Employee Attrition Report:

by Tselane Moeti

Introduction

Employee attrition, or the loss of employees over time, is a significant challenge for many organizations. Understanding and predicting whether an employee is likely to leave or stay with a company can provide valuable insights into HR strategies, employee satisfaction, and overall company performance. This code demonstrates how to train a machine learning model to predict employee attrition using a dataset containing various attributes like age, job role, job satisfaction, and more. The project involves exploratory data analysis (EDA), data preprocessing, training a logistic regression model, and deploying it using AWS SageMaker.

We first started by importing essential Python libraries:

- pandas and numpy for data manipulation.
- matplotlib and seaborn for data visualization.
- warnings to ignore warning messages

The dataset is loaded from a CSV file named train.csv. The first 10 rows are displayed to inspect the data.

```python
#import the packages or libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()
import seaborn as sns
pd.set_option('display.max_columns', None)
import warnings
warnings.filterwarnings('ignore')
```

Python

```python
#loading the data from the datasets
df=pd.read_csv('train.csv')
df.head(10)
```

Python

	Employee ID	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotions	Overtime	Distance from Home	Education Level	Marital Status	Numb Depend
0	8410	31	Male	19	Education	5390	Excellent	Medium	Average	2	No	22	Associate Degree	Married	
1	64756	59	Female	4	Media	5534	Poor	High	Low	3	No	21	Master's Degree	Divorced	
2	30257	24	Female	10	Healthcare	8159	Good	High	Low	0	No	11	Bachelor's Degree	Married	
													High		

We then performed various EDA steps to better understand the dataset.

```python
df.describe()
```

Python

	Employee ID	Age	Years at Company	Monthly Income	Number of Promotions	Distance from Home	Number of Dependents	Company Tenure
count	59598.000000	59598.000000	59598.000000	59598.000000	59598.000000	59598.000000	59598.000000	59598.000000
mean	37227.118729	38.565875	15.753901	7302.397983	0.832578	50.007651	1.648075	55.758415
std	21519.150028	12.079673	11.245981	2151.457423	0.994991	28.466459	1.555689	25.411090
min	1.000000	18.000000	1.000000	1316.000000	0.000000	1.000000	0.000000	2.000000
25%	18580.250000	28.000000	7.000000	5658.000000	0.000000	25.000000	0.000000	36.000000
50%	37209.500000	39.000000	13.000000	7354.000000	1.000000	50.000000	1.000000	56.000000
75%	55876.750000	49.000000	23.000000	8880.000000	2.000000	75.000000	3.000000	76.000000
max	74498.000000	59.000000	51.000000	16149.000000	4.000000	99.000000	6.000000	128.000000

```python
#get the number of rows and columns
df.shape
```

Python

```
(59598, 24)
```

```python
#Get the data types
df.dtypes
```

Python

```
Employee ID              int64
Age                      int64
Gender                  object
Years at Company         int64
Job Role                object
Monthly Income           int64
Work-Life Balance       object
Job Satisfaction        object
Performance Rating      object
Number of Promotions     int64
Overtime                object
Distance from Home       int64
Education Level         object
Marital Status          object
Number of Dependents     int64
```

df.describe() - Provides summary statistics for each column.

df.shape and df.dtypes - Gives the number of rows and columns and the data types of each feature.

```python
#Get the count of the empty values for each columns
df.isna().sum()
```

```
Employee ID                 0
Age                         0
Gender                      0
Years at Company            0
Job Role                    0
Monthly Income              0
Work-Life Balance           0
Job Satisfaction            0
Performance Rating          0
Number of Promotions        0
Overtime                    0
Distance from Home          0
Education Level             0
Marital Status              0
```

df.isna().sum() and df.isnull().values.any() - Checks for missing values in the dataset.

Categoric Data inspection

```python
#Get the count of the number of Employee that stayed or left the company
df['Attrition'].value_counts()
```

```
Attrition
Stayed     31260
Left       28338
Name: count, dtype: int64
```

```python
#Get all the data types and their unique values
for column in df.columns:
    if df[column].dtype == object:
        print(str(column)+ ' : '+ str(df[column].unique()))
        print(df[column].value_counts())
        print('_________________')
```

```
Gender : ['Male' 'Female']
Gender
Male      32739
Female    26859
Name: count, dtype: int64
_________________
Job Role : ['Education' 'Media' 'Healthcare' 'Technology' 'Finance']
Job Role
Technology    15507
Healthcare    13642
Education     12490
```

The loop above inspects each categorical column and displays its unique values and frequency counts.

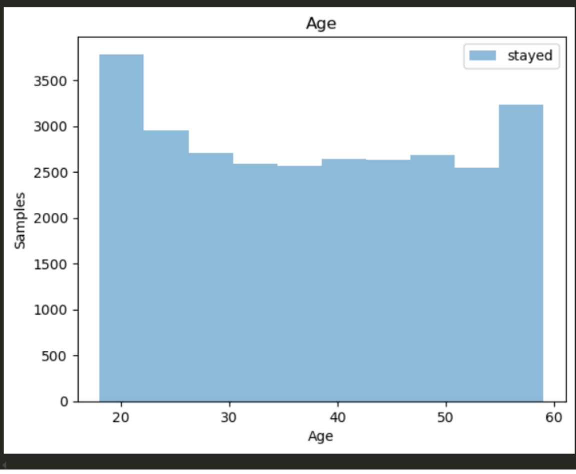
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Attrition']= le.fit_transform(df['Attrition'])
df.head(20)
```

	Employee ID	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotions	Overtime	Distance from Home	Education Level	Marital Status	Numl Depen
0	8410	31	Male	19	Education	5390	Excellent	Medium	Average	2	No	22	Associate Degree	Married	
1	64756	59	Female	4	Media	5534	Poor	High	Low	3	No	21	Master's Degree	Divorced	
2	30257	24	Female	10	Healthcare	8159	Good	High	Low	0	No	11	Bachelor's Degree	Married	
3	65791	36	Female	7	Education	3989	Good	High	High	1	No	27	High School	Single	
4	65026	56	Male	41	Education	4821	Fair	Very High	Average	0	Yes	71	High School	Divorced	
5	24368	38	Female	3	Technology	9977	Fair	High	Below Average	3	No	37	Bachelor's Degree	Married	

The Attrition column is label-encoded to convert categorical values into numerical values for machine learning. The mapping is Stayed: 0 and Left: 1

Visual Analysis

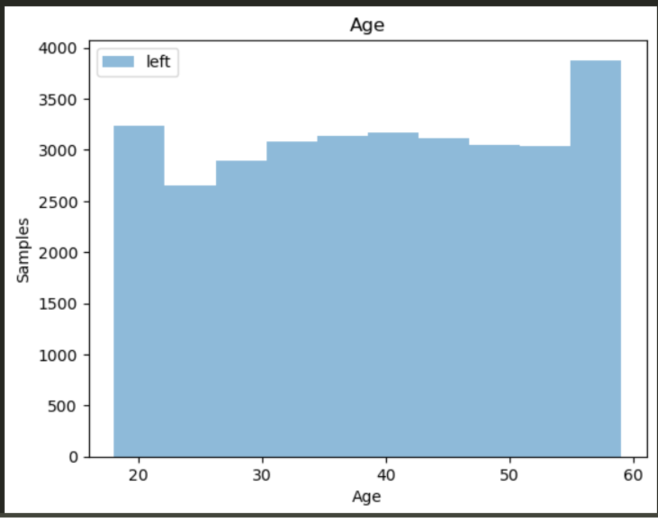
```python
plt.hist(df[stayed].Age, alpha=0.5, label='stayed')
plt.title('Age')
plt.xlabel ('Age')
plt.ylabel('Samples')
plt.legend()
plt.show()
```



The above histogram shows the age distribution for employees who stayed.

```python
plt.hist(df[left].Age, alpha=0.5, label='left')
plt.title('Age')
plt.xlabel ('Age')
plt.ylabel('Samples')
plt.legend()
plt.show()
```

[61]



The above histogram shows the age distribution for employees who stayed left.

```python
#Define the columns to be laebl encoded
labels_cols = ['Gender','Job Role','Overtime','Education Level','Marital Status','Company Size','Remote Work',
               'Leadership Opportunities', 'Innovation Opportunities','Work-Life Balance', 'Job Satisfaction','Performance Rating',
               'Company Reputation','Job Level', 'Employee Recognition']

#initialize label encoders
label_encoders = {col: LabelEncoder() for col in labels_cols}

#Apply the label Encoding
for col in labels_cols:
    df[col] = label_encoders[col].fit_transform(df[col])
```

Python

```python
df.head(10)
```

Python

	Employee ID	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotions	Overtime	Distance from Home	Education Level	Marital Status	Number of Dependents
0	8410	31	1	19	0	5390	0	2	0	2	0	22	0	1	0
1	64756	59	0	4	3	5534	3	0	3	3	0	21	3	0	3
2	30257	24	0	10	2	8159	2	0	3	0	0	11	1	1	3
3	65791	36	0	7	0	3989	2	0	2	1	0	27	2	2	2

Multiple categorical columns are label-encoded to make them machine-readable.

Model Training

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

```python
xTrain, xTest, yTrain, yTest = train_test_split(x,y, test_size = 0.2)
```

```python
xTrain.head(5)
```

	Employee ID	Age	Gender	Years at Company	Job Role	Monthly Income	Work-Life Balance	Job Satisfaction	Performance Rating	Number of Promotions	Overtime	Distance from Home	Education Level	Marital Status	Numbe Depend
3671	62709	48	1	36	2	7765	3	0	0	0	0	26	1	1	
10513	30120	32	0	10	4	8990	2	3	2	1	0	85	4	1	
40348	31262	21	1	1	1	9501	1	2	0	1	0	77	1	2	
50069	21796	53	1	3	3	6091	2	0	0	0	1	98	1	2	
12487	46002	19	0	6	1	6838	1	2	0	1	1	38	0	1	

The data is split into training and test sets.

```python
trainDF.to_csv('traineddataattritions.csv',index=False, index_label='Row',header=False, columns=column)
```

```python
testDF.head()
```

	Age	Gender	Years at Company	Job Role	Marital Status	Education Level	Job Level	Number of Dependents	Monthly Income	Work-Life Balance	Job Satisfaction	Overtime	Distance from Home	Con
37979	41	1	21	0	1	1	1	0	5614	0	0	0	36	
41491	49	0	27	2	2	3	1	0	8864	2	0	0	79	
38958	52	0	2	2	0	2	1	1	7361	3	3	0	29	
2318	25	1	1	0	1	1	0	0	3437	1	3	1	81	
12923	26	1	7	0	1	3	1	1	3867	3	0	0	42	

```python
trainDF.to_csv('testddataattritions.csv',index=False, index_label='Row',header=False, columns=column)
```

We then saved the processed training and test datasets as CSV files and uploaded them to an S3 bucket.

```python
bucketNM = 'sagemakeremployeeattrition'
TrainFile = r'attritiondata/traineddataattritions/traineddataattritions.csv'
TestFile = r'attritiondata/testddataattritions/testddataattritions.csv'
ValFile = r'attritiondata/Val/Val.csv'
ModelFolder = r'attritiondata/model/'
```

Python

```python
s3ModelOutput = r's3://{0}/{1}'.format(bucketNM,ModelFolder)
s3Train = r's3://{0}/{1}'.format(bucketNM,TrainFile)
s3Test = r's3://{0}/{1}'.format(bucketNM,TestFile)
s3Val = r's3://{0}/{1}'.format(bucketNM,ValFile)
```

Python

```python
s3ModelOutput
```

Python

```
's3://sagemakeremployeeattrition/attritiondata/model/'
```

```python
with open('traineddataattritions.csv','rb') as f:
    boto3.Session().resource('s3').Bucket(bucketNM).Object(TrainFile).upload_fileobj(f)
```

Python

```python
with open('testddataattritions.csv','rb') as f:
    boto3.Session().resource('s3').Bucket(bucketNM).Object(TestFile).upload_fileobj(f)
```

Python

- **bucketNM**: The S3 bucket name is defined as sagemakeremployeeattrition.
- **File Paths**: The paths for the training data, testing data, validation data, and model folder are defined relative to the S3 bucket.
- **s3ModelOutput**: The S3 URI for storing the model output after training.
- **s3Train, s3Test, s3Val**: S3 URIs for the training, testing, and validation data files, respectively. These URIs are generated dynamically using Python's format method.
- The with open() statement reads the CSV files in binary mode ('rb').
- The boto3.Session().resource('s3').Bucket(bucketNM).Object(File).upload_fileobj(f) command uploads the opened file to the specified S3 location.

```python
LogisticModel=sagemaker.estimator.Estimator(image_uri=ECRdockercontainer,
                                            role=role,
                                             train_instance_count=1,
                                             train_instance_type='ml.m4.xlarge',
                                             output_path=s3ModelOutput,
                                             sagemaker_session=sagemakerSess,
                                             base_job_name = 'Logistic-Demo-v1'
                                            )
```

```
train_instance_count has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
train_instance_type has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

```python
LogisticModel.set_hyperparameters(predictor_type='binary_classifier', mini_batch_size=100)
```

```python
LogisticModel.hyperparameters()
```

```
{'predictor_type': 'binary_classifier', 'mini_batch_size': 100}
```

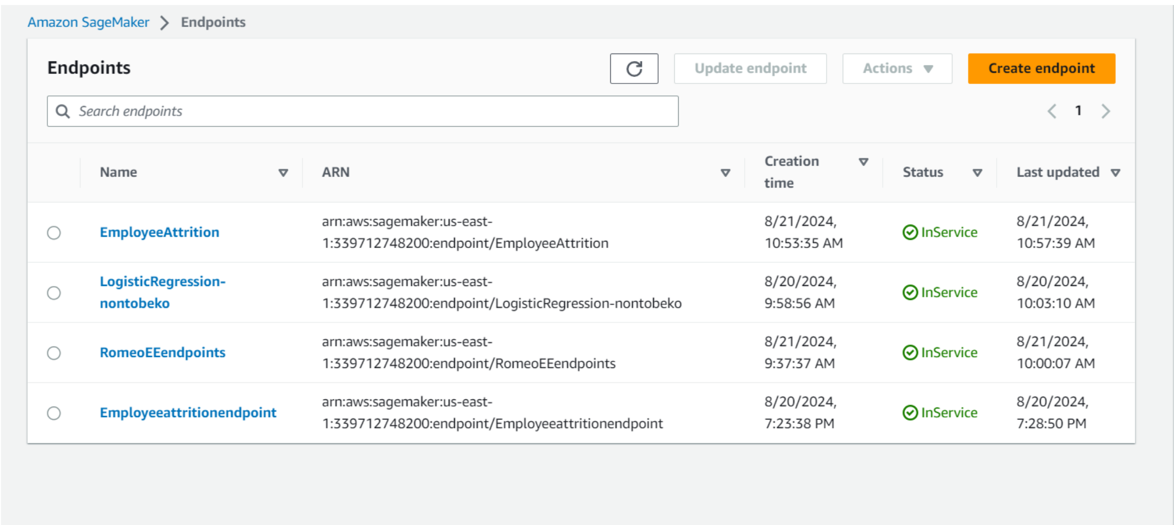
We then define a logistic regression model using SageMaker's Estimator and configures it. The model is trained on the uploaded training data.

```python
#Deploying the Trained Model

predictmodel=LogisticModel.deploy(initial_instance_count=1, instance_type='ml.m4.xlarge',
                                  endpoint_name = 'LogisticRegression-demo-v1')
```

```
INFO:sagemaker:Creating model with name: Logistic-Demo-v1-2024-08-19-17-25-43-398
INFO:sagemaker:Creating endpoint-config with name LogisticRegression-demo-v1
INFO:sagemaker:Creating endpoint with name LogisticRegression-demo-v1
---------!
```

The trained model is deployed on an endpoint for real-time predictions.

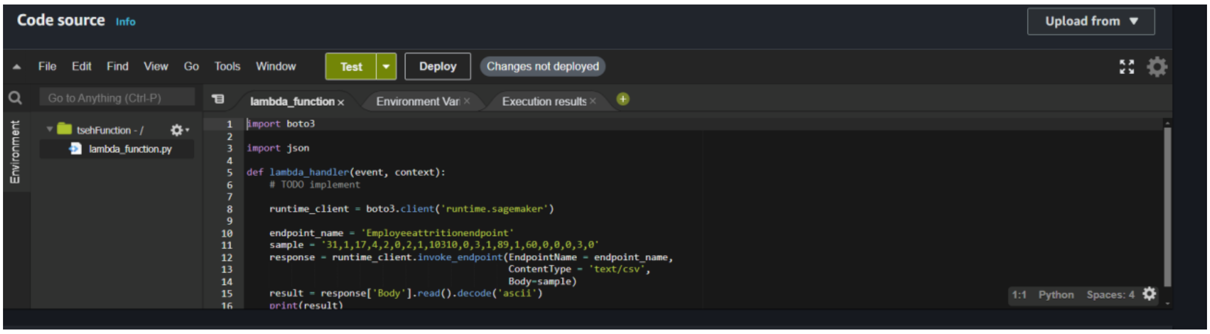

Amazon SageMaker > Endpoints

Endpoints

Name	ARN	Creation time	Status	Last updated
EmployeeAttrition	arn:aws:sagemaker:us-east-1:339712748200:endpoint/EmployeeAttrition	8/21/2024, 10:53:35 AM	InService	8/21/2024, 10:57:39 AM
LogisticRegression-nontobeko	arn:aws:sagemaker:us-east-1:339712748200:endpoint/LogisticRegression-nontobeko	8/20/2024, 9:58:56 AM	InService	8/20/2024, 10:03:10 AM
RomeoEEendpoints	arn:aws:sagemaker:us-east-1:339712748200:endpoint/RomeoEEendpoints	8/21/2024, 9:37:37 AM	InService	8/21/2024, 10:00:07 AM
Employeeattritionendpoint	arn:aws:sagemaker:us-east-1:339712748200:endpoint/Employeeattritionendpoint	8/20/2024, 7:23:38 PM	InService	8/20/2024, 7:28:50 PM

The above image shows the endpoints after successful deployment.

AWS Lambda function setup for deploying a SageMaker endpoint



Environment:

- The function is named lambda_function.py.
- It's part of a Lambda function deployed in a folder named tsehFunction.

Imports:

- The script imports boto3, which is the AWS SDK for Python used for interacting with AWS services.
- The script also imports json to handle JSON data.

Lambda Handler:

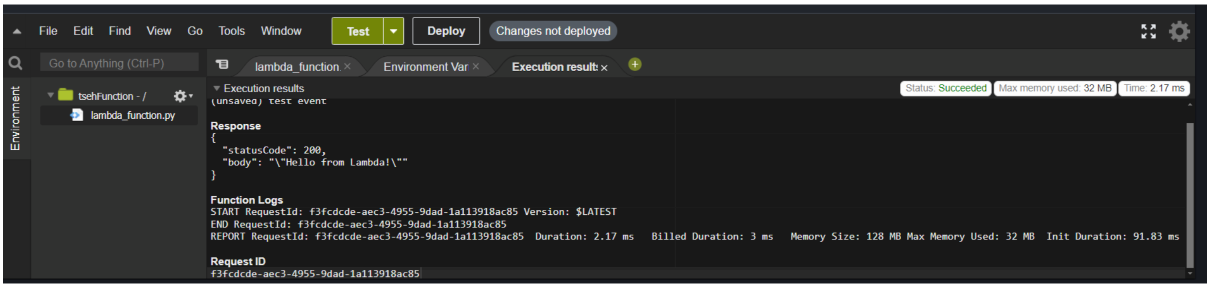
- The lambda_handler function is defined to be triggered when the Lambda is invoked.
- The core of the function involves invoking a SageMaker endpoint using the boto3 client.

SageMaker Endpoint Invocation:

- The code uses boto3.client('runtime.sagemaker') to create a client to interact with SageMaker runtime.
- The endpoint name is set as 'Employeeattritionendpoint'.
- A sample input is prepared as a CSV string: "31,1,17,4,2,0,2,1,10310,0,3,1,89,1,60,0,0,0,3,0".
- The invoke_endpoint method is used to send this data to the SageMaker endpoint, specifying the EndpointName and ContentType as text/csv.

Result Handling:

- The response is decoded using .decode('ascii'), indicating that the output from the SageMaker model is expected as text.
- The result is printed.



Test Invocation:

- A test event was triggered in the Lambda console, returning a JSON response
- This indicates that the basic Lambda setup is functioning correctly with a simple test, though it might not yet be fully integrated with the SageMaker invocation.

Logs:

- The logs confirm that the Lambda function executed successfully with a statusCode: 200.

<u>Conclusion</u>

Overall this analysis provides a comprehensive pipeline for training, testing, and deploying a machine learning model for predicting employee attrition. The use of AWS SageMaker ensures that the model can be scaled and integrated into production systems seamlessly. The setup and testing steps indicate that the initial integration between AWS Lambda and SageMaker is successful. The Lambda function is correctly set up to invoke a SageMaker endpoint and return a response. The environment and dependencies are in place, with successful test execution. The focus now should be on validating model predictions by passing actual data to the endpoint, which will determine the effectiveness of this deployment for predicting employee attrition. The project is on track, and with further testing, it can be confirmed if the deployment is fully operational.