**End-to-End Customer Churn Prediction on AWS**

This case study demonstrates a complete workflow for predicting customer churn using AWS services. We'll use a telecommunications customer dataset to predict which customers are likely to cancel their service.

**Prerequisites**

* An AWS account with appropriate permissions
* AWS CLI configured with your credentials
* Python 3.7 or later
* Required Python packages: boto3, sagemaker, pandas, numpy, matplotlib, seaborn

Install required packages:

bash

Copy

pip install boto3 sagemaker pandas numpy matplotlib seaborn

**Step 1: Data Preparation**

First, we'll create a sample dataset and upload it to S3. In a real-world scenario, you would replace this with your own data.

python

Copy

import pandas as pd

import numpy as np

import boto3

import io

*# Create sample data*

np.random.seed(42)

n\_customers = 1000

data = pd.DataFrame({

'customer\_id': range(1, n\_customers + 1),

'tenure': np.random.randint(1, 72, n\_customers),

'monthly\_charges': np.random.uniform(20, 100, n\_customers),

'total\_charges': np.random.uniform(100, 5000, n\_customers),

'contract': np.random.choice(['Month-to-month', 'One year', 'Two year'], n\_customers),

'online\_security': np.random.choice(['No', 'Yes'], n\_customers),

'tech\_support': np.random.choice(['No', 'Yes'], n\_customers),

'streaming\_tv': np.random.choice(['No', 'Yes'], n\_customers),

'streaming\_movies': np.random.choice(['No', 'Yes'], n\_customers),

'churn': np.random.choice([0, 1], n\_customers, p=[0.7, 0.3]) *# 30% churn rate*

})

*# Calculate total\_charges based on tenure and monthly\_charges*

data['total\_charges'] = data['tenure'] \* data['monthly\_charges']

*# Upload to S3*

bucket\_name = 'your-bucket-name' *# Replace with your S3 bucket name*

key = 'churn\_data/telco\_churn.csv'

s3 = boto3.client('s3')

csv\_buffer = io.StringIO()

data.to\_csv(csv\_buffer, index=False)

s3.put\_object(Bucket=bucket\_name, Key=key, Body=csv\_buffer.getvalue())

print(f"Data uploaded to s3://{bucket\_name}/{key}")

*# To use your own data, comment out the code above and use:*

*# s3.upload\_file('path/to/your/data.csv', bucket\_name, key)*

**Step 2: Data Exploration and Preprocessing**

We'll use SageMaker to explore and preprocess our data.

python

Copy

import sagemaker

from sagemaker.session import Session

from sagemaker.processing import ScriptProcessor, ProcessingInput, ProcessingOutput

sagemaker\_session = Session()

role = sagemaker.get\_execution\_role()

*# Create a ScriptProcessor*

processor = ScriptProcessor(

command=['python3'],

image\_uri=sagemaker.image\_uris.retrieve(

framework="sklearn",

region=sagemaker\_session.boto\_region\_name,

version="0.23-1"),

role=role,

instance\_count=1,

instance\_type='ml.m5.xlarge'

)

*# Run the processing job*

processor.run(

code='preprocess.py',

inputs=[ProcessingInput(

source=f's3://{bucket\_name}/{key}',

destination='/opt/ml/processing/input'

)],

outputs=[

ProcessingOutput(output\_name='train', source='/opt/ml/processing/train'),

ProcessingOutput(output\_name='validation', source='/opt/ml/processing/validation'),

ProcessingOutput(output\_name='test', source='/opt/ml/processing/test')

],

arguments=['--input-data', '/opt/ml/processing/input/telco\_churn.csv']

)

print("Data preprocessing completed.")

Create a file named preprocess.py with the following content:

python

Copy

import argparse

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

def preprocess\_telco\_data(df):

*# Convert categorical variables to numeric*

df['contract'] = pd.Categorical(df['contract']).codes

df['online\_security'] = pd.Categorical(df['online\_security']).codes

df['tech\_support'] = pd.Categorical(df['tech\_support']).codes

df['streaming\_tv'] = pd.Categorical(df['streaming\_tv']).codes

df['streaming\_movies'] = pd.Categorical(df['streaming\_movies']).codes

*# Separate features and target*

X = df.drop(['customer\_id', 'churn'], axis=1)

y = df['churn']

*# Scale features*

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

return pd.DataFrame(X\_scaled, columns=X.columns), y

if \_\_name\_\_=='\_\_main\_\_':

parser = argparse.ArgumentParser()

parser.add\_argument('--input-data', type=str, required=True)

args = parser.parse\_args()

*# Read input data*

input\_data = pd.read\_csv(args.input\_data)

print('Shape of input data:', input\_data.shape)

*# Preprocess data*

X, y = preprocess\_telco\_data(input\_data)

print('Shape of preprocessed features:', X.shape)

*# Split data into train, validation, and test sets*

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

*# Save preprocessed datasets*

pd.concat([X\_train, y\_train], axis=1).to\_csv('/opt/ml/processing/train/train.csv', index=False)

pd.concat([X\_val, y\_val], axis=1).to\_csv('/opt/ml/processing/validation/validation.csv', index=False)

pd.concat([X\_test, y\_test], axis=1).to\_csv('/opt/ml/processing/test/test.csv', index=False)

print('Preprocessing completed.')

**Step 3: Model Training**

We'll use SageMaker's built-in XGBoost algorithm to train our model.

python

Copy

from sagemaker.xgboost import XGBoost

*# Set up the estimator*

xgb = XGBoost(

entry\_point='train.py',

role=role,

instance\_count=1,

instance\_type='ml.m5.xlarge',

framework\_version='1.0-1',

output\_path=f's3://{bucket\_name}/model\_output',

hyperparameters={

'max\_depth': 5,

'eta': 0.2,

'gamma': 4,

'min\_child\_weight': 6,

'subsample': 0.8,

'objective': 'binary:logistic',

'num\_round': 100

}

)

*# Train the model*

xgb.fit({

'train': f's3://{bucket\_name}/churn\_data/train',

'validation': f's3://{bucket\_name}/churn\_data/validation'

})

print("Model training completed.")

Create a file named train.py with the following content:

python

Copy

import argparse

import os

import pandas as pd

import xgboost as xgb

if \_\_name\_\_ =='\_\_main\_\_':

parser = argparse.ArgumentParser()

*# Hyperparameters are described here*

parser.add\_argument('--num\_round', type=int, default=999)

parser.add\_argument('--max\_depth', type=int, default=3)

parser.add\_argument('--eta', type=float, default=0.1)

*# SageMaker specific arguments. Defaults are set in the environment variables.*

parser.add\_argument('--model\_dir', type=str, default=os.environ.get('SM\_MODEL\_DIR'))

parser.add\_argument('--train', type=str, default=os.environ.get('SM\_CHANNEL\_TRAIN'))

parser.add\_argument('--validation', type=str, default=os.environ.get('SM\_CHANNEL\_VALIDATION'))

args = parser.parse\_args()

train = pd.read\_csv(f'{args.train}/train.csv')

validation = pd.read\_csv(f'{args.validation}/validation.csv')

X\_train = train.drop('churn', axis=1)

y\_train = train['churn']

X\_validation = validation.drop('churn', axis=1)

y\_validation = validation['churn']

dtrain = xgb.DMatrix(X\_train, label=y\_train)

dvalidation = xgb.DMatrix(X\_validation, label=y\_validation)

params = {

'max\_depth': args.max\_depth,

'eta': args.eta,

'objective': 'binary:logistic'

}

model = xgb.train(params, dtrain, args.num\_round, evals=[(dvalidation, 'validation')])

model.save\_model(f'{args.model\_dir}/xgboost-model')

**Step 4: Model Deployment**

Now, let's deploy our trained model to a SageMaker endpoint.

python

Copy

predictor = xgb.deploy(initial\_instance\_count=1, instance\_type='ml.t2.medium')

print(f"Model deployed. Endpoint name: {predictor.endpoint\_name}")

**Step 5: Inference and Evaluation**

We'll use our deployed model to make predictions on the test set and evaluate its performance.

python

Copy

import io

import numpy as np

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score

import matplotlib.pyplot as plt

import seaborn as sns

*# Load test data*

test\_data = pd.read\_csv(f's3://{bucket\_name}/churn\_data/test/test.csv')

X\_test = test\_data.drop('churn', axis=1)

y\_test = test\_data['churn']

*# Make predictions*

predictions = predictor.predict(X\_test.values)

*# Convert raw predictions to binary predictions*

y\_pred = (predictions > 0.5).astype(int)

*# Calculate metrics*

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)

auc\_roc = roc\_auc\_score(y\_test, predictions)

print(f"Accuracy: {accuracy:.4f}")

print(f"Precision: {precision:.4f}")

print(f"Recall: {recall:.4f}")

print(f"F1 Score: {f1:.4f}")

print(f"AUC-ROC: {auc\_roc:.4f}")

*# Visualize feature importance*

feature\_importance = predictor.predict(X\_test.values, initial\_args={'pred\_contribs': 'True'})

feature\_names = X\_test.columns

plt.figure(figsize=(10, 6))

sns.barplot(x=np.mean(feature\_importance, axis=0), y=feature\_names)

plt.title('Feature Importance')

plt.xlabel('Mean SHAP Value')

plt.tight\_layout()

*# Save plot to S3*

img\_data = io.BytesIO()

plt.savefig(img\_data, format='png')

img\_data.seek(0)

s3.put\_object(Body=img\_data, Bucket=bucket\_name, Key='churn\_prediction/feature\_importance.png')

print(f"Feature importance plot saved to s3://{bucket\_name}/churn\_prediction/feature\_importance.png")

*# Clean up*

predictor.delete\_endpoint()

**Conclusion**

This end-to-end example demonstrates how to:

1. Prepare and upload data to S3
2. Preprocess data using SageMaker Processing
3. Train a model using SageMaker's built-in XGBoost algorithm
4. Deploy the trained model to a SageMaker endpoint
5. Make predictions and evaluate the model's performance
6. Visualize feature importance and save the plot to S3

Key points to remember:

* Replace the sample data with your own dataset for real-world applications
* Adjust hyperparameters and model architecture based on your specific use case
* Consider using SageMaker Experiments to track multiple training runs
* Implement proper error handling and logging for production environments
* Set up monitoring for the deployed model to track its performance over time

By following this workflow, you can build and deploy machine learning models on AWS for various business problems, leveraging the scalability and managed services provided by the AWS ecosystem.