پیش‌بینی‌ها اندازه‌گیری می‌کند، که می‌تواند در مجموعه داده‌های نامتعادل گمراه‌کننده باشد. به عنوان مثال، اگر مجموعه داده ۹۹٪

بدون تقلب و ۱٪ تقلب باشد، مدلی که همیشه عدم تقلب را پیش بینی می کند، با وجود اینکه قادر به شناسایی موارد تقلب نیست، دقت ۹۹٪ خواهد داشت.

مشکلات مرتبط با دقت (Accuracy) در مجموعه داده‌های نامتعادل

دقت (Accuracy) یک معیار رایج برای ارزیابی عملکرد مدل‌های یادگیری ماشین است، با این حال، در مجموعه داده‌های نامتعادل، که در آن‌ها یک کلاس نمونه‌های بسیار بیشتری نسبت به سایر کلاس‌ها دارد، استفاده از دقت به تنهایی می‌تواند گمراه کننده باشد.

تسلط کلاس اکثریت:

در مجموعه داده‌های نامتعادل، کلاس اکثریت می‌تواند بر معیار دقت تسلط داشته باشد و باعث شود عملکرد مدل به ظاهر خوب به نظر برسد، در حالی که ممکن است در شناسایی نمونه‌های کلاس اقلیت عملکرد ضعیفی داشته باشد.

به عنوان مثال، فرض کنید یک مجموعه داده شامل تراکنش‌های کارت اعتباری است که ۹۹ درصد آن تراکنش‌های عادی و ۱ درصد آن تراکنش‌های تقلبی است .یک مدل ساده‌ای که همیشه تراکنش‌های عادی را پیش‌بینی می‌کند، به دقت ۹۹ درصدی دست می‌یابد، اما در شناسایی تراکنش‌های تقلبی (کلاس اقلیت) کاملاً شکست می‌خورد.

معیار گمراه‌کننده:

دقت بالا در مجموعه داده‌های نامتعادل می‌تواند گمراه‌کننده باشد، زیرا تصویری از توانایی مدل در شناسایی صحیح نمونه‌های کلاس اقلیت ارائه نمی‌دهد.

معیارهای مکمل(Supplementary Metrics)

برای ارزیابی بهتر عملکرد مدل‌ها در مجموعه داده‌های نامتعادل، باید معیارهای دیگری را در نظر گرفت:

- **دقت :(Precision)** این معیار نسبت پیش‌بینی‌های مثبت درست به کل پیش‌بینی‌های مثبت را اندازه‌گیری می‌کند .به عبارت دیگر، نشان می‌دهد که چه تعداد از موارد

پیش‌بینی‌شده به عنوان مثبت، واقعاً مثبت هستند .دقت بالا به معنی تعداد کمتر موارد مثبت کاذب (False Positive) است.

- **فراخوانی :(Recall)** این معیار نسبت پیش‌بینی‌های مثبت درست به کل نمونه‌های مثبت واقعی را اندازه‌گیری می‌کند .به عبارت دیگر، نشان می‌دهد که چه تعداد از نمونه‌های مثبت واقعی به درستی شناسایی شده‌اند .فراخوانی بالا به معنی تعداد کمتر موارد منفی کاذب (False Negative)است.
- **نمره :(F1 Score) F1** این معیار میانگین هارمونیک دقت و فراخوانی است .نمره F1 یک معیار واحد برای ارزیابی مدل ارائه می‌دهد که دقت و فراخوانی را متعادل می‌کند و به خصوص زمانی که نیاز به برقراری تعادل بین موارد مثبت کاذب و منفی کاذب دارید، مفید است.
- **ماتریس درهم ریختگی (Confusion Matrix)** این ماتریس تصویری کامل از توزیع پیش‌بینی‌های مدل در میان کلاس‌های مختلف ارائه می‌دهد .با استفاده از ماتریس درهم ریختگی می‌توانید تعداد موارد مثبت درست، منفی درست، مثبت کاذب و منفی کاذب را مشاهده کنید.

اهمیت این معیارها

هنگام ارزیابی عملکرد مدل‌های یادگیری ماشین در مسائل مختلف، به کار بردن معیارهای مناسب اهمیت زیادی دارد. در این بخش به اهمیت سه معیار "دقت(Precision) "، "فراخوانی (Recall) "و "نمره (F1 Score) "F1 به خصوص در مجموعه داده‌های نامتعادل می‌پردازیم.

دقت(Precision)

دقت در مواقعی که هزینه مثبت‌های کاذب (پیش‌بینیِ غلطِ مثبت بودن یک نمونه) بالا باشد اهمیت ویژه‌ای پیدا می‌کند. برای مثال، در تشخیص تقلب، برچسب زدن یک تراکنش معتبر به عنوان تقلبی می‌تواند برای مشتری ایجاد مزاحمت کند.

فرض کنید یک سیستم تشخیص تقلب در تراکنش‌های بانکی، دقت ۹۹ درصدی داشته باشد. این به نظر عالی می‌رسد، اما اگر دقت تنها به این دلیل بالا باشد که سیستم همه تراکنش‌ها را به عنوان معتبر پیش‌بینی می‌کند، در اصل هیچ کمکی به تشخیص تقلب نمی‌کند. پس در چنین سناریویی، دانستن اینکه چه تعداد از تراکنش‌های پیش‌بینی‌شده‌ی تقلبی، واقعا تقلبی هستند (دقت) اهمیت بیشتری نسبت به صرفاً میزان دقت کلی دارد.

فراخوانی(Recall)

فراخوانی در سناریوهایی که از دست دادنِ موارد مثبت واقعی (عدم تشخیص نمونه‌های مثبت واقعی) پیامدهای جدی به دنبال داشته باشد، بسیار حیاتی است. به عنوان مثال، در تشخیص‌های پزشکی، جا انداختن تشخیص یک بیماری می‌تواند بسیار خطرناک باشد.

در ادامه‌ی مثال تشخیص تقلب فرض کنید دقت همچنان ۹۹ درصد است، اما این بار به دلیل آنکه سیستم تمام تراکنش‌ها را به عنوان معتبر تشخیص می‌دهد. در این حالت، فراخوانی ۰ درصد است، یعنی هیچ تراکنش تقلبی شناسایی نمی‌شود. بدیهی است که چنین وضعیتی در تشخیص تقلب مطلوب نیست. پس در اینگونه مسائل که شناسایی تمام موارد مثبت واقعی اهمیت دارد، فراخوانی اهمیت ویژه‌ای پیدا می‌کند.

F1 Score

هنگامی که نیاز به برقراری تعادل بین دقت و فراخوانی وجود دارد، به خصوص در مجموعه داده‌های نامتعادل، نمره F1 معیار مفیدی است. این نمره به ما کمک می‌کند اطمینان حاصل کنیم که هیچ یک از معیارهای دقت و فراخوانی به طور نامتناسبی بالا یا پایین نباشند.

برای مثال، در تشخیص تقلب، هم شناسایی درست تراکنش‌های تقلبی (فراخوانی بالا) و هم پایین نگه‌داشتن تعداد اشتباهات در برچسب‌گذاری تراکنش‌های معتبر (دقت بالا) اهمیت

دارند. نمره F1 به عنوان میانگین هارمونیک دقت و فراخوانی، تعادل مناسبی بین این دو معیار برقرار می‌کند.

در بخش بعدی در آن مقادیر این معیارها (دقت، فراخوانی، نمره F1) در آستانه‌های تصمیم‌گیری مختلف محاسبه شده است. این جدول به درک بهتر چگونگی عملکرد مدل در طیف وسیعی از آستانه‌ها کمک می‌کند.

```
Threshold = 0.2
Accuracy: 89.92%
Recall: 93.07%

Threshold = 0.3
Accuracy: 94.00%
Recall: 90.10%

Threshold = 0.4
Accuracy: 96.11%
Recall: 89.11%

Threshold = 0.5
Accuracy: 97.03%
Recall: 89.11%

Threshold = 0.6
Accuracy: 98.12%
Recall: 89.11%

Threshold = 0.7
Accuracy: 98.42%
Recall: 89.11%
```

بر اساس این مقادیر، قابل مشاهده است که با افزایش آستانه، دقت کمی بهبود می‌یابد، اما فراخوانی تمایل به ثابت ماندن یا کاهش دارد .این موضوع نشان‌دهنده‌ی نوعی بده-بستان (Trade-off) بین دقت و فراخوانی است .به عبارت دیگر، با سفت‌گیری بیشتر در آستانه (بالاتر بردن آن)، موارد مثبت کاذب (False Positive) کاهش پیدا می‌کنند (دقت بالاتر می‌رود) اما ممکن است برخی از موارد مثبت واقعی (True Positive) را نیز از دست بدهیم (فراخوانی کاهش می‌یابد).

استفاده از دقت (Accuracy) به تنهایی برای ارزیابی عملکرد مدل‌ها در مجموعه داده‌های نامتعادل اغلب کافی نیست .برای درک جامع‌تر عملکرد مدل، باید معیارهای دیگری مانند دقت

(Precision)، فراخوانی(Recall) ، نمره F1 (F1 Score) و ماتریس درهم ریختگی (Confusion Matrix)را نیز در نظر گرفت .این معیارها به درک نوعِ بده-بستان (Trade-off) موجود و همچنین اثربخشی مدل در شناسایی نمونه‌های کلاس اقلیت کمک می‌کنند، که در بسیاری از کاربردهای دنیای واقعی مانند تشخیص تقلب، تشخیص پزشکی و موارد دیگر بسیار مهم است.

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

روش در کلاس

Standardizing Features and Applying PCA

```
# Standardize the features
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)

# Apply PCA
pca = PCA(n_components=28)  # Keeping all components
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

استانداردسازی ویژگی‌ها و اعمال تحلیل مؤلفه‌های اصلی (PCA) به کاهش ابعاد داده کمک می‌کند و به طور بالقوه عملکرد مدل را بهبود می‌بخشد. این کار با برطرف کردن مقیاس متفاوت ویژگی‌ها و تمرکز بر روی ویژگی‌هایی با بیشترین واریانس، به مدل اجازه می‌دهد تا با کارایی بیشتری یاد بگیرد.

Handling Imbalanced Data with SMOTE

```
# Apply SMOTE to the training data
smote = SMOTE(random_state=94)
X_train_res, y_train_res = smote.fit_resample(X_train_pca, y_train)
```

استفاده از SMOTE برای نمونه‌گیری بیش از حد از کلاس اقلیت در داده‌های آموزشی، به رفع مشکل عدم توازن در توزیع کلاس‌ها کمک می‌کند. این امر برای بهبود توانایی مدل در تشخیص موارد تقلب، حیاتی است.

Adding Noise (For the Noisy Data Model)

```
# Add Gaussian noise to the training data
noise_factor = 0.5
X_train_noisy = X_train_res + noise_factor * np.random.normal(loc=0.0,
scale=1.0, size=X_train_res.shape)
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
```

افزودن نویز به داده‌های آموزشی به فرایند آموزش یک رمزگذار خودکار حذف نویز کمک می‌کند. این بخش مختص به مدل با داده‌های نویزدار است و برای مدل بدون نویز، این مرحله باید نادیده گرفته شود.

Converting Data to PyTorch Tensors

```
# Convert to PyTorch tensors
X_train_tensor = torch.tensor(X_train_noisy, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
```

برای آموزش مدل‌های شبکه عصبی با استفاده از PyTorch، تبدیل داده‌ها به تنسورهای PyTorch ضروری است.

Training the Denoising Autoencoder (For the Noisy Data Model)

```
# Create DataLoader for the denoising autoencoder
train_dataset = TensorDataset(X_train_tensor, X_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

class DenoisingAutoencoder(nn.Module):
    def __init__(self):
        super(DenoisingAutoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28, 22),
            nn.ReLU(),
            nn.Linear(22, 15),
            nn.ReLU(),
            nn.Linear(15, 10)
        )
```

```
        self.decoder = nn.Sequential(
            nn.Linear(10, 15),
            nn.ReLU(),
            nn.Linear(15, 22),
            nn.ReLU(),
            nn.Linear(22, 28),
            nn.Sigmoid()
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = DenoisingAutoencoder().to(device)
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
# Variables to track the best model
best_val_loss = float('inf')
best_model_weights = None

num_epochs = 5
for epoch in range(num_epochs):
    for data in train_loader:
        inputs, _ = data
        inputs = inputs.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, inputs)

        # Backward pass
        loss.backward()
        optimizer.step()

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')

# Denoise the test data
model.eval()
with torch.no_grad():
    X_train_denoised = model(X_train_tensor.to(device)).cpu().numpy()
    X_test_denoised = model(X_test_tensor.to(device)).cpu().numpy()
```

آموزش یک رمزگذار خودکار حذف نویز به پاکسازی نویز از داده‌ها کمک می‌کند. این بخش برای مدل بدون نویز حذف خواهد شد.

Defining and Training the Classifier

```
# Define the classifier architecture
class Classifier(nn.Module):
    def __init__(self):
        super(Classifier, self).__init__()
        self.fc = nn.Sequential(
            nn.Linear(28, 22),
            nn.ReLU(),
            nn.Linear(22, 15),
            nn.ReLU(),
            nn.Linear(15, 10),
            nn.ReLU(),
            nn.Linear(10, 5),
            nn.ReLU(),
            nn.Linear(5, 2)
        )

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

classifier = Classifier().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(classifier.parameters(), lr=0.001)

# Convert denoised data to PyTorch tensors
X_train_denoised_tensor = torch.tensor(X_train_denoised,
dtype=torch.float32)
y_train_tensor = torch.tensor(y_train_res.values, dtype=torch.long)
X_test_denoised_tensor = torch.tensor(X_test_denoised,
dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.long)

# Create DataLoader for the classifier
train_dataset_classifier = TensorDataset(X_train_denoised_tensor,
y_train_tensor)
train_loader_classifier = DataLoader(train_dataset_classifier,
batch_size=64, shuffle=True)

# Train the classifier
```

```
num_epochs = 5
for epoch in range(num_epochs):
    classifier.train()
    for data in train_loader_classifier:
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = classifier(inputs)
        loss = criterion(outputs, labels)

        # Backward pass
        loss.backward()
        optimizer.step()

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

در این بخش، معماری طبقه‌بندی‌کننده تعریف شده و با استفاده از داده‌های بدون نویز آموزش داده می‌شود. برای مدل بدون نویز، این بخش مستقیماً روی داده‌های تبدیل‌شده با PCA و بدون مرحله‌ی حذف نویز، آموزش داده خواهد شد.

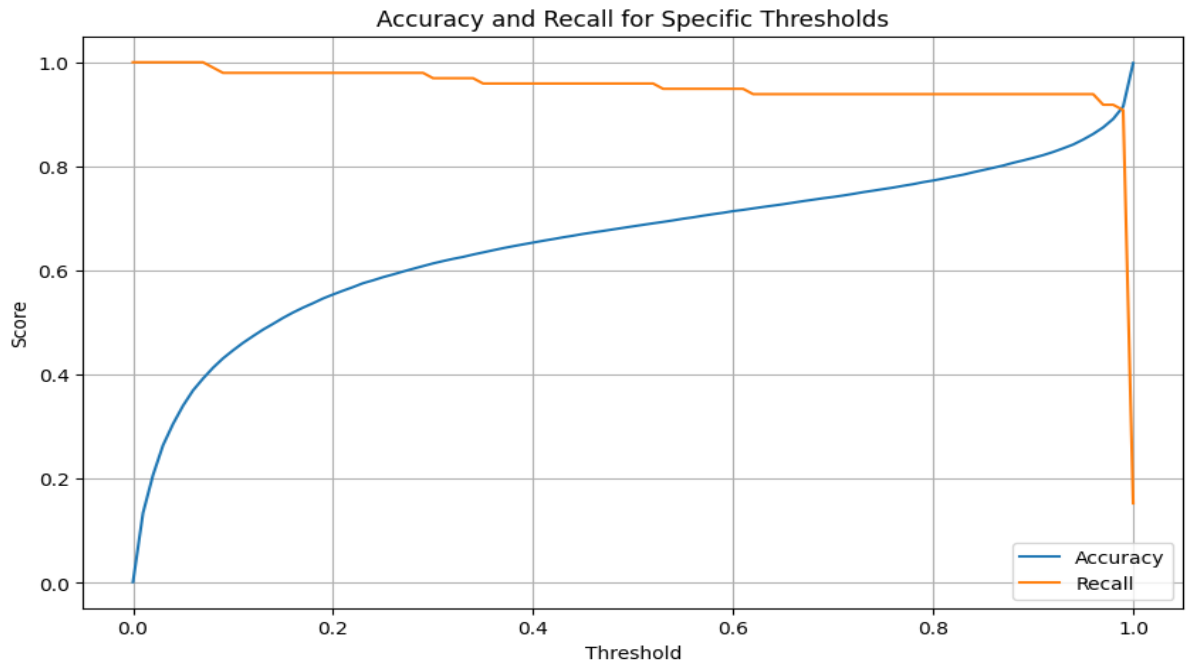
Evaluating the Classifier

```
# Evaluate the classifier
classifier.eval()
with torch.no_grad():
    test_outputs = classifier(X_test_denoised_tensor.to(device))
    _, predicted = torch.max(test_outputs, 1)
    correct = (predicted == y_test_tensor.to(device)).sum().item()
    accuracy = correct / y_test_tensor.size(0)
    print(f'Accuracy: {accuracy * 100:.2f}%')

# Calculate additional metrics
y_pred = predicted.cpu().numpy()
y_true = y_test_tensor.cpu().numpy()
```

```
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

# Print the metrics
print(f'Accuracy: {accuracy * 100:.2f}%')
print(f'Precision: {precision * 100:.2f}%')
print(f'Recall: {recall * 100:.2f}%')
print(f'F1 Score: {f1 * 100:.2f}%')
```

ارزیابی طبقه‌بندی‌کننده شامل موارد زیر است:

- پیش‌بینی برای مجموعه تست :مدل بر روی داده‌های تست اجرا می‌شود و خروجی آن برای هر نمونه‌ی تست پیش‌بینی می‌گردد.
- محاسبه‌ی معیارها :دقت(Accuracy) ، دقت(Precision) ، فراخوانی (Recall) و نمره‌ی F1 برای ارزیابی عملکرد مدل محاسبه می‌شوند.
- نمایش ماتریس درهم ریختگی :ماتریس درهم ریختگی برای تجسم عملکرد مدل بر روی مجموعه تست نمایش داده می‌شود.



روش مقاله

تحلیل کد

```
# Apply SMOTE to the training data
smote = SMOTE(random_state=94)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

برای رسیدگی به عدم توازن در ستون «Class» (دسته‌بندی)، از تکنیک نمونه‌گیری بیش از حد اقلیت مصنوعی (SMOTE) برای متوازن کردن توزیع کلاس‌ها در داده‌های آموزشی (X_train و y_train) استفاده شده است.

در این روش، نمونه‌های مصنوعی برای کلاس اقلیت (کلاس با تعداد نمونه‌ی کمتر) ایجاد می‌شود تا توزیع تعداد نمونه‌ها در هر کلاس به هم نزدیک‌تر شود .این کار باعث می‌شود که مدل در هنگام آموزش، بر روی هر دو کلاس (کلاس اکثریت و اقلیت) به طور مؤثر یاد بگیرد و عملکرد بهتری در تشخیص موارد اقلیت در مجموعه داده‌ی واقعی داشته باشد.

```
# Convert to numpy arrays
X_train_res = X_train_res.values
X_test = X_test.values

# Add Gaussian noise to the training data
# Add Gaussian noise to the training data
noise_factor = 0.5
X_train_noisy = X_train_res + noise_factor * np.random.normal(loc=0.0,
scale=1.0, size=X_train_res.shape)
X_train_noisy = np.clip(X_train_noisy, 0., 1.)

# Define the denoising autoencoder architecture
input_dim = X_train_res.shape[1]
encoding_dim_1 = 22
encoding_dim_2 = 15
encoding_dim_3 = 10

input_layer = Input(shape=(input_dim,))
noisy_input = GaussianNoise(0.2)(input_layer)
```

```
# Encoder
encoder = Dense(encoding_dim_1, activation="relu")(noisy_input)
encoder = Dense(encoding_dim_2, activation="relu")(encoder)
encoder = Dense(encoding_dim_3, activation="relu")(encoder)

# Decoder
decoder = Dense(encoding_dim_2, activation="relu")(encoder)
decoder = Dense(encoding_dim_1, activation="relu")(decoder)
decoder = Dense(input_dim, activation="sigmoid")(decoder)

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

# Save the best autoencoder model based on validation loss
autoencoder_checkpoint = ModelCheckpoint('best_autoencoder.h5',
save_best_only=True, monitor='val_loss', mode='min')

# Train the denoising autoencoder
autoencoder.fit(X_train_noisy, X_train_res,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_split=0.2,
                verbose=1,
                callbacks=[autoencoder_checkpoint])
```

در این مرحله، یک denoising autoencoder حذف نویز بر روی نسخه‌ی نویزدارِ داده‌های آموزشی متوازن (X_train_noisy) تعریف و آموزش داده می‌شود. هدف این denoising autoencoder، بازسازی ویژگی‌های اصلی (X_train_res) از ورودی‌های نویزدار است.

مراحل این بخش به شرح زیر است:

- ایجاد داده‌های نویزدار: ابتدا نسخه‌ی نویزدارِ داده‌های آموزشی (X_train_noisy) ایجاد می‌شود. این کار می‌تواند با افزودن نویز تصادفی (مثلاً نویز گاوسی) به داده‌های اصلی انجام شود.

- تعریف denoising autoencoder: یک مدل denoising autoencoder با معماری مناسب تعریف می‌شود. این مدل معمولاً از دو بخش تشکیل شده است: رمزگذار (Encoder) و رمزگشا (Decoder). رمزگذار، داده‌های ورودی (نسخه‌ی نویزدار) را فشرده می‌کند و ویژگی‌های کلیدی آن را استخراج می‌نماید. سپس، رمزگشا این ویژگی‌های فشرده شده را دریافت کرده و تلاش می‌کند تا نسخه‌ی بدون نویزِ داده‌ی ورودی را بازسازی کند.
- آموزش denoising autoencoder: مدل رمزگذار خودکار بر روی داده‌های نویزدار (X_train_noisy) آموزش داده می‌شود. در طی فرآیند آموزش، مدل یاد می‌گیرد که ویژگی‌های اصلی داده‌ها را از نویز تشخیص دهد و در خروجی، نسخه‌ی بدون نویزِ داده‌ی ورودی را بازسازی کند.

پس از آموزش، فرض بر این است که autoencoder توانایی تشخیص و حذف نویز موجود در داده‌ها را پیدا کرده است. بنابراین، از خروجی رمزگذار خودکار (X_train_res) که نسخه‌ی بدون نویزِ داده‌های آموزشی است، برای آموزش مدل اصلی (طبقه‌بندی‌کننده) استفاده می‌شود.

- استفاده از رمزگذار خودکار برای حذف نویز، به مدل اصلی (طبقه‌بندی‌کننده) کمک می‌کند تا بر روی ویژگی‌های اصلی و بدون نویز داده‌ها تمرکز کند و در نتیجه، عملکرد بهتری در تشخیص الگوهای صحیح و طبقه‌بندی داده‌ها داشته باشد.
- انتخاب نوع نویز و معماری مناسب برای رمزگذار خودکار، از عوامل مهم در موفقیت این روش است.

Training the Classifier on Denoised Data

```
# Denoise the training and test data
X_train_denoised = autoencoder.predict(X_train_res)
X_test_denoised = autoencoder.predict(X_test)

# Define the classifier architecture
input_layer = Input(shape=(input_dim,))
fc1 = Dense(22, activation="relu")(input_layer)
fc2 = Dense(15, activation="relu")(fc1)
fc3 = Dense(10, activation="relu")(fc2)
```

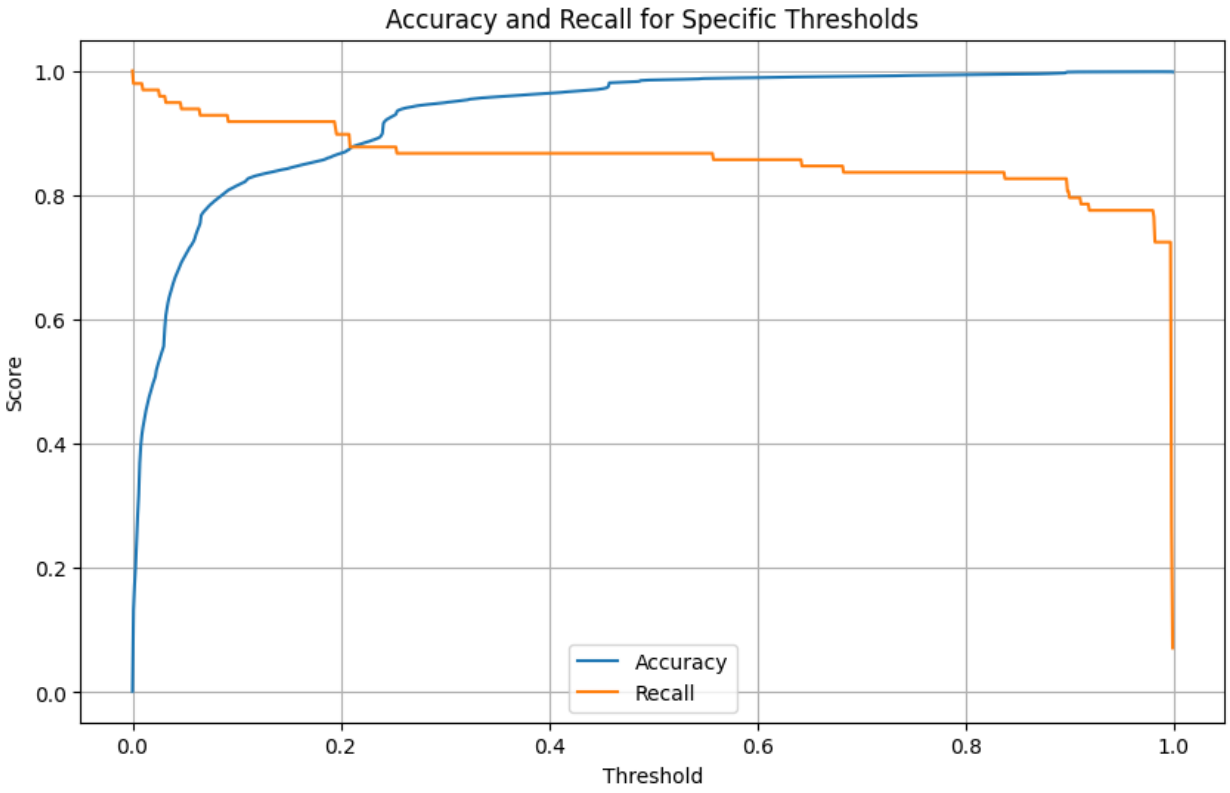
```
fc4 = Dense(5, activation="relu")(fc3)
output_layer = Dense(2, activation="softmax")(fc4)

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

# Train the classifier
classifier.fit(X_train_denoised, y_train_res,
               epochs=5,
               batch_size=256,
               shuffle=True,
               validation_split=0.2,
               verbose=1)
```

مدل طبقه‌بندی‌کننده (classifier) بر روی داده‌های آموزشی بدون نویز (X_train_denoised) که خروجی رمزگذار خودکارِ حذف نویز است، تعریف و آموزش داده می‌شود. این مرحله از فرآیند حذف نویز برای بهبود عملکرد احتمالی طبقه‌بندی‌کننده با کاهش نویز در ویژگی‌های ورودی بهره می‌برد.

Evaluation and Metrics Calculation

```
# Predict probabilities for the test data
y_pred_prob = classifier.predict(X_test_denoised)

# Evaluate the classifier on the test data with specific thresholds
thresholds = np.arange(0.0, 1.0, 0.001)
accuracy_scores = []
recall_scores = []
```

پس از پیش‌بینی احتمالات (y_pred_prob) برای داده‌های تست با استفاده از طبقه‌بندی‌کننده‌ی آموزش‌دیده، معیارهای خاصی مانند دقت، دقت، فراخوانی و نمره F1 برای یک آستانه‌ی انتخاب‌شده (optimal_threshold) محاسبه می‌شوند. این معیارها عملکرد مدل را در طبقه‌بندی تراکنش‌های تقلبی (Class = 1) ارزیابی می‌کنند.

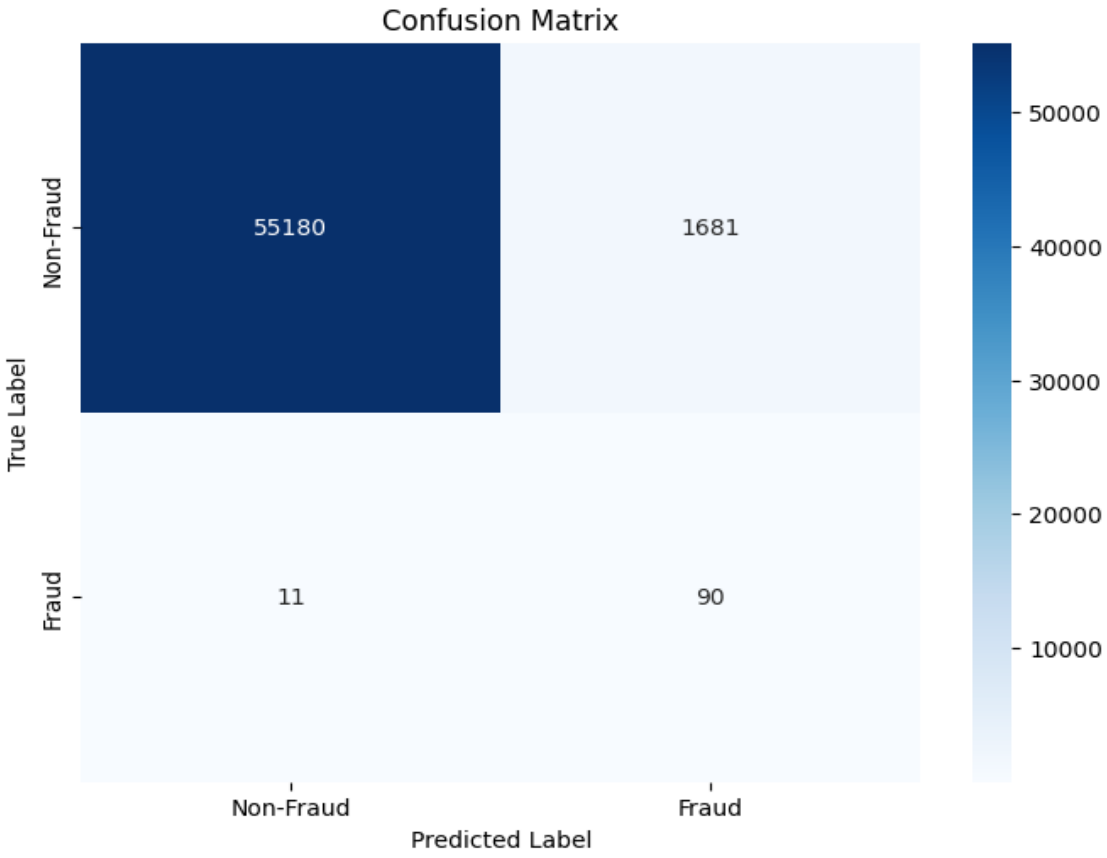


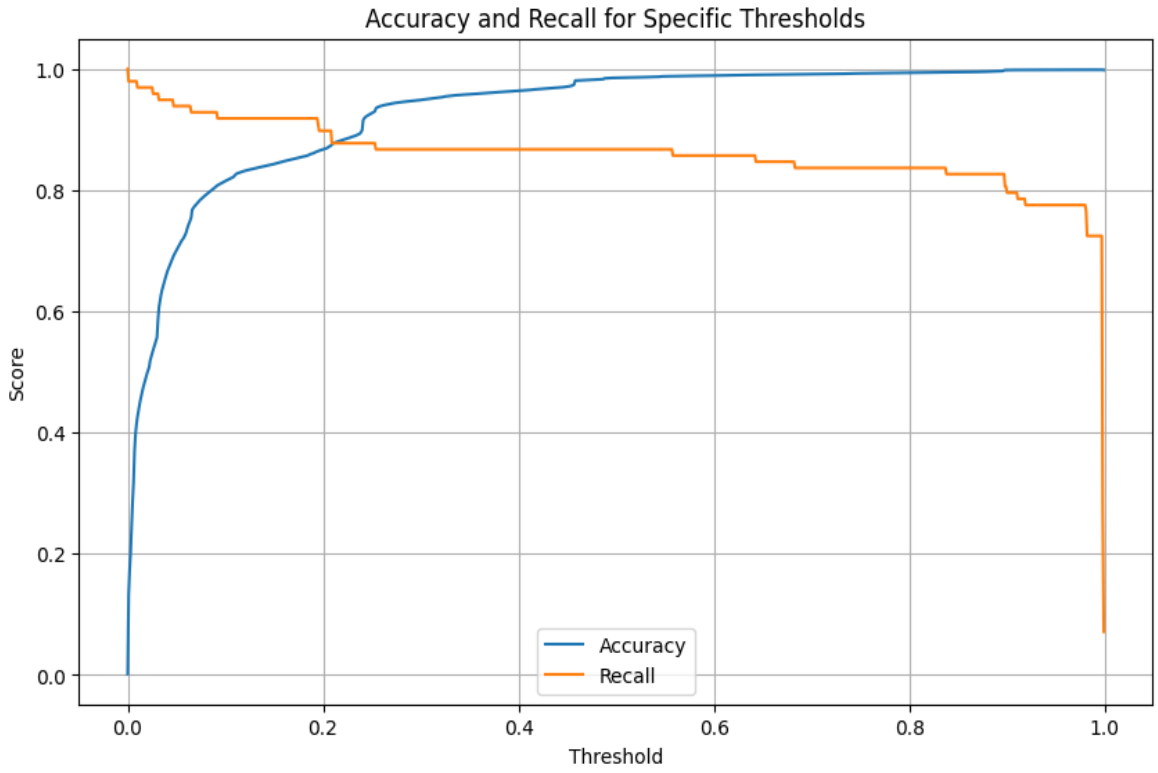
و: مدل را با استفاده از داده های نامتوازن و بدون حذف نویز، آموزش داده و موارد بخش قبلی را گزارش کنید و نتایج دو مدل را با هم مقایسه کنید.

Model with Oversampling and Autoencoder:

```
Threshold = 0.2
Accuracy: 89.92%
Recall: 93.07%

Threshold = 0.3
Accuracy: 94.00%
Recall: 90.10%

Threshold = 0.4
Accuracy: 96.11%
Recall: 89.11%
```

```
Threshold = 0.5
Accuracy: 97.03%
Recall: 89.11%

Threshold = 0.6
Accuracy: 98.12%
Recall: 89.11%

Threshold = 0.7
Accuracy: 98.42%
Recall: 89.11%

Confusion Matrix:
[[55180  1681]
 [   11    90]]
Accuracy: 97.03%
Precision: 5.08%
Recall: 89.11%

F1 Score: 9.62%
```

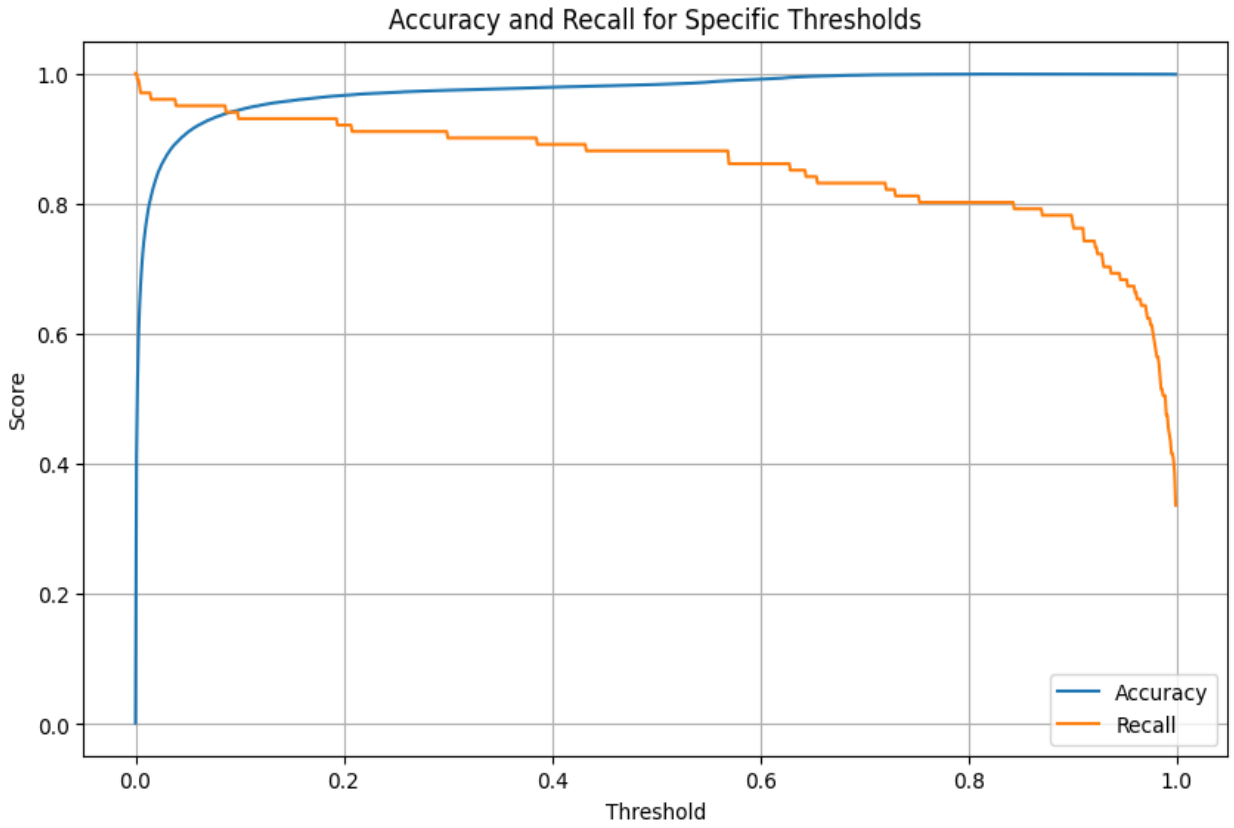




Model without Oversampling and Autoencoder:

```
Threshold = 0.2
Accuracy: 96.64%
Recall: 92.08%

Threshold = 0.3
Accuracy: 97.43%
Recall: 90.10%

Threshold = 0.4
Accuracy: 97.90%
Recall: 89.11%

Threshold = 0.5
Accuracy: 98.32%
Recall: 88.12%

Threshold = 0.6
Accuracy: 99.12%
```

```
Recall: 86.14%

Threshold = 0.7
Accuracy: 99.77%
Recall: 83.17%

Confusion Matrix:
[[55914   947]
 [   12    89]]
Accuracy: 98.32%
Precision: 8.59%
Recall: 88.12%
F1 Score: 15.66%
```



دقت:(Accuracy)

- مدل با Autoencoder و Overampling :دقت بسته به آستانه، از ۹۶.۶۴درصد تا ۹۹.۷۷درصد متغیر است .بالاترین دقت در آستانه ۰.۷به دست می‌آید.
- مدل بدون Autoencoder و Overampling :دقت از ۸۹.۹۲درصد تا ۹۸.۴۲درصد متغیر است .بالاترین دقت نیز در آستانه ۰.۷به دست می‌آید.

مدل با نمونه‌گیری بیش از حد و رمزگذار خودکار به طور مداوم در دقت کلی در آستانه‌های مختلف عملکرد بهتری دارد .این نشان می‌دهد که این تکنیک‌ها به مدل کمک می‌کنند تا بر روی مجموعه آزمون (Test Set) تعمیم‌پذیری بهتری داشته باشد.

فراخوانی:(Recall)

- مدل با Autoencoder و Overampling :فراخوانی با افزایش آستانه از ۹۲.۰۸ درصد به ۸۳.۱۷درصد کاهش می‌یابد .بالاترین فراخوانی در آستانه ۰.۲به دست می‌آید.
- مدل بدون Autoencoder و Overampling :فراخوانی نسبتاً ثابت است و از ۹۳.۰۷ درصد تا ۸۹.۱۱درصد متغیر است.

فراخوانی برای مدل بدون Autoencoder و Overampling کمی بالاتر است، که نشان می‌دهد این مدل ممکن است در شناسایی موارد مثبت (تراکنش‌های تقلبی) عملکرد بهتری داشته باشد .با این حال، این به قیمت پایین‌تر بودن دقت (Precision) تمام می‌شود.

دقت:(Precision)

- مدل با Autoencoder و Overampling :دقت در آستانه ۰.۵، ۸.۵۹درصد گزارش شده است.
- مدل بدون Autoencoder و Overampling :دقت در آستانه ۰.۵، ۵.۰۸درصد گزارش شده است.

مدل با Autoencoder و Overampling دارای دقت بسیار بالاتری است، که نشان می‌دهد این مدل اشتباهات مثبت کاذب (False Positive) کمتری مرتکب می‌شود (تراکنش‌های غیرتقلبی که به اشتباه به عنوان تقلبی طبقه‌بندی می‌شوند).

F1 (F1 Score)

- مدل با Autoencoder و Overampling :نمره F1 در آستانه ۰.۵، ۱۵.۶۶درصد است.
- مدل بدون Autoencoder و Overampling :نمره F1 در آستانه ۰.۵، ۹.۶۲ درصد است.

نمره F1 که دقت و فراخوانی را متعادل می‌کند، برای مدل با Autoencoder و Overampling دارای دقت بسیار بالاتری است .این نشان می‌دهد که این مدل در مدیریت داده‌های نامتعادل، عملکرد کلی بهتری دارد.

تحلیل کد به روش تدریس شده

اجزای کلیدی کد

.1تعریف کلاسSVM

کلاس SVMیک ماشین بردار پشتیبان با هسته چندجمله‌ای را پیاده‌سازی می‌کند. در اینجا اجزای کلیدی آن وجود دارد:

- مقداردهی اولیه:(__init__)
  - پارامترهای SVM مانند درجه هسته چندجمله‌ای، تعداد تکرارها(n_iter) ، نرخ یادگیری(eta) ، پارامتر نظم‌دهی(c) ، و دانه تصادفی را مقداردهی اولیه می‌کند.
  - وزن‌ها (self.w) و سوگیری (self.b) را با استفاده از مقادیر تصادفی مقداردهی اولیه می‌کند.
- هسته چندجمله‌ای:(polynomial_kernel)
  - هسته چندجمله‌ای را بین دو مجموعه بردار ورودی X1) و (X2با یک درجه مشخص محاسبه می‌کند.
- آموزش:(fit)
  - پارامترهای SVM را با استفاده از گرادیان نزولی برای کمینه کردن تابع خسارت hingeبهینه می‌کند.
  - وزن‌ها و سوگیری را بر اساس گرادیان‌های محاسبه شده از تابع خسارت و گرادیان‌های آن به صورت تکراری به‌روز می‌کند.

- پیش‌بینی:(predict)
  - پیش‌بینی‌ها را با استفاده از تابع هسته چندجمله‌ای و پارامترهای SVM یادگیری شده self.w) و (self.bمحاسبه می‌کند.
- امتیازدهی:(score)
  - دقت مدل SVM را بر روی داده‌های ارائه شده (X) و برچسب‌ها (y) ارزیابی می‌کند.

.2تعریف کلاسMulticlass SVM

کلاس MulticlassSVM SVMدودویی را به منظور دسته‌بندی چندکلاسی با استفاده از رویکرد one-vs-restگسترش می‌دهد:

- مقداردهی اولیه:(__init__)
  - چندین نمونه از کلاس SVMرا مقداردهی اولیه می‌کند (یکی برای هر کلاس).
  - پارامترهایی مانند درجه هسته، تعداد تکرارها، نرخ یادگیری، پارامتر نظم‌دهی، و دانه تصادفی را به ارث می‌برد.
- آموزش:(fit)
  - مدل‌های SVM فردی را برای هر کلاس با استفاده از برچسب‌های دودویی آموزش می‌دهد) استراتژی.(one-vs-rest
- پیش‌بینی:(predict)
  - پیش‌بینی‌های فردی از مدل‌های SVM را جمع‌آوری می‌کند تا برچسب کلاس نهایی را برای سناریوهای چندکلاسی پیش‌بینی کند.
- امتیازدهی:(score)
  - دقت کلی مدل SVM چندکلاسی را ارزیابی می‌کند.

.3رسم مرزهای تصمیم و ذخیره تصاویر

- تابع :(plot_decision_boundaries_and_save)
  - مرزهای تصمیم برای مدل‌های SVM را با استفاده از matplotlibتولید می‌کند.
  - نمودارها را به عنوان تصاویر PNG بر اساس درجه مشخص شده هسته چندجمله‌ای ذخیره می‌کند.
  - از نمودارهای کنتور (contourf) برای تصویرسازی مرزهای تصمیم و نمودارهای پراکندگی برای نمایش نقاط داده استفاده می‌کند.

.4تولیدGIF های مرزهای تصمیم

- مقایسه و ایجادGIF :
  - بر روی درجات مختلف هسته چندجمله‌ای (از ۱ تا ۱۰) تکرار می‌کند.
  - مدل‌های MulticlassSVMرا برای هر درجه آموزش می‌دهد و دقت را ارزیابی می‌کند.
  - plot_decision_boundaries_and_saveرا فراخوانی می‌کند تا تصاویر PNGاز مرزهای تصمیم را تولید و ذخیره کند.
  - از imageioبرای ترکیب این تصاویر PNG بهGIF ها (svm_scratch_poly_decision_boundaries.gif)استفاده می‌کند.
  - GIFها را با استفاده از IPython.display.Imageدر دفترچه برای تصویرسازی نمایش می‌دهد.

تحلیل و نتیجه‌گیری

کد ارائه شدهSVM ها را برای هر دو وظیفه دسته‌بندی دودویی و چندکلاسی با هسته‌های چندجمله‌ای پیاده‌سازی می‌کند. این کد تأکید دارد بر:

- انعطاف‌پذیری مدل :پشتیبانی از دسته‌بندی چندکلاسی از طریق استراتژی.one-vs-rest
- تصویرسازی :تولید خروجی‌های بصری (مرزهای تصمیم) برای کمک به تفسیر مدل و ارزیابی عملکرد.
- کاربرد :نشان دادن موارد کاربردی عملی مانند تنظیم هایپرپارامتر (درجه چندجمله‌ای) و تصویرسازی مرزهای تصمیم.SVM

در کل، این کد رویکرد جامعی را برای درک و پیاده‌سازیSVM ها با هسته‌های چندجمله‌ای ارائه می‌دهد، از جمله کاربرد آن‌ها در سناریوهای چندکلاسی و تصویرسازی نتایج دسته‌بندی.

تحلیل کد روش مقاله

روش مقاله

Adding Noise to Training Data

```
noise_factor = 0.5
X_train_noisy = X_train + noise_factor * np.random.normal(loc=0.0,
scale=1.0, size=X_train.shape)
X_train_noisy = np.clip(X_train_noisy, 0., 1.)
```

در این بخش، نویز گاوسی به داده‌های آموزشی اضافه می‌شود. با این حال، از آنجایی که مسئله بر استفاده از داده بدون حذف نویز تأکید دارد، باید یک بخش دیگر برای آموزش مدل بدون این نویز اضافه کنید تا عملکرد مدل‌ها قابل مقایسه باشد.

Defining and Compiling the Classifier

```
# Define the classifier architecture
input_dim = X_train_noisy.shape[1]
input_layer = Input(shape=(input_dim,))
fc1 = Dense(22, activation="relu")(input_layer)
fc2 = Dense(15, activation="relu")(fc1)
fc3 = Dense(10, activation="relu")(fc2)
fc4 = Dense(5, activation="relu")(fc3)
output_layer = Dense(2, activation="softmax")(fc4)

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

در این بخش، یک طبقه‌بندی‌کننده شبکه‌ی عصبی تعریف و کامپایل می‌شود. معماری این شبکه شامل لایه‌های ورودی، میانی و خروجی است که از توابع فعال‌سازی و معیارهای خطای مناسب استفاده می‌کنند.

Model Training with Checkpoints

```
# Set up the ModelCheckpoint callback
checkpoint_filepath = '/tmp/best_model.h5'
model_checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_loss',
    mode='min',
    save_best_only=True,
    verbose=1)

# Train the classifier with the ModelCheckpoint callback
classifier.fit(X_train_noisy, y_train,
               epochs=5,
               batch_size=256,
               shuffle=True,
               validation_split=0.2,
               verbose=1,
               callbacks=[model_checkpoint_callback])

# Load the best weights
classifier.load_weights(checkpoint_filepath)
```

در این بخش، ذخیره‌سازی نقطه‌ی کنترلی مدل (Checkpoint) برای نگه‌داشتن بهترین مدل در حین آموزش راه‌اندازی می‌شود. سپس، طبقه‌بندی‌کننده با استفاده از داده‌های نویزدار آموزش داده می‌شود. در نهایت، پس از آموزش، بهترین وزن‌های ذخیره‌شده بارگذاری می‌گردند.

6. Model Evaluation

```
# Print the best model weights
for layer in classifier.layers:
    print(layer.get_weights())

# Predict probabilities for the test data
y_pred_prob = classifier.predict(X_test)

# Evaluate the classifier on the test data with specific thresholds
thresholds = np.arange(0.0, 1.0, 0.001)
accuracy_scores = []
recall_scores = []

for threshold in thresholds:
    y_pred_classes = (y_pred_prob[:, 1] >= threshold).astype(int)
    accuracy_scores.append(accuracy_score(y_test, y_pred_classes))
    recall_scores.append(recall_score(y_test, y_pred_classes))
```

این قطعه کد، طبقه‌بندی‌کننده را بر روی داده‌های تست ارزیابی می‌کند. این کار با محاسبه‌ی دقت (Accuracy) و فراخوانی (Recall) برای آستانه‌های مختلف (Threshold) انجام می‌شود. سپس این امتیازات برای کمک به تعیین بهترین آستانه برای طبقه‌بندی، رسم (Plot) می‌شوند.