Questions

August 8, 2023

1 Predict Employee Attrition

[1]:
```
# If additional packages are needed but are not installed by default, uncomment␣
↪the last two lines of this cell
# and replace <package list> with a list of additional packages.
# This will ensure the notebook has all the dependencies and works everywhere

#import sys
#!{sys.executable} -m pip install <package list>
```

[2]:
```
# Libraries
import pandas as pd
import numpy as np
import sklearn

pd.set_option("display.max_columns", 101)
pd.set_option('display.max_colwidth', 100)
```

1.1 Data Description

Column	Description
`emp_id`	Unique ID corresponding to the employee
`MonthlyIncome`	Monthly Income of the employee
`EmployeeNumber`	Number of employees in the division of given employee
`Age`	Age of the employee
`DistanceFromHome`	Office distance from home.
`OverTime`	Employee works overtime or not
`TotalWorkingYears`	Total Working Experience of employee
`StockOptionLevel`	Company Stocks given to an employee
`YearsAtCompany`	Number of years at current company
`NumCompaniesWorked`	Number of companies in which an employee has worked before joining the current company.
`YearsWithCurrManager`	Number of years with current manager
`JobSatisfaction`	Job Satisfaction of Employee (1-Lowest, 4-Highest)

Column	Description
PercentSalaryHike	Average Annual Salary Hike in Percentages
Attrition	Employee Attrition or not(0-no, 1-yes)

[3]:
```
# Load train and test data, following given naming conventions
data = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

[4]:
```
data.dtypes
```

[4]:
```
emp_id                    int64
MonthlyIncome             int64
EmployeeNumber           object
Age                       int64
DistanceFromHome          int64
OverTime                 object
TotalWorkingYears        object
StockOptionLevel          int64
YearsAtCompany           object
NumCompaniesWorked        int64
YearsWithCurrManager      int64
JobSatisfaction           int64
PercentSalaryHike         int64
Attrition                 int64
dtype: object
```

[5]:
```
test.dtypes
```

[5]:
```
emp_id                    int64
MonthlyIncome             int64
EmployeeNumber           object
Age                       int64
DistanceFromHome          int64
OverTime                 object
TotalWorkingYears        object
StockOptionLevel          int64
YearsAtCompany           object
NumCompaniesWorked        int64
YearsWithCurrManager      int64
JobSatisfaction           int64
PercentSalaryHike         int64
dtype: object
```

[6]:
```
data.head()
```

[6]:
```
   emp_id  MonthlyIncome  EmployeeNumber  Age  DistanceFromHome OverTime  \
0       0          19468             972   51                 2      Yes
1       1           3462             443   28                 2      Yes
2       2           5295            1654   39                12       No
3       3           2073            1592   23                10       No
4       4           -100             106   55                 1       No

  TotalWorkingYears  StockOptionLevel YearsAtCompany  NumCompaniesWorked  \
0                24                 0             12                   3
1                 5                 0              3                   4
2                 7                 0              5                   4
3                 4                 1              2                   2
4                24                 1              1                   3

   YearsWithCurrManager  JobSatisfaction  PercentSalaryHike  Attrition
0                     6                1                 14          1
1                     2                3                 12          1
2                     0                2                 21          0
3                     2                3                 16          0
4                     0                4                 14          0
```

We see negative values for MonthlyIncome. We can either drop these records, use a measure of central tendency to replace invalid values, or set the invalid values equal to 0.

My decision is to drop records with negative values.

We also drop non-informative columns from train and test data

[7]:
```python
# Drop EmployeeNumber column as it's irrelevant
data = data.drop(columns=['EmployeeNumber'])
test = test.drop(columns=['EmployeeNumber'])

# Encode OverTime as 0-1 (trivial case of 1-Hot encoding)
data['OverTime'] = data['OverTime'].map({'Yes': 1, 'No': 0})
test['OverTime'] = test['OverTime'].map({'Yes': 1, 'No': 0})

# Convert 'YearsAtCompany' and 'TotalWorkingYears' to numeric
data['YearsAtCompany'] = pd.to_numeric(data['YearsAtCompany'], errors='coerce')
test['YearsAtCompany'] = pd.to_numeric(test['YearsAtCompany'], errors='coerce')
data['TotalWorkingYears'] = pd.to_numeric(data['TotalWorkingYears'],␣
↪errors='coerce')
test['TotalWorkingYears'] = pd.to_numeric(test['TotalWorkingYears'],␣
↪errors='coerce')

# Drop rows containing NaN values
data = data.dropna()
test = test.dropna()
```

```
# Drop rows where any numerical column is negative
data = data[(data.select_dtypes(include=['number']) >= 0).all(axis=1)]
test = test[(test.select_dtypes(include=['number']) >= 0).all(axis=1)]

# Separate features and target from the training data
X = data.drop(columns=['emp_id', 'Attrition'])
y = data['Attrition']

# Save the emp_id from the test data for later use in the submission file
test_emp_id = test['emp_id']

# Drop the emp_id column from the test data to prepare for prediction
X_test = test.drop(columns=['emp_id'])
```

1.2 Machine Learning

Build a machine learning model that can predict the attrition probability of an employee. - **The model's performance will be evaluated on the basis of AUC ROC.**

1.3 Logistic Regression

[8]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)  # Don't forget to scale the test data too!

# Create the logistic regression model
model = LogisticRegression(max_iter=10000)  # Increased max_iter

# Train the model
model.fit(X_train, y_train)

# Validate the model
y_val_pred = model.predict(X_val)
val_accuracy = accuracy_score(y_val, y_val_pred)
print(f'Validation Accuracy: {val_accuracy}')
```

```
Validation Accuracy: 0.6946107784431138
```

[9]:
```python
from sklearn.metrics import roc_auc_score

# Predict the probabilities for the validation set
y_val_proba = model.predict_proba(X_val)[:, 1]  # Get the probabilities for the␣
↪positive class

# Compute the ROC AUC score
roc_auc = roc_auc_score(y_val, y_val_proba)
print(f'ROC AUC Score: {roc_auc}')
```

```
ROC AUC Score: 0.7078713968957872
```

Since the ROC AUC score for Random Forest is higher, we use it as our model.

1.4 Random Forest

[10]:
```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Split the data into training and validation sets (80% training, 20%␣
↪validation)
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Ensure the test data has the same columns in the same order
X_test = pd.DataFrame(X_test, columns=X_train.columns)

# Create Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training data
model.fit(X_train, y_train)
```

[10]:
```
RandomForestClassifier(random_state=42)
```

[11]:
```python
from sklearn.metrics import accuracy_score

# Validate the model on the validation data
y_val_pred = model.predict(X_val)
accuracy = accuracy_score(y_val, y_val_pred)
print("Validation Accuracy: {:.2f}%".format(accuracy * 100))

# Make predictions on the test data (predicting the probabilities for class 1)
test_pred_proba = model.predict_proba(X_test)[:, 1]
```

```
Validation Accuracy: 78.44%
```

[12]:
```python
from sklearn.metrics import roc_auc_score

# Predict the probabilities for the validation set
y_val_proba = model.predict_proba(X_val)[:, 1]  # Get the probabilities for the
 ↪positive class

# Compute the ROC AUC score
roc_auc = roc_auc_score(y_val, y_val_proba)
print(f'ROC AUC Score: {roc_auc}')
```

```
ROC AUC Score: 0.809220251293422
```

[13]:
```python
# Create the submission DataFrame
submission_df = pd.DataFrame({
    'id': test_emp_id.astype(int),
    'pred_proba': test_pred_proba
})
```

[14]:
```python
submission_df.dtypes
```

[14]:
```
id              int64
pred_proba    float64
dtype: object
```

[15]:
```python
submission_df.head()
```

[15]:

	id	pred_proba
0	1104	0.78
3	1107	0.87
4	1108	0.74
5	1109	0.73
8	1112	0.83

Task:

- **Submit the predictions on the test dataset using your optimized model** Submit a CSV file with a header row plus each of the test entries, each on its own line.

The file (`submissions.csv`) should have exactly 2 columns:

Column	Description
`emp_id`	Unique ID corresponding to the employee
`pred_proba`	Predicted probability of class 1

[16]:
```
#Submission
submission_df.to_csv('submissions.csv', index=False)
```