life-insurance-churn-prediction-1

November 25, 2023

[1]:
```
!pip install keras
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score
```

```
Requirement already satisfied: keras in c:\users\neha\anaconda3\lib\site-
packages (2.13.1)
```

1 DATA CLEANING AND PREPROCESSING

[2]:
```
data = pd.read_csv('randomdata.csv')
```

2 #Data Exploration

[3]:
```
data.head()
```

[3]:
```
   Unnamed: 0     Customer Name  \
0           0   Christine Payne
1           1    Tony Fernandez
2           2   Christopher Kim
3           3      Nicole Allen
4           4        Linda Cruz

                                  Customer_Address  \
0  7627 Anderson Rest Apt. 265,Lake Heather, DC 3…
1  3953 Cindy Brook Apt. 147,East Lindatown, TN 4…
2        8693 Walters Mountains,South Tony, TX 88407
3            56926 Webster Coves,Shawnmouth, NV 04853
4       489 Thomas Forges Apt. 305,Jesseton, GA 36765

                    Company Name Claim Reason Data confidentiality  \
```

```
0  Williams, Henderson and Perez       Travel                 Low
1                  Moore-Goodwin      Medical                High
2                   Smith-Holmes        Phone              Medium
3                  Harrell-Perez        Phone              Medium
4     Simpson, Kramer and Hughes        Phone              Medium

   Claim Amount  Category Premium  Premium/Amount Ratio Claim Request output  \
0           377              4794              0.078640                   No
1          1440             14390              0.100069                   No
2           256              1875              0.136533                   No
3           233              1875              0.124267                   No
4           239              1875              0.127467                   No

   BMI Churn
0   21   Yes
1   24   Yes
2   18   Yes
3   24   Yes
4   21   Yes
```

[4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 12 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Unnamed: 0            200000 non-null  int64
 1   Customer Name         200000 non-null  object
 2   Customer_Address      200000 non-null  object
 3   Company Name          200000 non-null  object
 4   Claim Reason          200000 non-null  object
 5   Data confidentiality  200000 non-null  object
 6   Claim Amount          200000 non-null  int64
 7   Category Premium      200000 non-null  int64
 8   Premium/Amount Ratio  200000 non-null  float64
 9   Claim Request output  200000 non-null  object
 10  BMI                   200000 non-null  int64
 11  Churn                 200000 non-null  object
dtypes: float64(1), int64(4), object(7)
memory usage: 18.3+ MB
```

[5]:
```
data.describe()
```

[5]:
```
          Unnamed: 0   Claim Amount  Category Premium  Premium/Amount Ratio  \
count  200000.000000  200000.000000     200000.000000         200000.000000
mean    99999.500000    1120.478840       8963.783895              0.125024
```

```
std     57735.171256     796.660796       6114.737202                  0.034742
min         0.000000       1.000000        399.000000                  0.002506
25%     49999.750000     245.000000       1875.000000                  0.106741
50%     99999.500000    1390.000000      14390.000000                  0.125122
75%    149999.250000    1844.000000      14390.000000                  0.143155
max    199999.000000    2299.000000      14390.000000                  0.248120

                 BMI
count  200000.000000
mean       23.007205
std         3.164976
min        18.000000
25%        20.000000
50%        23.000000
75%        26.000000
max        28.000000
```

3 # Handling Missing Data

[6]:
```
# Step 1: Remove Duplicate Rows
data.drop_duplicates(inplace=True)
```

[7]:
```
# Step 2: Remove Irrelevant Columns
# Identify and drop columns that are not relevant for churn prediction
irrelevant_columns = ['Customer Name', 'Customer_Address', 'Company Name',␣
 ↪'Data confidentiality', 'Claim Amount', 'Category Premium', 'Premium/Amount␣
 ↪Ratio']
data.drop(columns=irrelevant_columns, inplace=True)
```

[8]:
```
# Step 3: Data Preprocessing
# After removing duplicates and irrelevant columns, you may proceed with data␣
 ↪preprocessing
# This may include handling missing data, encoding categorical variables, and␣
 ↪scaling/normalizing numerical features

# Handling Missing Data
data.dropna(subset=['Churn'], inplace=True)
```

[9]:
```
# Encoding Categorical Variables
# Identify categorical columns in your dataset
import pandas as pd
from sklearn.preprocessing import LabelEncoder
# Load your dataset
data = pd.read_csv('randomdata.csv')
# Identify categorical columns in your dataset
categorical_columns = ['Company Name', 'Claim Reason', 'Category Premium']
```

```python
# Perform label encoding for categorical columns
for column in categorical_columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
# Display the resulting DataFrame
print(data)
```

```
        Unnamed: 0    Customer Name  \
0                0  Christine Payne
1                1   Tony Fernandez
2                2  Christopher Kim
3                3     Nicole Allen
4                4       Linda Cruz
…              …               …
199995      199995  Matthew Estrada
199996      199996       James Bean
199997      199997      David Meyer
199998      199998     Martha Stone
199999      199999    Shannon Lewis

                                          Customer_Address  Company Name  \
0       7627 Anderson Rest Apt. 265,Lake Heather, DC 3…       122584
1       3953 Cindy Brook Apt. 147,East Lindatown, TN 4…        77347
2             8693 Walters Mountains,South Tony, TX 88407       106968
3                56926 Webster Coves,Shawnmouth, NV 04853        44952
4           489 Thomas Forges Apt. 305,Jesseton, GA 36765       104639
…                                                      …           …
199995       2024 Lopez Gateway,Lake Pamelafort, MS 35772        16350
199996             0268 Lori Falls,West Jeffrey, SC 49142       114158
199997         00573 Miller Cliff,New Allenbury, SC 68902       104547
199998          62681 Peters Cove,South Anthony, RI 99783         4850
199999                    Unit 6569 Box 2236,DPO AE 88045        98869

        Claim Reason Data confidentiality  Claim Amount  Category Premium  \
0                  3                  Low           377                 2
1                  0                 High          1440                 3
2                  2               Medium           256                 1
3                  2               Medium           233                 1
4                  2               Medium           239                 1
…              …                    …           …                …
199995             0                 High          1563                 3
199996             0                 High          1342                 3
199997             0                 High          2278                 3
199998             3                  Low           532                 2
199999             0                 High          1755                 3

        Premium/Amount Ratio Claim Request output  BMI Churn
```

```
0                   0.078640              No    21    Yes
1                   0.100069              No    24    Yes
2                   0.136533              No    18    Yes
3                   0.124267              No    24    Yes
4                   0.127467              No    21    Yes
…                      …             …   …    …
199995              0.108617              No    18    Yes
199996              0.093259              No    22    Yes
199997              0.158304              No    19    Yes
199998              0.110972              No    24    Yes
199999              0.121960              No    22    Yes

[200000 rows x 12 columns]
```

[10]:
```python
#Scaling Numerical Features
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data[['Claim Amount', 'Premium/Amount Ratio', 'BMI']] = scaler.
 ↪fit_transform(data[['Claim Amount', 'Premium/Amount Ratio', 'BMI']])
```

[11]:
```python
# Data Splitting
import pandas as pd
from sklearn.model_selection import train_test_split
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, data['Churn'],␣
 ↪test_size=0.2, random_state=42)
# Print the shape of the training and testing sets
print("Training set shape:", X_train.shape)
print("Testing set shape:", X_test.shape)
```

```
Training set shape: (160000, 12)
Testing set shape: (40000, 12)
```

[12]:
```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
# Load your dataset

data = pd.read_csv('randomdata.csv')
# Identify numerical columns in your dataset
numerical_columns = ['Claim Amount', 'Premium/Amount Ratio', 'BMI']
# Create a MinMaxScaler object
scaler = MinMaxScaler()
# Fit the MinMaxScaler to the numerical columns
scaler.fit(data[numerical_columns])
# Transform the numerical columns using the MinMaxScaler
data[numerical_columns] = scaler.transform(data[numerical_columns])
```

```python
# Display the resulting DataFrame
print(data)
```

```
        Unnamed: 0    Customer Name  \
0                0  Christine Payne
1                1   Tony Fernandez
2                2  Christopher Kim
3                3     Nicole Allen
4                4       Linda Cruz
…              …              …
199995      199995  Matthew Estrada
199996      199996       James Bean
199997      199997      David Meyer
199998      199998     Martha Stone
199999      199999    Shannon Lewis

                                          Customer_Address  \
0       7627 Anderson Rest Apt. 265,Lake Heather, DC 3…
1       3953 Cindy Brook Apt. 147,East Lindatown, TN 4…
2             8693 Walters Mountains,South Tony, TX 88407
3                56926 Webster Coves,Shawnmouth, NV 04853
4           489 Thomas Forges Apt. 305,Jesseton, GA 36765
…                                                     …
199995       2024 Lopez Gateway,Lake Pamelafort, MS 35772
199996             0268 Lori Falls,West Jeffrey, SC 49142
199997         00573 Miller Cliff,New Allenbury, SC 68902
199998          62681 Peters Cove,South Anthony, RI 99783
199999                    Unit 6569 Box 2236,DPO AE 88045

                          Company Name Claim Reason Data confidentiality  \
0       Williams, Henderson and Perez       Travel                  Low
1                      Moore-Goodwin      Medical                 High
2                       Smith-Holmes        Phone               Medium
3                      Harrell-Perez        Phone               Medium
4          Simpson, Kramer and Hughes        Phone               Medium
…                                 …          …                   …
199995               Carlson-Matthews      Medical                 High
199996               Trevino-Cardenas      Medical                 High
199997                    Simon-Evans      Medical                 High
199998        Baker, Brooks and Porter       Travel                  Low
199999         Roth, Merritt and Grant      Medical                 High

        Claim Amount  Category Premium  Premium/Amount Ratio  \
0           0.163621              4794              0.309973
1           0.626197             14390              0.397222
2           0.110966              1875              0.545682
3           0.100957              1875              0.495739
```

```
4           0.103568              1875              0.508767
…              …               …                  …
199995      0.679721             14390              0.432023
199996      0.583551             14390              0.369494
199997      0.990862             14390              0.634321
199998      0.231070              4794              0.441611
199999      0.763272             14390              0.486346

       Claim Request output  BMI  Churn
0                        No  0.3    Yes
1                        No  0.6    Yes
2                        No  0.0    Yes
3                        No  0.6    Yes
4                        No  0.3    Yes
…                     …   …   …
199995                   No  0.0    Yes
199996                   No  0.4    Yes
199997                   No  0.1    Yes
199998                   No  0.6    Yes
199999                   No  0.4    Yes

[200000 rows x 12 columns]
```

DATA SPLITTING 80 20

[13]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split

# Load your dataset

data = pd.read_csv('randomdata.csv')

# Assuming 'Churn' is your target variable and you want to predict it
X = data.drop(columns=['Churn'])  # Features
y = data['Churn']  # Target variable

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 ↪random_state=42)

# Display the sizes of the resulting sets
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (160000, 11)
X_test shape: (40000, 11)
```

```
y_train shape: (160000,)
y_test shape: (40000,)
```

DATA SPLITTING 75 25

[14]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load your dataset

data = pd.read_csv('randomdata.csv')

# Assuming 'Churn' is your target variable and you want to predict it
X = data.drop(columns=['Churn'])  # Features
y = data['Churn']  # Target variable

# Split the data into training and testing sets (75% training, 25% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=42)

# Display the sizes of the resulting sets
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (150000, 11)
X_test shape: (50000, 11)
y_train shape: (150000,)
y_test shape: (50000,)
```

DATA SPLITTING 85 15

[15]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load your dataset
# Replace 'randomdata.csv' with the actual path to your dataset
data = pd.read_csv('randomdata.csv')

# Assuming 'Churn' is your target variable and you want to predict it
X = data.drop(columns=['Churn'])  # Features
y = data['Churn']  # Target variable

# Split the data into training and testing sets (85% training, 15% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15,␣
↪random_state=42)

# Display the sizes of the resulting sets
```

```python
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (170000, 11)
X_test shape: (30000, 11)
y_train shape: (170000,)
y_test shape: (30000,)
```

4 FEATURE SELECTION - Ensemble method

[16]:

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

data = pd.read_csv('randomdata.csv')

# Assuming 'Churn' is your target variable and you want to predict it
# Drop non-numeric columns and target variable
X = data.drop(columns=['Churn', 'Customer Name', 'Customer_Address'])
y = data['Churn']  # Target variable

# Encode categorical variables using LabelEncoder
label_encoder = LabelEncoder()
for column in X.columns:
    if X[column].dtype == 'object':
        X[column] = label_encoder.fit_transform(X[column])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Initialize a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the classifier to your data
rf_classifier.fit(X_train, y_train)

# Evaluate the classifier on the testing data
accuracy = rf_classifier.score(X_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

[17]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from sklearn.preprocessing import LabelEncoder
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report


data = pd.read_csv('randomdata.csv')

# Assuming 'Churn' is your target variable and you want to predict it
# Drop non-numeric columns and target variable
X = data.drop(columns=['Churn', 'Customer Name', 'Customer_Address'])
y = data['Churn']  # Target variable

# Encode categorical variables using LabelEncoder
label_encoder = LabelEncoder()
for column in X.columns:
    if X[column].dtype == 'object':
        X[column] = label_encoder.fit_transform(X[column])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Perform oversampling on the training data
oversampler = RandomOverSampler(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = oversampler.fit_resample(X_train,␣
 ↪y_train)

# Initialize a Deep Learning model (MLPClassifier is used here as an example)
clf = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=1000,␣
 ↪random_state=42)

# Fit the classifier to your resampled training data
clf.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
classification_report_result = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report_result)
```

```
Accuracy: 0.6986
Classification Report:
              precision    recall  f1-score   support

          No       0.72      0.28      0.41     14536
         Yes       0.70      0.94      0.80     25464

    accuracy                           0.70     40000
   macro avg       0.71      0.61      0.60     40000
weighted avg       0.70      0.70      0.66     40000
```

5 DEEP LEARNING MODEL - Ensemble method

[18]:
```python
# Define a function to create a deep learning model
def create_model(input_dim):
    model = Sequential()
    model.add(Dense(64, input_dim=input_dim, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
    return model
```

[19]:
```python
# Create three deep learning models
import numpy as np
from tensorflow.keras.models import Sequential  # Import the Sequential class
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler



model1 = create_model(X_train.shape[1])
model2 = create_model(X_train.shape[1])
model3 = create_model(X_train.shape[1])
```

[20]:
```python
from sklearn.preprocessing import LabelEncoder

# Encode the target variable into numerical values (0 and 1)
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)

# Train the deep learning models
model1.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)
```

```python
model2.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)
model3.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0)
```

[20]: <keras.src.callbacks.History at 0x200bebbd6d0>

[21]:
```python
# Make predictions using the individual models
pred1 = model1.predict(X_test)
pred2 = model2.predict(X_test)
pred3 = model3.predict(X_test)
```

```
1250/1250 [==============================] - 1s 621us/step
1250/1250 [==============================] - 1s 601us/step
1250/1250 [==============================] - 1s 610us/step
```

[22]:
```python
# Ensemble the predictions using a simple averaging method
ensemble_preds = np.round((pred1 + pred2 + pred3) / 3)
```

[23]:
```python
# Calculate accuracy of the ensemble model
ensemble_accuracy = accuracy_score(y_test, ensemble_preds)

print("Ensemble Model Accuracy:", ensemble_accuracy)
```

```
Ensemble Model Accuracy: 0.69195
```

[24]:
```python
from sklearn.metrics import accuracy_score, precision_score, f1_score,␣
↪recall_score

# Make predictions using the individual models
pred1 = model1.predict(X_test)
pred2 = model2.predict(X_test)
pred3 = model3.predict(X_test)

# Ensemble the predictions using Voting Classifier
ensemble_preds = np.round((pred1 + pred2 + pred3) / 3)

# Calculate classification metrics
ensemble_accuracy = accuracy_score(y_test, ensemble_preds)
ensemble_precision = precision_score(y_test, ensemble_preds)
ensemble_f1 = f1_score(y_test, ensemble_preds)
ensemble_recall = recall_score(y_test, ensemble_preds)

# Print the evaluation metrics
print("Ensemble Model Metrics:")
print("Accuracy:", ensemble_accuracy)
print("Precision:", ensemble_precision)
print("F1 Score:", ensemble_f1)
print("Recall:", ensemble_recall)
```

```
1250/1250 [==============================] - 1s 617us/step
1250/1250 [==============================] - 1s 635us/step
1250/1250 [==============================] - 1s 732us/step
Ensemble Model Metrics:
Accuracy: 0.69195
Precision: 0.6776137960860633
F1 Score: 0.8027217419148255
Recall: 0.984487904492617
```

Experimenting with another ensemble method - bagging

[25]:
```
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier

# Create a BaggingClassifier with a RandomForest base estimator
bagging_classifier = BaggingClassifier(
    base_estimator=RandomForestClassifier(n_estimators=100, random_state=42),
    n_estimators=10, random_state=42)

# Fit the bagging ensemble to the data
bagging_classifier.fit(X_train, y_train)

# Evaluate the bagging ensemble
bagging_accuracy = bagging_classifier.score(X_test, y_test)

print("Bagging Ensemble Accuracy:", bagging_accuracy)
```

```
C:\Users\Neha\anaconda3\Lib\site-packages\sklearn\ensemble\_base.py:166:
FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and
will be removed in 1.4.
  warnings.warn(
```

```
Bagging Ensemble Accuracy: 1.0
```