binary-classification-xgboost-1

November 15, 2024

0.1 Preprocessing the Train Data

[29]:
```
# This Python 3 environment comes with many helpful analytics libraries
↪installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list
↪all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
↪gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved
↪outside of the current session
```

```
/kaggle/input/playground-series-s4e7/sample_submission.csv
/kaggle/input/playground-series-s4e7/train.csv
/kaggle/input/playground-series-s4e7/test.csv
```

[30]:
```
df = pd.read_csv("/kaggle/input/playground-series-s4e7/train.csv")
df.head()
```

[30]:
```
   id  Gender  Age  Driving_License  Region_Code  Previously_Insured  \
0   0    Male   21                1         35.0                   0
1   1    Male   43                1         28.0                   0
2   2  Female   25                1         14.0                   1
3   3  Female   35                1          1.0                   0
4   4  Female   36                1         15.0                   1
```

```
  Vehicle_Age Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  Vintage  \
0    1-2 Year            Yes         65101.0                 124.0      187
1   > 2 Years            Yes         58911.0                  26.0      288
2    < 1 Year             No         38043.0                 152.0      254
3    1-2 Year            Yes          2630.0                 156.0       76
4    1-2 Year             No         31951.0                 152.0      294

   Response
0         0
1         1
2         0
3         0
4         0
```

0.1.1 Checking the null values in the columns

[31]:
```
df.isnull().sum()
```

[31]:
```
id                      0
Gender                  0
Age                     0
Driving_License         0
Region_Code             0
Previously_Insured      0
Vehicle_Age             0
Vehicle_Damage          0
Annual_Premium          0
Policy_Sales_Channel    0
Vintage                 0
Response                0
dtype: int64
```

0.1.2 Unique Value count

[32]:
```
df.nunique()
```

[32]:
```
id                      11504798
Gender                         2
Age                           66
Driving_License                2
Region_Code                   54
Previously_Insured             2
Vehicle_Age                    3
Vehicle_Damage                 2
Annual_Premium             51728
```

```
Policy_Sales_Channel         152
Vintage                      290
Response                       2
dtype: int64
```

0.1.3 Description of the Dataset

[33]:
```python
df.describe()
```

[33]:
```
                 id           Age  Driving_License   Region_Code  \
count  1.150480e+07  1.150480e+07     1.150480e+07  1.150480e+07
mean   5.752398e+06  3.838356e+01     9.980220e-01  2.641869e+01
std    3.321149e+06  1.499346e+01     4.443120e-02  1.299159e+01
min    0.000000e+00  2.000000e+01     0.000000e+00  0.000000e+00
25%    2.876199e+06  2.400000e+01     1.000000e+00  1.500000e+01
50%    5.752398e+06  3.600000e+01     1.000000e+00  2.800000e+01
75%    8.628598e+06  4.900000e+01     1.000000e+00  3.500000e+01
max    1.150480e+07  8.500000e+01     1.000000e+00  5.200000e+01

       Previously_Insured  Annual_Premium  Policy_Sales_Channel       Vintage  \
count        1.150480e+07    1.150480e+07          1.150480e+07  1.150480e+07
mean         4.629966e-01    3.046137e+04          1.124254e+02  1.638977e+02
std          4.986289e-01    1.645475e+04          5.403571e+01  7.997953e+01
min          0.000000e+00    2.630000e+03          1.000000e+00  1.000000e+01
25%          0.000000e+00    2.527700e+04          2.900000e+01  9.900000e+01
50%          0.000000e+00    3.182400e+04          1.510000e+02  1.660000e+02
75%          1.000000e+00    3.945100e+04          1.520000e+02  2.320000e+02
max          1.000000e+00    5.401650e+05          1.630000e+02  2.990000e+02

           Response
count  1.150480e+07
mean   1.229973e-01
std    3.284341e-01
min    0.000000e+00
25%    0.000000e+00
50%    0.000000e+00
75%    0.000000e+00
max    1.000000e+00
```

[34]:
```python
columns_to_drop = [ 'Region_Code']

# Using the `drop` method
df_new = df.drop(columns=columns_to_drop)
```

[35]:
```python
df_new
```

```
[35]:                 id  Gender  Age  Driving_License  Previously_Insured  \
0                0    Male   21                1                   0
1                1    Male   43                1                   0
2                2  Female   25                1                   1
3                3  Female   35                1                   0
4                4  Female   36                1                   1
…              …     …  …               …                 …
11504793  11504793    Male   48                1                   0
11504794  11504794  Female   26                1                   0
11504795  11504795  Female   29                1                   1
11504796  11504796  Female   51                1                   0
11504797  11504797    Male   25                1                   1

         Vehicle_Age Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  \
0           1-2 Year            Yes         65101.0                 124.0
1          > 2 Years            Yes         58911.0                  26.0
2           < 1 Year             No         38043.0                 152.0
3           1-2 Year            Yes          2630.0                 156.0
4           1-2 Year             No         31951.0                 152.0
…                …            …             …                   …
11504793    1-2 Year            Yes         27412.0                  26.0
11504794    < 1 Year            Yes         29509.0                 152.0
11504795    < 1 Year             No          2630.0                 152.0
11504796    1-2 Year            Yes         48443.0                  26.0
11504797    < 1 Year             No         32855.0                 152.0

          Vintage  Response
0             187         0
1             288         1
2             254         0
3              76         0
4             294         0
…             …       …
11504793      218         0
11504794      115         1
11504795      189         0
11504796      274         1
11504797      189         0

[11504798 rows x 11 columns]
```

0.2 Binning Age and Annual Premium

**** To bin the Age and Annual_Premium columns in the dataset, we first define the respective bins. For Age, we set the bins as [10, 20, 30, 40, 50, 60, 70, 80, 90], and for Annual_Premium, the bins are [0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000]. Using the pd.cut function, we categorize the Age data into these specified bins with labels representing the age ranges, and similarly, categorize the Annual_Premium data into the premium bins with numerical labels. After binning the Age column, we drop the original Age column from the DataFrame to avoid redundancy. The same binning process is applied to the Annual_Premium column. This procedure ensures that the age and premium data are properly categorized into meaningful intervals, facilitating subsequent analysis or modeling.****

[36]:
```python
age_bins = [10, 20, 30, 40, 50, 60, 70, 80, 90]


# Apply binning using cut
df_new['Age_Binned'] = pd.cut(df_new['Age'], bins=age_bins,␣
 ↪labels=[f'{age}-{age+10}' for age in age_bins[:-1]])

df_new.drop(columns=['Age'], inplace=True)
```

[37]:
```python
bins = [0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000,␣
 ↪100000]
labels = range(len(bins)-1)

df_new['Annual_Premium_Binned'] = pd.cut(df_new['Annual_Premium'], bins=bins,␣
 ↪labels=labels)
```

[38]:
```python
df_new
```

[38]:
```
                id  Gender  Driving_License  Previously_Insured Vehicle_Age  \
0                0    Male                1                   0    1-2 Year
1                1    Male                1                   0   > 2 Years
2                2  Female                1                   1    < 1 Year
3                3  Female                1                   0    1-2 Year
4                4  Female                1                   1    1-2 Year
…              …       …               …                  …          …
11504793  11504793    Male                1                   0    1-2 Year
11504794  11504794  Female                1                   0    < 1 Year
11504795  11504795  Female                1                   1    < 1 Year
11504796  11504796  Female                1                   0    1-2 Year
11504797  11504797    Male                1                   1    < 1 Year

         Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  Vintage  \
0                   Yes         65101.0                 124.0      187
1                   Yes         58911.0                  26.0      288
2                    No         38043.0                 152.0      254
3                   Yes          2630.0                 156.0       76
4                    No         31951.0                 152.0      294
…                   …             …                    …      …
11504793            Yes         27412.0                  26.0      218
11504794            Yes         29509.0                 152.0      115
11504795             No          2630.0                 152.0      189
```

```
11504796            Yes         48443.0                   26.0       274
11504797             No         32855.0                  152.0       189

          Response Age_Binned Annual_Premium_Binned
0                0      20-30                     6
1                1      40-50                     5
2                0      20-30                     3
3                0      30-40                     0
4                0      30-40                     3
…             …        …                   …
11504793         0      40-50                     2
11504794         1      20-30                     2
11504795         0      20-30                     0
11504796         1      50-60                     4
11504797         0      20-30                     3

[11504798 rows x 12 columns]
```

[39]:
```python
df_new.isnull().sum()
```

[39]:
```
id                          0
Gender                      0
Driving_License             0
Previously_Insured          0
Vehicle_Age                 0
Vehicle_Damage              0
Annual_Premium              0
Policy_Sales_Channel        0
Vintage                     0
Response                    0
Age_Binned                  0
Annual_Premium_Binned    2307
dtype: int64
```

[40]:
```python
rand_samp = df_new.sample(n=10)

rand_samp
```

[40]:
```
                id  Gender  Driving_License  Previously_Insured Vehicle_Age  \
8174007    8174007  Female                1                   1    < 1 Year
2246841    2246841    Male                1                   1    1-2 Year
10180753  10180753    Male                1                   0    1-2 Year
7944036    7944036    Male                1                   0    < 1 Year
6829332    6829332  Female                1                   0    1-2 Year
2770997    2770997    Male                1                   1    1-2 Year
4141434    4141434  Female                1                   1    1-2 Year
6541288    6541288  Female                1                   0    1-2 Year
```

```
7021528    7021528  Female                1                   1    1-2 Year
601478      601478  Female                1                   0    1-2 Year

         Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  Vintage  \
8174007              No         23595.0                 152.0      213
2246841              No         28578.0                 124.0       76
10180753            Yes          2630.0                 157.0      299
7944036             Yes         20229.0                 124.0      257
6829332             Yes         31875.0                 124.0      241
2770997              No          2630.0                  26.0      175
4141434              No         24002.0                 152.0      185
6541288             Yes         29364.0                 124.0      223
7021528              No         52639.0                  26.0      187
601478              Yes         29026.0                 124.0      182

          Response Age_Binned Annual_Premium_Binned
8174007          0      30-40                     2
2246841          0      40-50                     2
10180753         1      30-40                     0
7944036          0      20-30                     2
6829332          1      30-40                     3
2770997          0      30-40                     0
4141434          0      30-40                     2
6541288          1      30-40                     2
7021528          0      50-60                     5
601478           0      30-40                     2
```

0.3 Label Encoding

- **Import Libraries**:
  - We import pandas for data manipulation and LabelEncoder from sklearn.preprocessing for label encoding.
- **Define LabelEncoder**:
  - An instance of LabelEncoder is created to perform the encoding.
- **Columns to Encode**:
  - We specify the list `columns_to_encode` containing names of categorical columns ('Gender', 'Vehicle_Age', 'Vehicle_Damage', 'Age_Binned') in `df_new` that we want to encode.
- **Apply Label Encoding**:
  - Using a loop, we iterate through each column in `columns_to_encode` and apply `fit_transform` method of `LabelEncoder` to convert each categorical column into numerical labels.

[41]:
```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Assuming df is your DataFrame
```

```python
label_encoder = LabelEncoder()

# Columns to encode
columns_to_encode = ['Gender', 'Vehicle_Age', 'Vehicle_Damage', 'Age_Binned']

# Apply label encoding to each column
for column in columns_to_encode:
    df_new[column] = label_encoder.fit_transform(df_new[column])
```

[42]:

```python
df_new.drop(columns=['Annual_Premium'], inplace=True)
df_new
```

[42]:

```
                id  Gender  Driving_License  Previously_Insured  Vehicle_Age  \
0                0       1                1                   0            0
1                1       1                1                   0            2
2                2       0                1                   1            1
3                3       0                1                   0            0
4                4       0                1                   1            0
…              …     …               …                 …          …
11504793  11504793       1                1                   0            0
11504794  11504794       0                1                   0            1
11504795  11504795       0                1                   1            1
11504796  11504796       0                1                   0            0
11504797  11504797       1                1                   1            1

          Vehicle_Damage  Policy_Sales_Channel  Vintage  Response  Age_Binned  \
0                      1                 124.0      187         0           1
1                      1                  26.0      288         1           3
2                      0                 152.0      254         0           1
3                      1                 156.0       76         0           2
4                      0                 152.0      294         0           2
…                    …                   …      …       …         …
11504793               1                  26.0      218         0           3
11504794               1                 152.0      115         1           1
11504795               0                 152.0      189         0           1
11504796               1                  26.0      274         1           4
11504797               0                 152.0      189         0           1

          Annual_Premium_Binned
0                             6
1                             5
2                             3
3                             0
4                             3
…                           …
11504793                      2
11504794                      2
```

```
11504795                  0
11504796                  4
11504797                  3

[11504798 rows x 11 columns]
```

[43]:
```
df_new.isnull().sum()
```

[43]:
```
id                        0
Gender                    0
Driving_License           0
Previously_Insured        0
Vehicle_Age               0
Vehicle_Damage            0
Policy_Sales_Channel      0
Vintage                   0
Response                  0
Age_Binned                0
Annual_Premium_Binned  2307
dtype: int64
```

[44]:
```
df_new.dtypes
```

[44]:
```
id                          int64
Gender                      int64
Driving_License             int64
Previously_Insured          int64
Vehicle_Age                 int64
Vehicle_Damage              int64
Policy_Sales_Channel      float64
Vintage                     int64
Response                    int64
Age_Binned                  int64
Annual_Premium_Binned    category
dtype: object
```

1 Handling Missing Values in 'Annual_Premium_Binned'

To handle missing values in the 'Annual_Premium_Binned' column of `df_new`, we first determine the mode (most frequent value) of the column using `df_new['Annual_Premium_Binned'].mode()[0]`. Next, we fill the missing values in the column with this mode value using `fillna(mode_value, inplace=True)`. Finally, to ensure the column contains integer values, we convert it to integer type using `astype(int)`.

[45]:
```
mode_value = df_new['Annual_Premium_Binned'].mode()[0]
df_new['Annual_Premium_Binned'].fillna(mode_value, inplace=True)
```

```python
# Ensure the column is integer type
df_new['Annual_Premium_Binned'] = df_new['Annual_Premium_Binned'].astype(int)
```

```
/tmp/ipykernel_13/2197081812.py:2: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df_new['Annual_Premium_Binned'].fillna(mode_value, inplace=True)
```

[46]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/site-
packages (2.1.0)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.10/site-packages (from xgboost) (2.20.5)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/site-packages
(from xgboost) (1.26.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/site-packages
(from xgboost) (1.13.1)
WARNING: Running pip as the 'root' user can result in broken permissions

and conflicting behaviour with the system package manager. It is recommended to

use a virtual environment instead: https://pip.pypa.io/warnings/venv


[notice] A new release of pip is
available: 23.0.1 -> 24.1.2
[notice] To update, run:
pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

2 XGBoost Workflow Overview

XGBoost, or eXtreme Gradient Boosting, sequentially enhances weak models like decision trees to minimize errors and improve predictions. By combining their outputs, it creates a powerful ensemble model that excels in accuracy and efficiency, making it widely used across diverse fields for its superior performance in complex data analysis tasks.

2.0.1 Steps:

Parameter Distributions:

- Define `param_dist` to specify the range of values for hyperparameters (`n_estimators`, `max_depth`, `learning_rate`, `gamma`, `reg_alpha`, `reg_lambda`) used in RandomizedSearchCV.

XGBoost Model Initialization:

- Initialize an `XGBClassifier` with a set `random_state` for reproducibility.

RandomizedSearchCV Setup:

- Configure `RandomizedSearchCV` to perform hyperparameter tuning with `n_iter=50` iterations, 5-fold cross-validation (`cv=5`), and using accuracy (`scoring='accuracy'`) as the evaluation metric.

Model Fitting:

- Fit `random_search` to the training data (`X_train`, `y_train`) to explore different hyperparameter combinations and identify the best-performing model.

Best Model Evaluation:

- Retrieve the best parameters (`best_params`) and best model (`best_model`) identified by `RandomizedSearchCV`.
- Make predictions on the test set (`X_test`) using `best_model`.
- Evaluate the model's performance using accuracy score, confusion matrix, and classification report.

This approach optimizes the XGBoost model's hyperparameters to achieve higher accuracy and robustness on unseen data, leveraging randomized search and cross-validation for effective model tuning.

[48]:
```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint, uniform

# Define the parameter distributions for RandomizedSearchCV
param_dist = {
    'n_estimators': randint(50, 200),  # Random integer values between 50 and␣
 ↪200
    'max_depth': randint(3, 8),  # Random integer values between 3 and 7
    'learning_rate': uniform(0.001, 0.1),  # Random float values between 0.001␣
 ↪and 0.1
    'gamma': uniform(0, 0.5),  # Random float values between 0 and 0.5
    'reg_alpha': uniform(0, 0.5),  # Random float values between 0 and 0.5
    'reg_lambda': uniform(1, 2),  # Random float values between 1 and 2
}

# Initialize the XGBoost model
xgb_model = XGBClassifier(random_state=42)
```

```
# Perform RandomizedSearchCV with cross-validation
random_search = RandomizedSearchCV(estimator=xgb_model,␣
↪param_distributions=param_dist,
                                   n_iter=50,  # Number of parameter settings␣
↪that are sampled
                                   cv=5,  # 5-fold cross-validation
                                   scoring='accuracy',  # Use accuracy for␣
↪scoring
                                   random_state=42,  # Random state for␣
↪reproducibility
                                   n_jobs=-1,  # Use all available cores
                                   verbose=1)  # Print detailed messages

# Fit the RandomizedSearchCV object to the training data
random_search.fit(X_train, y_train)

# Get the best parameters and model
best_params = random_search.best_params_
best_model = random_search.best_estimator_

# Print the best parameters found
print("Best Parameters:", best_params)

# Make predictions with the best model
y_pred = best_model.predict(X_test)

# Evaluate the best model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print evaluation results
print(f'Accuracy: {accuracy:.2f}')
print('Confusion Matrix:')
print(conf_matrix)
print('Classification Report:')
print(class_report)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
```

```
/usr/local/lib/python3.10/site-
packages/joblib/externals/loky/process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  warnings.warn(
```

```
Best Parameters: {'gamma': 0.28163778598819184, 'learning_rate':
```

```
0.07055160864261276, 'max_depth': 7, 'n_estimators': 192, 'reg_alpha':
0.37777556927152434, 'reg_lambda': 1.457596330983245}
Accuracy: 0.88
Confusion Matrix:
[[2007659   10117]
 [ 268842   14342]]
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.99      0.94   2017776
           1       0.59      0.05      0.09    283184

    accuracy                           0.88   2300960
   macro avg       0.73      0.52      0.51   2300960
weighted avg       0.85      0.88      0.83   2300960
```

2.1 Preprocessing the Test Data

(Following the same what we did in case of Train data)

[49]:
```python
data = pd.read_csv("/kaggle/input/playground-series-s4e7/test.csv")
data
```

[49]:
```
               id  Gender  Age  Driving_License  Region_Code  \
0        11504798  Female   20                1         47.0
1        11504799    Male   47                1         28.0
2        11504800    Male   47                1         43.0
3        11504801  Female   22                1         47.0
4        11504802    Male   51                1         19.0
…             …     …   …               …          …
7669861  19174659    Male   57                1         28.0
7669862  19174660    Male   28                1         50.0
7669863  19174661    Male   47                1         33.0
7669864  19174662    Male   30                1         28.0
7669865  19174663    Male   23                1         46.0

         Previously_Insured Vehicle_Age Vehicle_Damage  Annual_Premium  \
0                         0    < 1 Year             No          2630.0
1                         0    1-2 Year            Yes         37483.0
2                         0    1-2 Year            Yes          2630.0
3                         1    < 1 Year             No         24502.0
4                         0    1-2 Year             No         34115.0
…                       …          …              …              …
7669861                   0    1-2 Year            Yes         51661.0
7669862                   1    < 1 Year             No         25651.0
7669863                   1    1-2 Year             No          2630.0
7669864                   0    < 1 Year            Yes         38866.0
```

```
7669865                      1     < 1 Year             No          27498.0

         Policy_Sales_Channel  Vintage
0                       160.0      228
1                       124.0      123
2                        26.0      271
3                       152.0      115
4                       124.0      148
…                           …    …
7669861                 124.0      109
7669862                 152.0      184
7669863                 138.0       63
7669864                 124.0      119
7669865                 152.0       79

[7669866 rows x 11 columns]
```

[50]:

```python
columns_to_drop = ['Region_Code']

# Using the `drop` method
data_new = data.drop(columns=columns_to_drop)
```

[51]:

```python
age_bins = [10, 20, 30, 40, 50, 60, 70, 80, 90]
#premium_bins = [0,10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000,␣
↪90000,100000]

# Apply binning using cut
data_new['Age_Binned'] = pd.cut(data_new['Age'], bins=age_bins,␣
↪labels=[f'{age}-{age+10}' for age in age_bins[:-1]])

data_new.drop(columns=['Age'], inplace=True)
```

[52]:

```python
# Assuming df is your DataFrame
bins = [0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000,␣
↪100000]
labels = range(len(bins)-1)

data_new['Annual_Premium_Binned'] = pd.cut(data_new['Annual_Premium'],␣
↪bins=bins, labels=labels)
```

[53]:

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Assuming df is your DataFrame
label_encoder = LabelEncoder()

# Columns to encode
```

```
columns_to_encode = ['Gender', 'Vehicle_Age', 'Vehicle_Damage', 'Age_Binned']

# Apply label encoding to each column
for column in columns_to_encode:
    data_new[column] = label_encoder.fit_transform(data_new[column])
```

[54]:
```
data_new.drop(columns=['Annual_Premium'], inplace=True)
data_new
```

[54]:
```
               id  Gender  Driving_License  Previously_Insured  Vehicle_Age  \
0        11504798       0                1                   0            1
1        11504799       1                1                   0            0
2        11504800       1                1                   0            0
3        11504801       0                1                   1            1
4        11504802       1                1                   0            0
…             …     …            …                 …          …
7669861  19174659       1                1                   0            0
7669862  19174660       1                1                   1            1
7669863  19174661       1                1                   1            0
7669864  19174662       1                1                   0            1
7669865  19174663       1                1                   1            1

         Vehicle_Damage  Policy_Sales_Channel  Vintage  Age_Binned  \
0                     0                 160.0      228           0
1                     1                 124.0      123           3
2                     1                  26.0      271           3
3                     0                 152.0      115           1
4                     0                 124.0      148           4
…                 …                   …      …         …
7669861               1                 124.0      109           4
7669862               0                 152.0      184           1
7669863               0                 138.0       63           3
7669864               1                 124.0      119           1
7669865               0                 152.0       79           1

        Annual_Premium_Binned
0                           0
1                           3
2                           0
3                           2
4                           3
…                         …
7669861                     5
7669862                     2
7669863                     0
7669864                     3
7669865                     2
```

```
[7669866 rows x 10 columns]
```

[55]:
```python
mode_value = data_new['Annual_Premium_Binned'].mode()[0]
data_new['Annual_Premium_Binned'].fillna(mode_value, inplace=True)

# Ensure the column is integer type
data_new['Annual_Premium_Binned'] = data_new['Annual_Premium_Binned'].
↪astype(int)
```

```
/tmp/ipykernel_13/2780489815.py:2: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  data_new['Annual_Premium_Binned'].fillna(mode_value, inplace=True)
```

2.2 Prediction

[56]:
```python
import pandas as pd
from xgboost import XGBClassifier

# Assuming 'data_new' is your new dataframe for prediction
# data_new = pd.read_csv('path_to_new_data.csv')  # Load your new data if not␣
↪already loaded

# Predict probabilities for data_new using the best model
y_pred_proba = best_model.predict_proba(data_new)[:, 1]  # Predict probability␣
↪of positive response

# Add 'Response' column with predicted probabilities to data_new
data_new['Response'] = y_pred_proba

# Save to CSV file with only 'id' and 'Response' columns
output_file = 'predictions.csv'
data_new[['id', 'Response']].to_csv(output_file, index=False)

print(f"Predictions saved to '{output_file}'")
```

```
Predictions saved to 'predictions.csv'
```

[57]:

```
prediction = pd.read_csv("predictions.csv")
prediction.sample(n=10)
```

[57]:

	id	Response
6851316	18356114	0.049780
1231893	12736691	0.499255
4835523	16340321	0.244009
6284369	17789167	0.000577
1462131	12966929	0.212542
6512807	18017605	0.002250
1348706	12853504	0.215294
126158	11630956	0.274941
5066806	16571604	0.123153
2860125	14364923	0.000431