Peer-graded Assignment Course Final Project

August 29, 2024

[350]:
```
# Peer-graded Assignment: Course Final Project
```

[351]:
```
import pandas as pd
import numpy as np
```

[352]:
```
# Install xgboost if not already installed
!pip install xgboost
```

```
Requirement already satisfied: xgboost in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (1.6.2)
Requirement already satisfied: numpy in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from xgboost)
(1.21.6)
Requirement already satisfied: scipy in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from xgboost)
(1.7.3)
```

[353]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix
from xgboost import XGBClassifier
```

[354]:
```
# Suppress DeprecationWarnings
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

[355]:
```
# Creating a synthetic dataset for employee attrition
np.random.seed(0)
n = 1000  # Number of samples

data = {
    'Age': np.random.randint(20, 60, size=n),
    'DistanceFromHome': np.random.randint(1, 30, size=n),
    'Education': np.random.randint(1, 5, size=n),
    'JobRole': np.random.choice(['Sales Executive', 'Research Scientist',␣
 ↪'Laboratory Technician', 'Manager', 'Healthcare Representative'], size=n),
    'MonthlyIncome': np.random.randint(2000, 15000, size=n),
    'NumCompaniesWorked': np.random.randint(0, 10, size=n),
    'TotalWorkingYears': np.random.randint(1, 40, size=n),
    'TrainingTimesLastYear': np.random.randint(1, 7, size=n),
    'WorkLifeBalance': np.random.randint(1, 5, size=n),
    'Attrition': np.random.choice([0, 1], size=n)  # Binary target variable
}

df = pd.DataFrame(data)
```

[356]:
```
# Load the synthetic dataset
file_path = 'synthetic_employee_attrition.csv'
```

[357]:
```
# Update this path if necessary
df = pd.read_csv(file_path)
```

[358]:
```
# Load the dataset
```

[359]:
```
# Use a different dataset or check the dataset URL path
file_path = 'synthetic_employee_attrition.csv'  # Ensure this path is correct
df = pd.read_csv(file_path)
```

[360]:
```
# Display the first few rows of the dataset
df.head()
```

[360]:

	Age	DistanceFromHome	Education	JobRole	MonthlyIncome	NumCompaniesWorked	TotalWorkingYears	TrainingTimesLastYear	\
0	20	5	4	Manager	2358	3	5	6	
1	23	29	4	Research Scientist	6553	0	30	6	
2	23	2	2	Research Scientist	10856	7	19	4	
3	59	18	3	Laboratory Technician	3757	1	38	5	
4	29	18	4	Sales Executive	11733				

```
4                   7                 3                       3

   WorkLifeBalance  Attrition
0                2          1
1                4          1
2                2          1
3                4          1
4                1          1
```

[361]:
```
# Summary of the dataset
df.info()
df.describe(include='all')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    1000 non-null   int64
 1   DistanceFromHome       1000 non-null   int64
 2   Education              1000 non-null   int64
 3   JobRole                1000 non-null   object
 4   MonthlyIncome          1000 non-null   int64
 5   NumCompaniesWorked     1000 non-null   int64
 6   TotalWorkingYears      1000 non-null   int64
 7   TrainingTimesLastYear  1000 non-null   int64
 8   WorkLifeBalance        1000 non-null   int64
 9   Attrition              1000 non-null   int64
dtypes: int64(9), object(1)
memory usage: 78.2+ KB
```

[361]:
```
                Age  DistanceFromHome    Education  JobRole  MonthlyIncome  \
count   1000.000000        1000.00000  1000.000000     1000    1000.000000
unique          NaN               NaN          NaN        5            NaN
top             NaN               NaN          NaN  Manager            NaN
freq            NaN               NaN          NaN      216            NaN
mean      39.322000           15.03700     2.539000      NaN    8548.185000
std       11.608923            8.28458     1.103493      NaN    3757.125205
min       20.000000            1.00000     1.000000      NaN    2013.000000
25%       29.000000            8.00000     2.000000      NaN    5285.750000
50%       39.000000           15.00000     3.000000      NaN    8479.500000
75%       49.000000           22.00000     4.000000      NaN   11873.000000
max       59.000000           29.00000     4.000000      NaN   14994.000000

        NumCompaniesWorked  TotalWorkingYears  TrainingTimesLastYear  \
count           1000.00000        1000.000000            1000.000000
unique                 NaN                NaN                    NaN
```

```
top                    NaN             NaN                    NaN
freq                   NaN             NaN                    NaN
mean               4.55100       20.049000               3.510000
std                2.90996       11.257501               1.666937
min                0.00000        1.000000               1.000000
25%                2.00000       10.000000               2.000000
50%                5.00000       21.000000               3.000000
75%                7.00000       30.000000               5.000000
max                9.00000       39.000000               6.000000

        WorkLifeBalance    Attrition
count       1000.000000  1000.000000
unique              NaN          NaN
top                 NaN          NaN
freq                NaN          NaN
mean           2.531000     0.489000
std            1.142079     0.500129
min            1.000000     0.000000
25%            1.000000     0.000000
50%            3.000000     0.000000
75%            4.000000     1.000000
max            4.000000     1.000000
```

[ ]:

[362]:
```
# Load the dataset from local file path
file_path = 'path/to/employee_attrition.csv'
```

[363]:
```
# Data Cleaning and Preprocessing
```

[364]:
```
# Drop columns if necessary
```

[365]:
```
# In this synthetic dataset, we have no unnecessary columns to drop
```

[366]:
```
# Define features and target variable
X = df.drop('Attrition', axis=1)
y = df['Attrition']
```

[367]:
```
# Separate categorical and numerical features
categorical_features = X.select_dtypes(include=['object']).columns
numerical_features = X.select_dtypes(exclude=['object']).columns
```

[368]:
```
# Preprocessing for numerical data
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

[369]:
```
# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

[370]:
```
# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

[371]:
```
# Create a pipeline with preprocessing and classifier
def build_pipeline(model):
    return Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', model)
    ])
```

[372]:
```
# Define models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(),
    'XGBoost': XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
}
```

[373]:
```
# Train and evaluate models
results = {}
for name, model in models.items():
    print(f"Training {name}...")
```

```
Training Logistic Regression…
Training Random Forest…
Training XGBoost…
```

[374]:
```
    # Create pipeline
    pipeline = build_pipeline(model)
```

[375]:
```
    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[376]:
```
pip install --upgrade numpy scikit-learn xgboost
```

```
Requirement already satisfied: numpy in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (1.21.6)
```

```
Requirement already satisfied: scikit-learn in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (1.0.2)
Requirement already satisfied: xgboost in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (1.6.2)
Requirement already satisfied: scipy>=1.1.0 in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from scikit-
learn) (1.7.3)
Requirement already satisfied: joblib>=0.11 in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from scikit-
learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages (from scikit-
learn) (3.1.0)
Note: you may need to restart the kernel to use updated packages.
```

[377]:
```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

[378]:
```
    # Train model
    pipeline.fit(X_train, y_train)
```

[378]:
```
Pipeline(memory=None,
     steps=[('preprocessor', ColumnTransformer(n_jobs=None, remainder='drop',
sparse_threshold=0.3,
         transformer_weights=None,
         transformers=[('num', Pipeline(memory=None,
     steps=[('imputer', SimpleImputer(copy=True, fill_value=None,
missing_values=nan,
       strategy='median', verbo…       tree_method='exact',
use_label_encoder=False, validate_parameters=1,
       verbosity=None))])
```

[379]:
```
    # Make predictions
    y_pred = pipeline.predict(X_test)
```

[380]:
```
    # Evaluate model
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
```

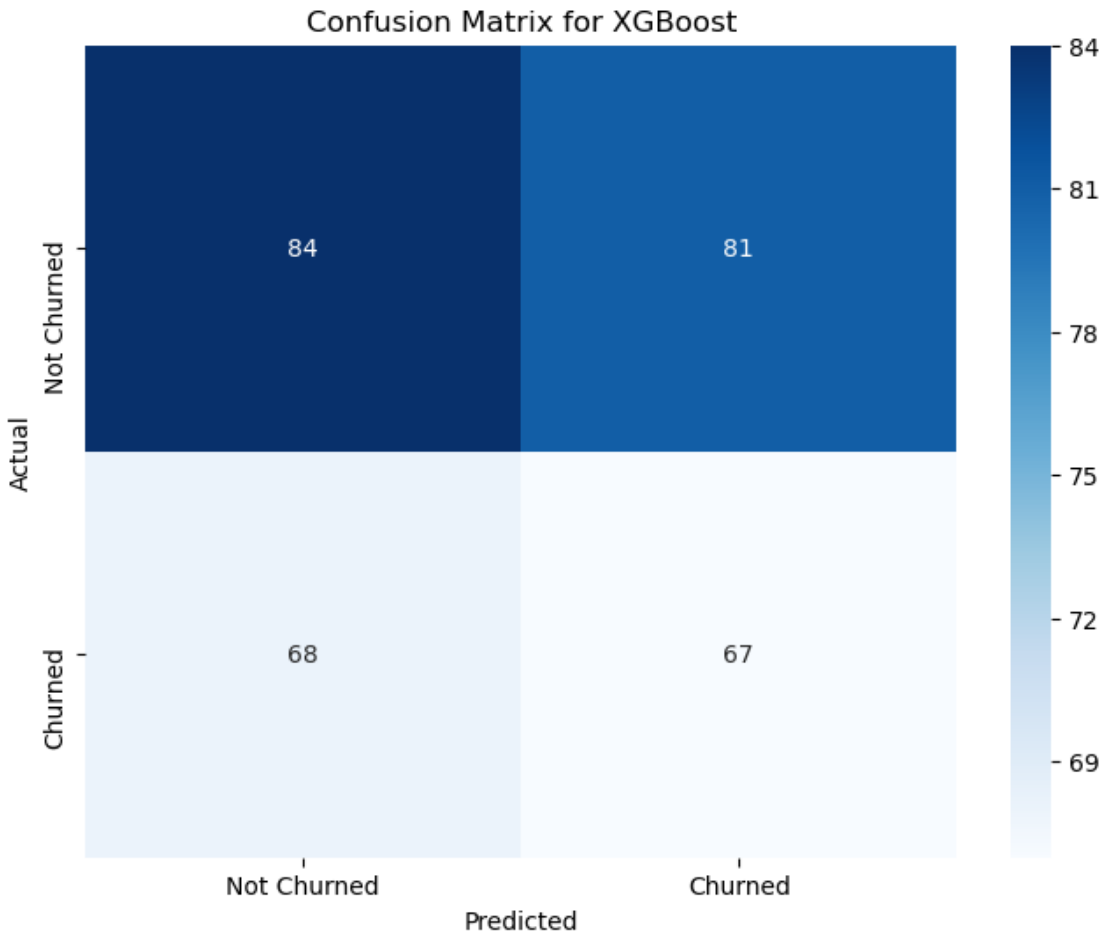
[381]:
```
    # Store results
    results[name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
```

```
    }
```

[382]:
```
    # Print evaluation results
    print(f"{name} Results:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
```

```
XGBoost Results:
Accuracy: 0.5033
Precision: 0.4527
Recall: 0.4963
F1 Score: 0.4735
```

[383]:
```
    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Not Churned', 'Churned'],
                yticklabels=['Not Churned', 'Churned'])
    plt.title(f'Confusion Matrix for {name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```



[384]:
```
# Summary of results
results_df = pd.DataFrame(results).T
results_df.sort_values(by='F1 Score', ascending=False, inplace=True)
results_df
```

[384]:

	Accuracy	F1 Score	Precision	Recall
XGBoost	0.503333	0.473498	0.452703	0.496296

[ ]: