مدل رگرسیون برای پیش بینی اعتبار مالی افراد

پروژه اول درس یادگیری ماشین

استاد درس:

دکتر محمد کیانی

یسنا طالبی – ۴۰۰۳۶۱۳۰۴۵

خرداد ۱۴۰۳

Table of Contents

Preprocessing on the Raw Data

برای بهتر عمل کردن مدل‌های یادگیری ماشین، نیاز است پیش‌پردازش‌های مختلفی روی داده‌ها انجام شود.

Dropping Unnecessary Columns

بعد از خواندن فایل csv میبینیم که داده‌های یکی از ستون‌ها تماما null است. به همین دلیل این ستون و همچنین ستون مربوط به شماره‌ی مشتریان را دراپ میکنیم زیرا این دو ستون تاثیری در عملکرد مدل ما ندارند. چه بسا عملکرد آن را بدتر کنند.

```python
import pandas as pd

df = pd.read_csv('CreditPrediction.csv')

df.info()

# all the data in 'Unnamed: 19' column is zero, so we delete this column
df.drop('Unnamed: 19', axis=1, inplace=True)
df.drop('CLIENTNUM', axis=1, inplace=True)
```

Executed at 2024.05.27 11:48:15 in 125ms

```
 13  Total_Revolving_Bal       10167 non-null  int64
 14  Total_Amt_Chng_Q4_Q1      10167 non-null  float64
 15  Total_Trans_Amt           10167 non-null  int64
 16  Total_Trans_Ct            10167 non-null  int64
 17  Total_Ct_Chng_Q4_Q1       10167 non-null  float64
 18  Avg_Utilization_Ratio     10167 non-null  float64
 19  Unnamed: 19               0 non-null      float64
dtypes: float64(8), int64(7), object(5)
memory usage: 1.6+ MB
```

Filling the NaNs

برای پر کردن NaN ها یک روش این است که کل آن ردیف داده را حذف کنیم اما با استفاده از این روش ممکن است داده‌های ارزشمندی را از دست دهیم. برای جلوگیری از این موضوع، داده‌های جا افتاده ستون‌های Gender و Marital_Status را با مد آن ستون جایگزین کردم. دلیل استفاده از مد این است که این داده ها گسسته و غیرعددی هستند و ممکن است از با میانگین گرفتن موجب ایجاد عدد اعشاری شویم.

برای دو ستون بعد، به دلیل اینکه داده‌ها پیوسته هستند مقادیر جا افتاده را با میانگین جایگزین کردم و در نهایت ردیف داده‌های تکراری را نیز حذف کردم.

```
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])
df['Marital_Status'] = df['Marital_Status'].fillna(df['Marital_Status'].mode()[0])
df['Card_Category'] = df['Card_Category'].fillna(df['Card_Category'].mode()[0])
df['Months_on_book'] = df['Months_on_book'].fillna(df['Months_on_book'].mean().round())
df['Total_Relationship_Count'] = df['Total_Relationship_Count'].fillna(df['Total_Relationship_Count'].mean()
 .round())

df = df.drop_duplicates()
df.info()
```

Executed at 2024.05.27 11:55:19 in 159ms

```
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Customer_Age            10128 non-null  float64
 1   Gender                  10128 non-null  object
 2   Dependent_count         10128 non-null  int64
 3   Education_Level         10128 non-null  object
 4   Marital_Status          10128 non-null  object
 5   Income_Category         10128 non-null  object
 6   Card_Category           10128 non-null  object
 7   Months_on_book          10128 non-null  float64
```

Converting Alphabetical Data to Numeric Data

برای استفاده از داده‌ها نیاز داریم تمام داده‌ها بصورت عددی باشند. برای تبدیل داده‌های حروفی به داده‌های عددی، از روش One-Hot استفاده شده است. مزیت این روش نسبت به روش‌هایی مثل mapping این است که وقتی داده‌های ما ترتیب خاصی ندارند، این روش بدون اینکه ترتیبی قائل شود داده‌ها را عددی میکند.

در اینجا مقدار Unknown که در بعضی از ستون‌ها وجود داشت نیز برابر NaN درنظر گرفته شده و مقدار آن با مد هر ستون جایگزین شده است.

```
18 df['Marital_Status'] = df['Marital_Status'].replace("Unknown", df['Marital_Status'].mode()[0])
19 df_encoded = pd.get_dummies(df['Marital_Status'], prefix='Marital_Status')
20 df_encoded = df_encoded.astype(int)
21 df = pd.concat([df, df_encoded], axis=1)
22 df.drop('Marital_Status', axis=1, inplace=True)
23
24 df_encoded = pd.get_dummies(df['Card_Category'], prefix='Card_Category')
25 df_encoded = df_encoded.astype(int)
26 df = pd.concat([df, df_encoded], axis=1)
27 df.drop('Card_Category', axis=1, inplace=True)
28
29 df.info()
```

Executed at 2024.05.27 12:22:35 in 142ms

```
22 Income_Category_$60K - $80K     10128 non-null  int32
23 Income_Category_$80K - $120K    10128 non-null  int32
24 Income_Category_Less than $40K  10128 non-null  int32
25 Marital_Status_Divorced         10128 non-null  int32
26 Marital_Status_Married          10128 non-null  int32
27 Marital_Status_Single           10128 non-null  int32
28 Card_Category_Blue              10128 non-null  int32
29 Card_Category_Gold              10128 non-null  int32
30 Card_Category_Platinum          10128 non-null  int32
31 Card_Category_Silver            10128 non-null  int32
```

Handling Outlier Data

برای هندل کردن داده‌های پرت یک رویکرد این است که کل ردیف حاوی داده‌ی پرت را حذف کنیم که این باعث از دست رفتن اطلاعات میشود. رویکرد دیگر این است که داده‌های پرت را تشخیص داده و آنها را با یک مقدار معتبر جایگزین کنیم. در اینجا رویکرد دوم با استفاده از الگوریتم IQR پیاده شده که داده‌های پرت توسط IQR تشخیص داده شده و سپس این مقادیر با میانگین هر ستون جایگزین شده است.

این روش که یک معیار پراکندگی آماری است، نشان دهنده‌ی محدوده‌ای است که ۵۰٪ میانی داده ها در آن قرار دارند. این مقدار به صورت اختلاف بین چارک سوم و چارک اول محاسبه میشود. این الگوریتم نسبت به الگوریتم Z-score کمتر تحت تاثیر داده‌های پرت قرار میگیرد زیرا فقط ۵۰٪ داده‌های میانی را در نظر میگیرد. در صورتیکه Z-score تحت تاثیر تمام داده‌ها، از جمله داده‌های پرت، قرار میگیرد.

دلیل دیگری که موجب استفاده از IQR میشود این است که این الگوریتم برای داده‌هایی که از نوع توزیع آنها اطلاعی نداریم مناسب است و برای داده داده‌هایی که توزیع نرمال دارند میتوانیم از الگوریتم های گاوسی استفاده کنیم.

در نهایت هم الگوریتم IQR و هم Z-score ممکن است داده‌های چالشی را به عنوان داده‌های پرت درنظر بگیرند و هیچ تضمینی در این مورد وجود ندارد.

```python
# Define a function to replace outliers with the mode of each column using IQR
def replace_outliers_with_mode_iqr(data):
    # Initialize an empty DataFrame to store the results
    result = pd.DataFrame(index=data.index, columns=data.columns)

    # Loop through each column in the DataFrame
    for col in data.columns:
        Q1 = data[col].quantile(0.25)  # 25th percentile (Q1)
        Q3 = data[col].quantile(0.75)  # 75th percentile (Q3)
        IQR = Q3 - Q1  # Interquartile Range (IQR)
        lower_bound = Q1 - 1.5 * IQR  # Lower bound for outliers
        upper_bound = Q3 + 1.5 * IQR  # Upper bound for outliers

        # Replace outliers with mode for the current column
        mode_val = data.loc[(data[col] >= lower_bound) & (data[col] <= upper_bound), col].mean()
        result[col] = data[col].apply(lambda x: mode_val if x < lower_bound or x > upper_bound else x)

    return result

# Apply the function to replace outliers with mode for each column separately
df = replace_outliers_with_mode_iqr(df)
df['Customer_Age'].describe()
```

در اینجا همانطور که مشاهده می‌کنید، در جدول سمت چپ که خروجی قسمت قبل است ماکسیمم سن مشتریان ۳۵۲ است که برای سن این یک عدد معتبر نیست.

بعد از اعمال الگوریتم IQR و جایگزین کردن داده‌های پرت، در جدول سمت راست میبینیم که بدون حذف هیچ داده‌ای، ماکسیمم سن مشتریان به ۶۸ رسیده است و دیگر داده‌ی پرتی وجود ندارد.

	Customer_Age
count	10128.000000
mean	46.759188
std	13.540138
min	26.000000
25%	41.000000
50%	46.000000
75%	52.000000
max	352.330517

	Customer_Age
count	10128.000000
mean	46.307859
std	8.004431
min	26.000000
25%	41.000000
50%	46.000000
75%	52.000000
max	68.000000

Normalizing Data

برای نرمال کردن داده‌ها نیاز داریم که اول داده‌های train و test را جدا کنیم زیرا عمل scale کردن باید فقط روی داده‌های train انجام شود و بعد روی داده‌های test منتقل شود. این کار به این دلیل است که ما اجازه نداریم داده‌های test را نگاه کنیم. بر روی Y ها و label ها هیچ عملیاتی انجام نمی‌شود.

برای scale کردن داده ها از StandardScaler استفاده شده است زیرا با استفاده از این روش تمام داده‌ها بین ۳- و ۳ قرار میگیرند و مقایسه‌ی ویژگی‌های مختلف را آسان‌تر می‌سازد.

```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Initialize StandardScaler
scaler = StandardScaler()

Y = df['Credit_Limit']
df.drop('Credit_Limit', axis=1, inplace=True)

X = df

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, shuffle=True)

# Apply Standard scaling to the DataFrame
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)

X_train['Customer_Age'].describe()
```

	Customer_Age
count	7.596000e+03
mean	1.066375e-16
std	1.000066e+00
min	-2.537400e+00
25%	-6.661895e-01
50%	-4.245285e-02
75%	7.060311e-01
max	2.701988e+00

بعد از scale کردن داده‌ها میبینیم که تمام داده‌ها بین ۳- و ۳ قرار گرفته‌اند.

Regression Models

مدل‌های استفاده شده در این پروژه شامل : Linear Regression ، Polynomial Regression ، Gradient Boosting و Random Forest است که بر اساس بدترین دقت تا بهترین دقت شرح داده می‌شوند.

Linear Regression

مدل رگرسیون خطی به دلیل ساده بود و در نظر گرفتن روابط بصورت خطی، برای مسائلی که شامل پیچیدگی‌هایی هستند و به عوامل مختلفی بستگی دارند ممکن است دقت خوبی در مقایسه با مدل‌های دیگر نداشته باشد. همانطور که مشاهده میکنید MSE این مدل در حدود ۱۵ میلیون است و در بین مدل‌های دیگر خطای نسبتا زیادی داشته است.

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

linear_regression = LinearRegression()

linear_regression.fit(X_train, Y_train)

Y_pred_LR = linear_regression.predict(X_test)

mse_LR = mean_squared_error(Y_test, Y_pred_LR)

print("MSE : ", mse_LR)
```

Executed at 2024.05.27 17:27:56 in 106ms

```
MSE :  15256508.975096827
```

Polynomial Regression with Ridge Regression

رگرسیون چندجمله‌ای میتواند روابط غیرخطی را با تبدیل کردن ویژگی‌های اصلی به چند جمله‌ای های درجه بالاتر مدل سازی کند. الگوریتم Ridge نیز با جلوگیری از صرف تمرکز روی MSE باعث میشود تا از Overfitting جلوگیری شود. با تمام این ویژگی‌ها، این الگوریتم در درجه‌های بالا و ویژگی‌های زیاد از نظر محاسبات بسیار هزینه‌بر است. این الگوریتم با اینکه روابط غیرخطی را شناسایی میکند اما ممکن است از الگوریتم‌هایی که ساختار پیچیده‌تری دارند، دقت کمتری داشته باشد.

در اینجا نیز مشاهده میکنید که MSE این الگوریتم حدودا ۱۴ میلیون است که ۱ میلیون از MSE رگرسیون خطی کمتر است اما به نسبت الگوریتم‌های پیچیده‌تر دقت خیلی خوبی ندارد.

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge

poly = PolynomialFeatures(degree=2)

X_train_poly = poly.fit_transform(X_train)

X_test_poly = poly.transform(X_test)

ridge = Ridge()
ridge.fit(X_train_poly, Y_train)

Y_pred_poly = ridge.predict(X_test_poly)

mse_poly = mean_squared_error(Y_test, Y_pred_poly)

print(mse_poly)
```

Executed at 2024.05.27 17:27:57 in 216ms

```
13969406.734455178
```

Gradient Boosting

این الگوریتم چندین مدل را برای تشکیل یک پیش‌بینی ترکیب میکند و بر اصلاح خطاهای مدل‌های قبلی تمرکز دارد. . این مدل میتواند روابط و تعملات پیچیده در داده‌ها را مدل سازی کند، که این مورد این الگوریتم را برای پیش بینی اعتبار مالی مناسب میسازد اما با اینکه برای انواع داده‌ها میتواند خیلی خوب عمل کند ، ممکن است در دام Overfitting بیوفتد. این مدل اغلب به دلیل افزایش خطر Overfitting کمی ضعیف‌تر از Random Forest عمل میکند.

در کد زیر میبینید که MSE این مدل ۹ میلیون است که بسیار بهتر از مدل‌های قبلی است.

```
1  from sklearn.ensemble import GradientBoostingRegressor
2
3  gb = GradientBoostingRegressor()
4
5  gb.fit(X_train, Y_train)
6
7  Y_pred_gb = gb.predict(X_test)
8
9  mse_gb = mean_squared_error(Y_test, Y_pred_gb)
10
11 print("MSE : ", mse_gb)
```

Executed at 2024.05.27 17:27:59 in 2s 390ms

```
MSE :  9104653.34039725
```

Random Forest

این مدل چندین درخت تصمیم‌گیری را برای بهبود دقت پیش‌بینی و کنترل Overfitting ترکیب میکند. مدل Random Forest با تعداد زیادی ویژگی به خوبی کار میکند و میتواند روابط پیچیده در داده‌ها را شناسایی کند. به همین دلیل این مدل، میتواند یک مدل مناسب برای تحلیل داده‌های اعتبار مالی باشد. تنها نقاط ضعف این مدل این است که در ابعاد زیاد، به منابع محاسباتی زیادی نیاز دارد و درصورت تنظیم نادرست ممکن است منجر به Overfitting شود.

در این مدل از ۱۰۰ درخت جست‌و جو استفاده شده است (n_estimatores = 100) و در هر درخت، ماکسیمم عمقی که چک میشود برابر ۱۵ است. (max_depth = 15)

این مقادیر (که با تست‌های فراوان نتیجه گرفته شده‌اند) بهترین جواب را برای مدل Random Forest تولید می‌کنند و همزمان با این، هزینه‌ی سخت افزاری زیادی ایجاد نمی‌کنند.

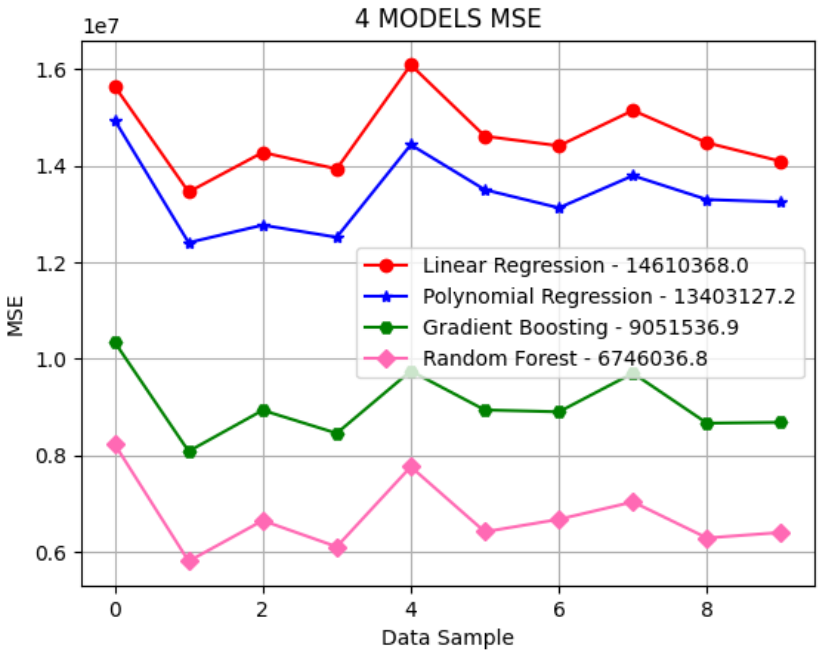
همانطور که مشاهده میکنید MSE این مدل به کمتر از ۶ میلیون رسیده که این ثابت می‌کند مدل Random Forest در بین این چهار مدل بهترین نتیجه را میدهد.

```python
from sklearn.ensemble import RandomForestRegressor

random_forest = RandomForestRegressor(n_estimators=100, max_depth=15)

random_forest.fit(X_train, Y_train)

Y_pred_rf = random_forest.predict(X_test)

mse_rf = mean_squared_error(Y_test, Y_pred_rf)

print("MSE : ", mse_rf)
```

Executed at 2024.05.27 21:57:17 in 12s 612ms

```
MSE :  5846240.742473985
```

Comparison Between 4 Models



با ۱۰ بار اجرا شدن هر چهار مدل با داده‌های train و test متفاوت، نموداری رسم شده که در بالا آن را مشاهده می‌کنید. همانطور که در برچسب نمودار آمده است میانگین MSE مدل Random Forest از سه مدل دیگر کمتر و برابر ۶.۷ میلیون است و در هر دور از ده دور اجرا، بهترین عملکرد را به نسبت سایر مدل‌ها داشته است.

در جایگاه دوم مدل Gradient Boosting است که میانگین MSE برابر ۹ میلیون دارد.

در جایگاه سوم مدل Polynomial Regression است که میانگین MSE آن برابر ۱۳ میلیون است.

ضعیف‌ترین مدل هم Linear Regression با میانگین MSE برابر ۱۴.۶ میلیون بوده است.

مورد دوم این نمودار بررسی میزان پایداری مدل‌ها است. همانطور که مشاهده می‌کنید هر چهار مدل نسبتا پایدار هستند و نهایتا در حدود ۲ میلیون نوسان MSE داشته‌اند که بستگی به این داشته که چه مقدار داده‌ی چالشی در دسته‌ی test آمده است.

Challenges & Suggestions

برای رسیدن به بهترین عملکرد، احتمالا استفاده از الگوریتم‌های یادگیری عمیق بهترین گزینه باشد. اما هر بهترینی هزینه دارد! اگر مسئله به حدی پیچیده باشد که با الگوریتم‌های یادگیری ماشین به نتیجه‌ی خوبی نمیرسد و از نظر تجهیزاتی مانند CPU و GPU مشکلی نداریم (که داریم :)) احتمالا با استفاده از الگوریتم‌های یادگیری عمیق به نتیجه‌ی قابل قبولی خواهیم رسید.