مبانی سیستم‌های هوشمند

مینی پروژه شماره ۱

نام و نام خانوادگی	محمد خلیلی
شمارهٔ دانشجویی	۹۹۲۵۹۷۳
تاریخ	آذر ۱۴۰۳
GitHub	m۱۵kh/proj۱_kntu
Colab	drive

فهرست مطالب

فهرست تصاویر

۱ سوال اول

۱.۱ درباره دیتاست

- هدف این دیتاست این است که به بانک کمک کند تا مشتریانی را که احتمال دارد خدمات کارت اعتباری را ترک کنند، شناسایی کند. با استفاده از اطلاعات موجود در دیتاست، بانک می‌تواند الگوها و عواملی را که منجر به ترک خدمات توسط مشتریان می‌شود، تحلیل کند. این پیش‌بینی به بانک اجازه می‌دهد تا به‌صورت پیشگیرانه به این مشتریان مراجعه کند، خدمات بهتری به آن‌ها ارائه دهد و تلاش کند تا تصمیم آن‌ها را برای ترک خدمات تغییر دهد. این رویکرد می‌تواند به بانک کمک کند تا نرخ نگهداری مشتریان خود را افزایش داده و از کاهش درآمد جلوگیری کند.تعداد فیچر ها ی دیتاست عبارت اند
- تعداد فیچر ها با دستور زیر بدست می آید

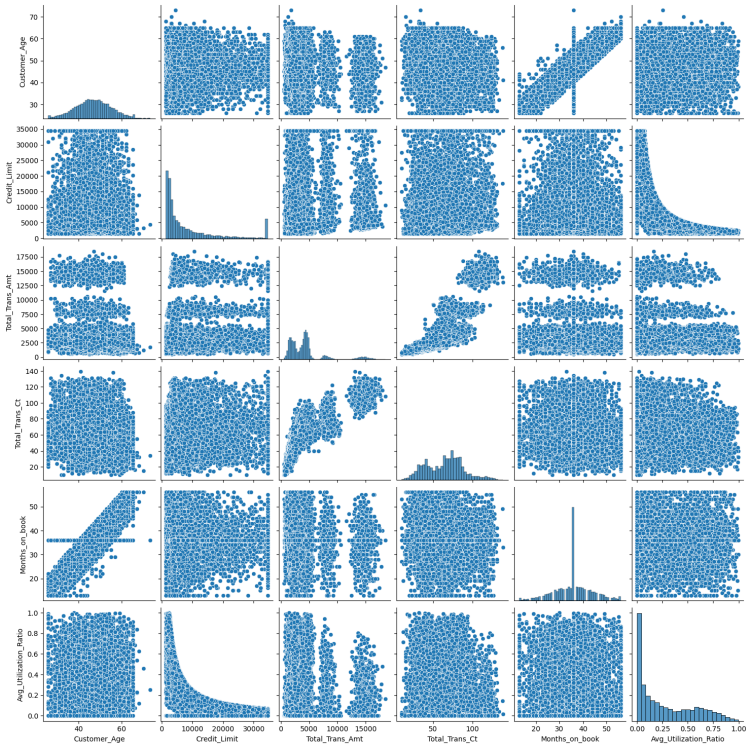
```
import pandas as pd
# Load the xsv file (assuming it's a CSV file for this example)
df = pd.read_csv('BankChurners.csv', delimiter=',')  # Adjust the delimiter if necessary
# Print the title of each column
for column in df.columns:
print(column)

```

- تعداد سمبل های دیتاست برابر با ۱۰۱۲۷ هست

۲.۱ نمایش فیچر ها با sns.pairplot

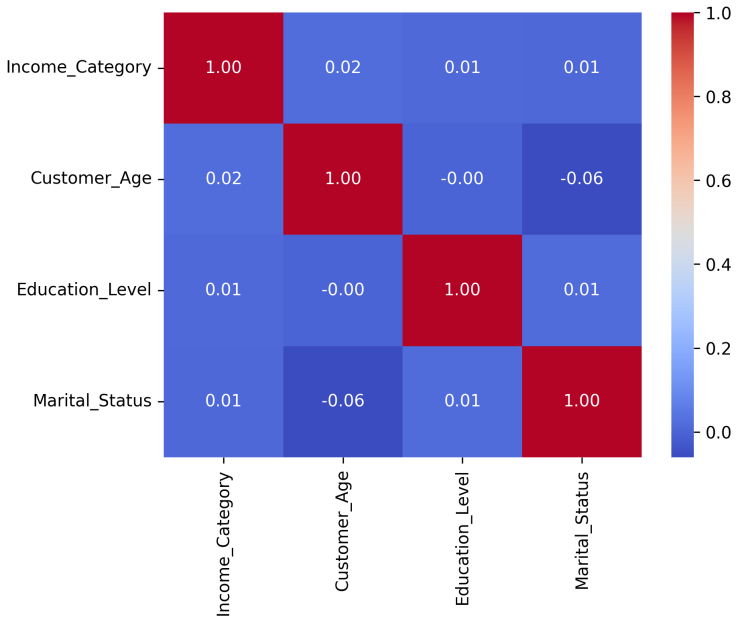
در این بخش، ویژگی‌های انتخابی که برای نمایش در نمودار پخش (pairplot) استفاده کردیم عبارتند از:

- `Customer_Age`
- `Credit_Limit`
- `Total_Trans_Amt`
- `Total_Trans_Ct`
- `Months_on_book`
- `Avg_Utilization_Ratio`



شکل ۱: نمایش برخی ازویژگی ها

۳.۱ نمایش همبستگی فیچر ها



شکل ۲: نمایش heatmap

۴.۱ Nans بودن دیتا

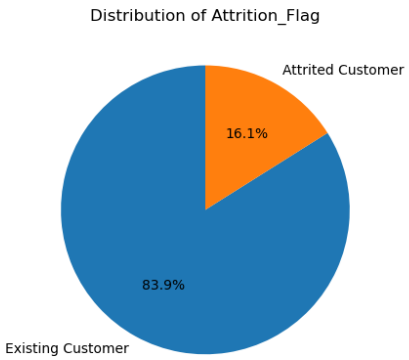
با دستور زیر متوجه میشویم آیا داده nans وجود دارد یا نه

```
if df.isnull().values.any():
    print("There are NaN values in the dataset.")
else:
    print("No NaN values in the dataset.")

```

خروجی این کد مشخص می‌کند که داده خالی nans وجود ندارد

۵.۱ درباره ویژگی Flag Attrition

کلا دوتا کلاس داریم بنام‌های ['Existing Customer', 'Attrited Customer'].



شکل ۳: نمایش plot pie

برای بررسی عدم بالانس داده از کد زیر استفاده می‌کنیم:

```
print(df["Attrition_Flag"].value_counts())
```

عدم بالانس داده‌ها

عدم بالانس داده‌ها (Imbalanced Data) یکی از چالش‌های مهم در آموزش مدل‌های یادگیری ماشین است. زمانی که توزیع داده‌های کلاس‌های مختلف در مجموعه داده‌ها نابرابر باشد، مدل تمایل دارد به سمت کلاسی که تعداد نمونه‌های بیشتری دارد، Bias شود. این موضوع می‌تواند منجر به کاهش دقت در پیش‌بینی کلاس‌های با تعداد کمتر شود و عملکرد مدل روی داده‌های واقعی را ضعیف کند.

تاثیر عدم بالانس بر مدل

وقتی کلاس‌های یک ویژگی نامتعادل هستند، مدل در فرآیند آموزش تمایل دارد وزن بیشتری به داده‌هایی بدهد که تعدادشان بیشتر است. به عنوان مثال، اگر ویژگی Attrition_Flag دو کلاس داشته باشد و یکی از کلاس‌ها ۹۰٪ و دیگری ۱۰٪ داده‌ها را شامل شود، مدل ممکن است صرفاً با پیش‌بینی کلاس غالب (۹۰٪) به دقت کلی بالا دست پیدا کند اما عملاً عملکرد درستی روی کلاس دیگر نداشته باشد.

دلایل اهمیت مدیریت عدم بالانس

- مدل‌ها معمولاً برای افزایش دقت کلی (Accuracy) طراحی شده‌اند و در صورت وجود عدم تعادل، به راحتی Bias می‌شوند.
- در مسائل حساس (مانند تشخیص بیماری یا پیش‌بینی تقلب)، دقت کلاس‌های اقلیت بسیار اهمیت دارد.
- عدم بالانس می‌تواند بر معیارهایی مانند Precision، Recall و F1-Score تأثیر بگذارد و ارزیابی مدل را گمراه‌کننده کند.

راهکارهای مقابله با عدم بالانس

برای حل مشکل عدم بالانس می‌توان از روش‌های زیر استفاده کرد:

۱. تغییر در داده‌ها (Data Level)

- افزایش نمونه‌های کلاس اقلیت (**Oversampling**): از روش‌هایی مانند SMOTE استفاده می‌شود که داده‌های مصنوعی برای کلاس اقلیت تولید می‌کند:

```
    from imblearn.over_sampling import SMOTE
    smote = SMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(X, y)

```

- کاهش نمونه‌های کلاس غالب (**Undersampling**): از داده‌های کلاس غالب کمتر استفاده می‌شود تا تعادل برقرار شود:

```
from imblearn.under_sampling import RandomUnderSampler
rus = RandomUnderSampler(random_state=42)
X_resampled, y_resampled = rus.fit_resample(X, y)

```

۲. تغییر در مدل (Algorithm Level)

- استفاده از الگوریتم‌هایی که وزن بیشتری به کلاس‌های اقلیت می‌دهند، مانند کلاس‌بندی‌کننده‌های وزن‌دار (Weighted Classifiers):

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)

```

- استفاده از متریک‌های جایگزین برای ارزیابی مدل به جای دقت، مانند ROC-AUC یا F1-Score.

۳. تقسیم داده‌ها برای آموزش و ارزیابی

قبل از آموزش، داده‌ها به مجموعه‌های آموزش و آزمون تقسیم می‌شوند تا از تاثیر عدم بالانس روی فرآیند ارزیابی جلوگیری شود:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
    stratify=y)

```

نتیجه‌گیری

در نتیجه، مدیریت عدم بالانس داده یکی از گام‌های کلیدی برای آموزش مدل‌های قوی و بی‌طرف است. بدون این مدیریت، خروجی مدل نمی‌تواند به خوبی نمایانگر عملکرد واقعی باشد و مدل ممکن است در محیط واقعی شکست بخورد.

متعادل‌سازی داده‌ها: قبل یا بعد از تقسیم‌بندی؟

زمانی که بخواهیم از یک الگوریتم برای متعادل کردن مجموعه داده‌ها استفاده کنیم، سوال اصلی این است که این فرآیند باید قبل از تقسیم‌بندی داده‌ها به بخش‌های آموزش و آزمون انجام شود یا پس از آن.

به طور کلی، متعادل‌سازی داده‌ها باید فقط روی داده‌های آموزشی اعمال شود و نه روی کل مجموعه داده یا داده‌های آزمون. به دلایل زیر:

دلایل متعادل‌سازی پس از تقسیم‌بندی

- جلوگیری از نشت داده (**Data Leakage**): اگر متعادل‌سازی روی کل مجموعه داده انجام شود و سپس داده‌ها تقسیم‌بندی شوند، اطلاعات مربوط به داده‌های آزمون ممکن است به داده‌های آموزشی نشت کند. این موضوع منجر به ارزیابی غیرواقعی و نتایج گمراه‌کننده می‌شود.
- بازتاب دقیق دنیای واقعی: در محیط‌های واقعی، داده‌های آزمون (داده‌هایی که مدل روی آن‌ها ارزیابی می‌شود) معمولاً نامتعادل هستند. بنابراین، متعادل‌سازی نباید بر داده‌های آزمون اعمال شود تا عملکرد واقعی مدل روی داده‌های نامتعادل بررسی شود.
- ایجاد مدل قابل اعتماد: متعادل‌سازی تنها باید روی داده‌های آموزشی اعمال شود تا مدل بتواند با درک بهتر از داده‌های اقلیت، تعادل را برقرار کند. سپس این مدل روی داده‌های آزمون ارزیابی شود تا مشخص شود آیا راهکار اعمال شده موثر بوده است یا خیر.

مراحل پیشنهادی

۱. ابتدا مجموعه داده را به دو بخش آموزشی و آزمون تقسیم کنید.

۲. تنها روی داده‌های آموزشی متعادل‌سازی انجام دهید.

۳. مدل را با داده‌های متعادل‌شده آموزش دهید.

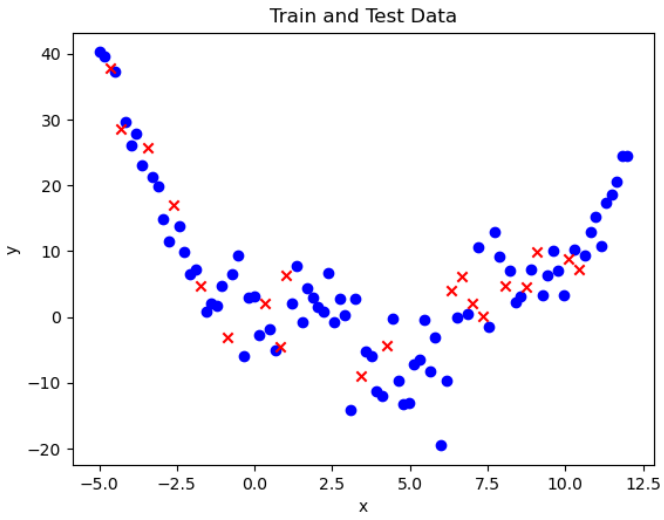
۴. عملکرد مدل را با استفاده از داده‌های آزمون (بدون اعمال متعادل‌سازی) ارزیابی کنید.

نتیجه‌گیری

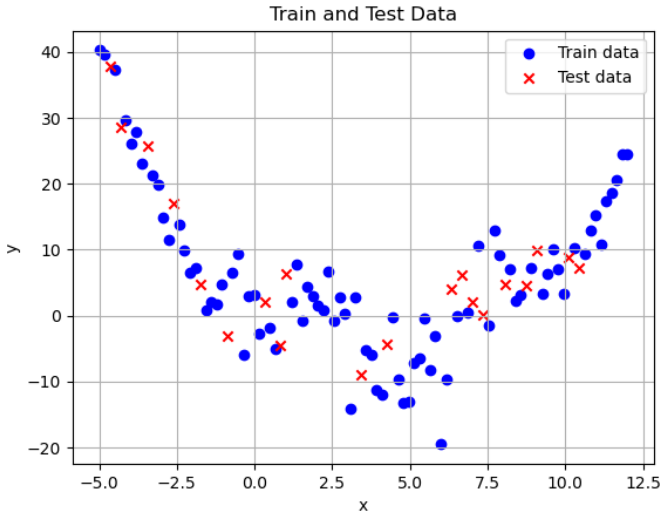
متعادل‌سازی داده‌ها فرآیندی است که باید بعد از تقسیم‌بندی داده‌ها و فقط روی داده‌های آموزشی انجام شود. این روش تضمین می‌کند که مدل به درستی آموزش داده شده و عملکرد واقعی آن روی داده‌های آزمون نامتعادل ارزیابی می‌شود.

۲ پرسش دوم

۱.۲ نمایش داده های ترین و تست



شکل ۴: نمایش داده های train ، test



شکل ۵: نمایش داده های train ، test

۲.۲ معیار برای سنجش عملکرد مدل های رگرسیون

در این بخش به بررسی دو معیار مهم برای سنجش عملکرد مدل‌های رگرسیون می‌پردازیم:

- میانگین خطای مطلق (Mean Absolute Error - MAE):

این معیار مقدار میانگین تفاضل‌های مطلق بین مقادیر واقعی (y_i) و مقادیر پیش‌بینی‌شده ($\hat{y}_i$) را محاسبه می‌کند. فرمول آن به‌صورت زیر است:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

این معیار تفسیر ساده‌ای دارد و نسبت به مقادیر پرت حساسیت کمتری نشان می‌دهد.

- میانگین مربعات خطا (Mean Squared Error - MSE): این معیار مربع اختلافات بین مقادیر واقعی و پیش‌بینی‌شده را محاسبه کرده و میانگین آن‌ها را می‌گیرد. فرمول آن به صورت زیر است:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

این معیار (خطاهای بزرگ را بیشتر جریمه می‌کند) و مناسب برای مواقعی است که نیاز به دقت بالا در پیش‌بینی مقادیر پرت داریم.

- ریشه میانگین مربعات خطا (Root Mean Square Error - RMSE): یکی از معیارهای معروف برای محاسبه خطا، ریشه میانگین مربعات خطا است که به صورت زیر تعریف می‌شود:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

که در آن:

- y_i: مقدار واقعی یا مشاهده‌شده (Value Observed) که نشان‌دهنده مقدار واقعی است که از داده‌ها یا مشاهدات تجربی به‌دست آمده است.
- $\hat{y}_i$: مقدار پیش‌بینی‌شده (Value Predicted) که توسط مدل پیش‌بینی به‌دست آمده است.
- n: تعداد نمونه‌ها که نشان‌دهنده تعداد داده‌ها یا مشاهدات است که برای محاسبه معیار از آن‌ها استفاده می‌شود.

این معیار برای مقادیر پرت حساسیت بیشتری نشان میدهد.

تفاوت و مزایا و معایب معیارهای خطا

۱. میانگین خطای مطلق (Mean Absolute Error - MAE)

- مزایا:

- تفسیر ساده: MAE به‌طور متوسط خطاها را در واحد اصلی داده‌ها اندازه‌گیری می‌کند و تفسیر آن نسبتاً ساده است.
- حساسیت کمتر به مقادیر پرت: MAE به دلیل اینکه خطاها به‌طور مطلق محاسبه می‌شوند، تأثیر زیادی از مقادیر پرت نمی‌گیرد و برای داده‌های با مقادیر پرت زیاد مناسب است.

- معایب:

- عدم توجه به خطاهای بزرگ: MAE به اندازه خطاها اهمیت می‌دهد، اما نمی‌تواند خطاهای بزرگ را بیشتر مجازات کند. بنابراین در مسائلی که نیاز به دقت بالا در پیش‌بینی مقادیر پرت داریم، ممکن است کارایی کمتری داشته باشد.

- استفاده:

- مناسب برای داده‌هایی که مقادیر پرت زیادی ندارند.
- زمانی که نیازی به حساسیت زیاد به خطاهای بزرگ نداریم.
- زمانی که نیاز به تفسیر ساده‌تر و قابل‌فهم‌تر از خطاها داریم.

۲. ریشه میانگین مربعات خطا (RMSE - Root Mean Square Error)

- مزایا:

- حساسیت به مقادیر پرت: RMSE به دلیل اینکه خطاها را به توان ۲ می‌رساند، به مقادیر پرت حساس است. اگر مدل باید توجه بیشتری به خطاهای بزرگ داشته باشد، RMSE می‌تواند مفید باشد.
- تفسیر آسان: مقادیر RMSE مشابه واحد داده‌ها هستند، بنابراین تفسیر آن نسبت به MSE آسان‌تر است.

- معایب:

- حساسیت زیاد به مقادیر پرت: چون خطاها به توان ۲ می‌رسند، حتی مقادیر کوچک پرت می‌توانند تأثیر زیادی روی نتیجه نهایی بگذارند.
- در محیط‌های بدون مقادیر پرت زیاد ممکن است تأثیر نهایی RMSE اغراق‌آمیز باشد.

- استفاده:

- زمانی که به خطاهای بزرگ و مقادیر پرت توجه بیشتری نیاز داریم.
- برای مدل‌هایی که به دقت بالا در پیش‌بینی مقادیر خاص اهمیت دارند.
- مناسب برای داده‌هایی که می‌خواهید خطاهای بزرگ را بیشتر مجازات کنید.

۳. میانگین خطای مربعی (MSE - Mean Squared Error)

- مزایا:

- مجازات خطاهای بزرگ: MSE به دلیل اینکه تفاوت‌ها را به توان ۲ می‌رساند، برای مدل‌هایی که باید خطاهای بزرگ را بیشتر مجازات کنند، مفید است.
- دقیق‌تر بودن: به‌دلیل حساسیت به خطاهای بزرگ، MSE می‌تواند در مدل‌هایی که نیاز به دقت بالا دارند مناسب باشد.

- معایب:

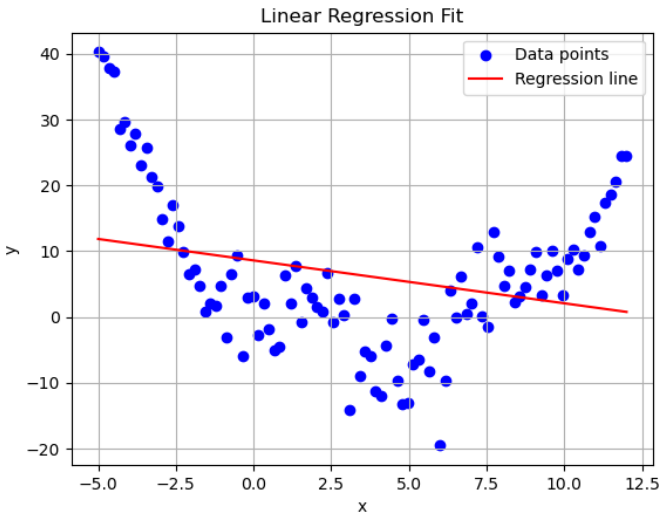
- تفسیر دشوارتر: مقادیر MSE واحدی متفاوت از داده‌ها دارند که می‌تواند تفسیر آن را برای بسیاری از افراد پیچیده کند.
- حساسیت زیاد به مقادیر پرت: مشابه با RMSE، MSE هم به‌خاطر مجازات خطاها به توان ۲ حساسیت زیادی به مقادیر پرت دارد. بنابراین در داده‌های با مقادیر پرت زیاد، می‌تواند به‌طور غیرمنصفانه مدل را تحت تأثیر قرار دهد.

- استفاده:
  - مناسب برای مدل‌هایی که به کاهش خطاهای بزرگ اهمیت دارند.
  - زمانی که به دقت بالا در پیش‌بینی و کم کردن اشتباهات بزرگ نیاز داریم.
  - در مسائل پیچیده‌تر که برای جریمه کردن بیشتر خطاهای بزرگ طراحی شده‌اند.

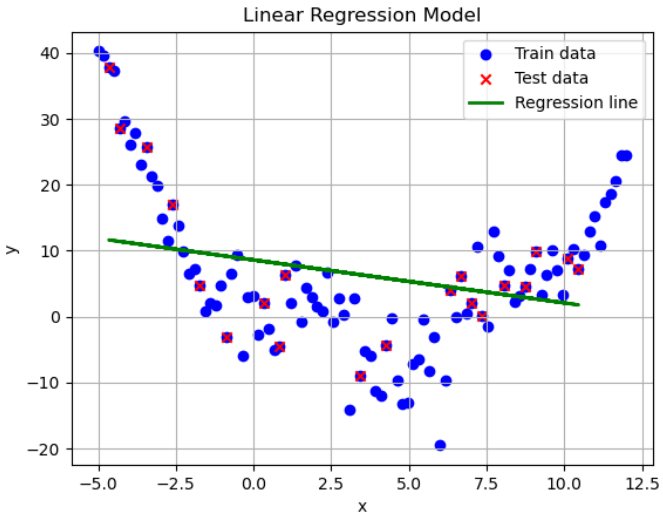
نتیجه‌گیری و انتخاب معیار مناسب:

- MAE: بهترین انتخاب برای داده‌هایی است که مقادیر پرت زیادی ندارند و نیازی به حساسیت زیاد به خطاهای بزرگ نداریم. مناسب برای زمانی است که نیاز به تفسیر ساده و قابل‌فهم داریم.
- RMSE: برای مدل‌هایی که به مقادیر پرت حساسیت بیشتری دارند و می‌خواهیم خطاهای بزرگ را مجازات کنیم، مناسب است.
- MSE: بهترین انتخاب برای زمانی است که می‌خواهیم به‌طور دقیق‌تر و با حساسیت بیشتر به خطاهای بزرگ توجه کنیم، به‌ویژه در مسائلی که خطاهای بزرگ باید بیشتر مجازات شوند.

۳.۲ مدل رگرسیون خطی درجه اول



شکل ۶: پیشبینی مدل با مدل خطی



شکل ۷: پیش‌بینی مدل با دیتا ترین و تست

طبق شکل ۵ داده‌های train و test نمایش داده شده‌اند.

- با توجه به این که داده‌های ما دارای شکل سهمی هستند، استفاده از یک مدل خطی درجه اول (یعنی معادله‌ای از نوع خط مستقیم) نمی‌تواند رفتار این داده‌ها را به خوبی توضیح دهد. در واقع، مدل خطی درجه اول تنها می‌تواند رابطه‌ای خطی بین متغیرهای ورودی و خروجی برقرار کند، در حالی که داده‌های ما به وضوح نشان‌دهنده وجود یک رابطه غیرخطی (از نوع سهمی یا درجه دوم) هستند.

  برای مدل‌سازی صحیح این داده‌ها، حداقل به یک مدل با معادله درجه دوم نیاز داریم که توانایی تشخیص این نوع رابطه غیرخطی را داشته باشد. این معادله می‌تواند شامل جمله‌های مربعی باشد مانند:

$$ax^2 + bx + c$$

  که امکان تطابق دقیق‌تری با داده‌ها را فراهم می‌کند.

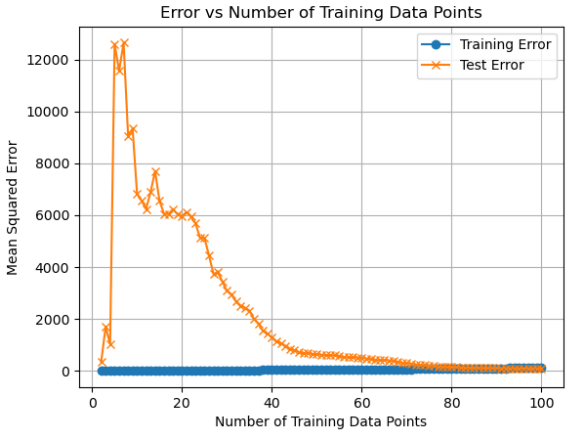
  در نتیجه، پیش‌بینی داده‌ها با یک مدل خطی درجه اول نه تنها ناکافی است، بلکه ممکن است منجر به خطای پیش‌بینی قابل توجه شود. از سوی دیگر، استفاده از یک مدل درجه دوم به دلیل تطابق بهتر با داده‌های سهمی‌شکل، نتایج بهتری ارائه می‌دهد.

  بنابراین، پیشنهاد می‌شود از مدل‌های درجه دوم یا حتی مدل‌های پیشرفته‌تر (در صورت پیچیدگی بیشتر داده‌ها) برای مدل‌سازی استفاده شود تا دقت بیشتری در پیش‌بینی‌ها حاصل شود.

۴.۲ Iteration ثابت

تحلیل روند تغییر خطا با افزایش داده‌های آموزشی

در این بخش به بررسی تغییرات Mean Squared Error (MSE) برای داده‌های آموزش و آزمون با افزایش تعداد داده‌های آموزشی پرداخته می‌شود.



H

شکل ۸: points data traning of number

خطای داده‌های آموزش

- با افزایش تعداد داده‌های آموزشی، مدل به تدریج بهتر می‌تواند داده‌های آموزشی را تطبیق دهد.
- بنابراین، خطای داده‌های آموزشی به طور یکنواخت کاهش می‌یابد.

خطای داده‌های آزمون

- در مراحل اولیه، مدل به دلیل اطلاعات محدود، تعمیم‌پذیری خوبی ندارد و خطای آزمون بالا خواهد بود.
- با افزایش داده‌های آموزشی، مدل توانایی تعمیم‌دهی به داده‌های جدید را بهتر یاد می‌گیرد و خطای آزمون کاهش می‌یابد.
- پس از رسیدن به تعداد کافی از داده‌ها، خطای آزمون ممکن است ثابت بماند یا اندکی افزایش یابد (در صورت بروز Overfitting).

نتیجه‌گیری

- افزایش داده‌های آموزشی به طور کلی به بهبود دقت مدل و کاهش خطای آزمون منجر می‌شود.
- این موضوع نشان می‌دهد که در دسترس بودن داده‌های بیشتر می‌تواند به تعمیم‌پذیری بهتر مدل کمک کند.
- با این حال، انتخاب مناسب مدل و جلوگیری از پیچیدگی بیش‌ازحد نیز از اهمیت ویژه‌ای برخوردار است.

۵.۲ ۵-۲

با افزایش تعداد داده‌های آموزشی، مدل یادگیری ماشین می‌تواند خطای خود را به طور قابل توجهی کاهش دهد. این کاهش به دلایل زیر رخ می‌دهد:

- افزایش دقت مدل: با اضافه شدن داده‌های بیشتر، مدل اطلاعات بیشتری برای یادگیری الگوهای داده دارد و به همین دلیل می‌تواند پیش‌بینی‌های دقیق‌تری انجام دهد.
- کاهش خطای آموزشی: مدل با داده‌های بیشتر می‌تواند به شکلی بهتر به داده‌های آموزشی تطبیق پیدا کند که منجر به کاهش خطای آموزشی می‌شود.
- پایداری خطای تست: با وجود افزایش داده‌های آموزشی، خطای تست معمولاً ابتدا کاهش می‌یابد و سپس ممکن است به مقدار ثابتی برسد، چرا که مدل توانایی تعمیم‌دهی به داده‌های جدید را حفظ می‌کند.

با این حال، اگر داده‌های آموزشی به اندازه کافی متنوع نباشند یا مدل بیش از حد پیچیده باشد، ممکن است مشکل بیش‌برازش (Overfitting) ایجاد شود که باعث افزایش خطای تست می‌شود.

به طور کلی، با افزایش مناسب داده‌های آموزشی، خطای مدل می‌تواند حتی کمتر از خطای انسان کاهش یابد، به شرط آنکه از تعادل بین کاهش خطای آموزشی و حفظ توانایی تعمیم‌دهی مدل اطمینان حاصل شود.

۶.۲ مدل رگرسیون با درجات بالاتر

آیا خطای آزمون همواره کاهش می‌یابد؟

خیر، در ابتدا با افزایش پیچیدگی مدل (افزودن جملات جدید) خطای آزمون کاهش می‌یابد زیرا مدل توانایی بهتری در تطابق با داده‌ها پیدا می‌کند. اما با افزایش بیش از حد پیچیدگی (به عنوان مثال درجات بالاتر چندجمله‌ای)، مدل دچار overfitting می‌شود. در این حالت، مدل بسیار دقیق با داده‌های آموزش تطابق پیدا می‌کند، اما عملکرد ضعیفی روی داده‌های آزمون دارد، و بنابراین خطای آزمون افزایش می‌یابد.

نتیجه‌گیری

افزایش تعداد جملات مدل فقط تا حد مشخصی مفید است. بعد از آن، پیچیدگی اضافی به مدل آسیب می‌زند و باعث افزایش خطای آزمون می‌شود. این مفهوم به عنوان تعادل بین بایاس و واریانس شناخته می‌شود.

۷.۲ الگوریتم های رگرسیون

۱- رگرسیون خطی (**Linear Regression**):

این روش ساده‌ترین نوع رگرسیون است و بر اساس مدل خطی $y = ax + b$ عمل می‌کند. در این الگوریتم، سعی می‌شود خطی پیدا شود که مجموع مربعات خطاها (فاصله نقاط داده تا خط) را به حداقل برساند. این روش برای داده‌هایی که به صورت خطی قابل مدل‌سازی هستند، مناسب است اما برای روابط غیرخطی عملکرد خوبی ندارد.

```
from sklearn.linear_model import LinearRegression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
```

۲- رگرسیون Ridge:

این الگوریتم نسخه‌ای پیشرفته‌تر از رگرسیون خطی است که یک جریمه (penalty) به ضرایب مدل اضافه می‌کند تا از بیش‌برازش (Overfitting) جلوگیری شود. این جریمه به صورت $\lambda \sum w_i^2$ به تابع هزینه اضافه می‌شود، که در آن λ یک پارامتر قابل تنظیم است.

```
from sklearn.linear_model import Ridge
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)
```

۳- رگرسیون Lasso:

مشابه Ridge Regression است، اما جریمه‌ای که اضافه می‌کند به صورت مقدار مطلق ضرایب $\lambda \sum |w_i|$ است. این ویژگی باعث می‌شود برخی ضرایب صفر شوند، به این معنا که این روش می‌تواند به انتخاب متغیرها کمک کند. این الگوریتم برای مواقعی که تعداد متغیرها زیاد است و نیاز به ساده‌سازی مدل داریم، بسیار مناسب است.

```
from sklearn.linear_model import Lasso
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(X_train, y_train)
```

bonus ۸.۲

انواع Regularization

- **L1 Regularization (Lasso):** این روش با اضافه کردن مجموع قدرمطلق ضرایب به تابع هزینه کار می‌کند. فرمول تابع هزینه به شکل زیر تغییر می‌یابد:

$$J(w) = \text{MSE} + \lambda \sum_{i=1}^{n} |w_i|$$

که در آن:

- $J(w)$: تابع هزینه با Regularization.
- MSE: میانگین مربع خطا (Mean Squared Error).
- λ: مقدار Regularization که پیچیدگی مدل را کنترل می‌کند.
- w_i: ضرایب مدل.

- **L2 Regularization (Ridge):** این روش با اضافه کردن مجموع مربعات ضرایب به تابع هزینه عمل می‌کند. فرمول تابع هزینه در این حالت به صورت زیر است:

$$J(w) = \text{MSE} + \lambda \sum_{i=1}^{n} w_i^2$$

این روش ضرایب را کوچک‌تر کرده و از نوسانات شدید در مقادیر آنها جلوگیری می‌کند.

مزایا و معایب **Regularization**

- مزایا:
  - جلوگیری از Overfitting و بهبود تعمیم‌پذیری (Generalization) مدل.
  - کاهش پیچیدگی مدل و ساده‌تر شدن ضرایب.
- معایب:
  - اگر مقدار λ بسیار بزرگ باشد، مدل ممکن است بیش از حد ساده شده و دچار **Underfitting** شود.
  - انتخاب مقدار مناسب λ نیازمند تنظیم دقیق (Hyperparameter Tuning) است.

کاربرد **Regularization** در مدل‌های چندجمله‌ای

در مدل‌های چندجمله‌ای، به دلیل پیچیدگی بالای درجات بالاتر، مدل به راحتی می‌تواند دچار **Overfitting** شود. استفاده از Regularization در این مدل‌ها باعث می‌شود ضرایب مربوط به درجات بالاتر کاهش پیدا کند و تأثیر آنها کمتر شود. این امر منجر به تعادل بین دقت مدل و تعمیم‌پذیری (Generalization) آن می‌شود.

نتیجه‌گیری

Regularization یکی از ابزارهای کلیدی در Machine Learning است که به کمک آن می‌توان از پیچیدگی بیش از حد مدل جلوگیری کرد. انتخاب نوع Regularization (L1 یا L2) و مقدار مناسب λ از اهمیت بالایی برخوردار است، چرا که تعادل بین **Bias** و **Variance** را تعیین می‌کند.