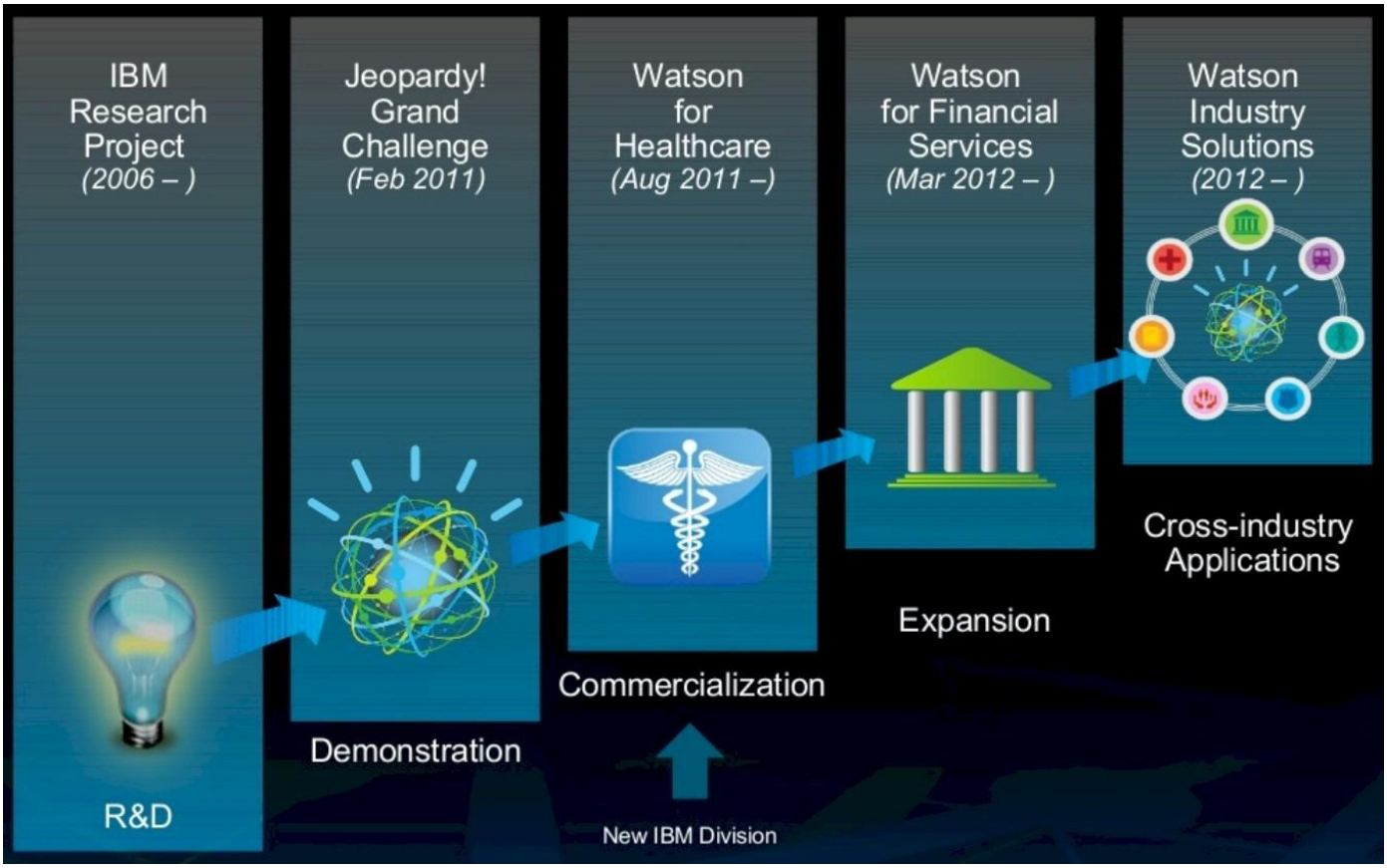
Machine Learning Model Deployment with IBM Cloud Watson Studio

Project Title: ML Models With IBM Watson

Phase 4: Development Part2

Continue building the project by deploying the model and integrating it into applications



Introduction:

» Deploying a trained model as a web service in IBM Cloud Watson Studio is a powerful way to make your machine learning models accessible and usable by a wider audience. By providing an API endpoint, you can seamlessly integrate the deployed model into various applications, allowing them to leverage the model's predictive capabilities.

» The process of deploying a machine learning model in IBM Cloud Watson Studio, highlighting the steps involved in hosting your model in the cloud. Once deployed, we will discuss how to use the provided API endpoint to access the model's predictions from within your applications.

» This integration is particularly valuable for applications that require real-time analysis, such as recommendation engines, sentiment analysis, fraud detection, and more. By following this guide, you will learn how to take your machine learning projects to the next level by making them readily available to other developers and stakeholders through a user-friendly API.

Dataset:

	0	1	2	3	4
customerID	7590-VHVEG	5575-GNVDE	3668-QPYBK	7795-CFOCW	9237-HQITU
gender	Female	Male	Male	Male	Female
SeniorCitizen	0	0	0	0	0
Partner	Yes	No	No	No	No
Dependents	No	No	No	No	No
tenure	1	34	2	45	2
PhoneService	No	Yes	Yes	No	Yes
MultipleLines	No phone service	No	No	No phone service	No
InternetService	DSL	DSL	DSL	DSL	Fiber optic
OnlineSecurity	No	Yes	Yes	Yes	No
OnlineBackup	Yes	No	Yes	No	No
DeviceProtection	No	Yes	No	Yes	No
TechSupport	No	No	No	Yes	No
StreamingTV	No	No	No	No	No
StreamingMovies	No	No	No	No	No
Contract	Month-to-month	One year	Month-to-month	One year	Month-to-month
PaperlessBilling	Yes	No	Yes	No	Yes
PaymentMethod	Electronic check	Mailed check	Mailed check	Bank transfer (automatic)	Electronic check
MonthlyCharges	29.85	56.95	53.85	42.3	70.7
TotalCharges	29.85	1889.5	108.15	1840.75	151.65
Churn	No	No	Yes	No	Yes

Deploy and test your model:

Necessary Steps to fillow:

Import data:

```
import pandas as pd
import numpy as np
import missingno as msno
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')
```

Test the Model:

Testing the model before deploying it is a critical step in the machine learning development lifecycle. Testing the model involves evaluating its performance using a separate dataset (not used during training) to ensure that it generalizes well to new, unseen data. The goal is to assess how well the model is expected to perform in a real-world scenario.

Program:

```
input_data = preprocess_data(sample_data)
predictions = model.predict(input_data)
predicted_class = predictions.argmax(axis=-1)
print("Predicted Class:", predicted_class)
print("Predicted Probabilities:", predictions)
actual_labels = [1, 0, 1, ...] # Actual labels corresponding to the sample data
from sklearn.metrics import accuracy_score, mean_squared_error
accuracy = accuracy_score(actual_labels, predicted_class)
print("Accuracy:", accuracy)
```

```
mse = mean_squared_error(actual_labels, predictions)
print("Mean Squared Error:", mse)
```

Prepare Data for API Calls:

Preparing data for API calls when deploying a trained model involves formatting the input data in a way that can be sent via HTTP requests to the API endpoint. Typically, the data needs to be serialized into a format like JSON or binary, depending on the requirements of the deployed model.

Program:

```
import json
input_data =
{
"feature1": 0.8,
"feature2": 1.2,
}

# Convert data to JSON format
json_data = json.dumps({"input": input_data})
```

Monitor and Manage the Deployed Service:

Monitoring and managing the deployed service after deploying the trained model is crucial to ensure its performance, availability, and security.

Program:

```
import requests
import time
import logging
logging.basicConfig(filename='service_logs.log',
level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s')
def make_api_request(data):
try:
response = requests.post(API_ENDPOINT, json=data)
response.raise_for_status() # Raise HTTPError for bad
requests
return response.json()
except requests.exceptions.HTTPError as errh:
logging.error(f"HTTP Error: {errh}")
```

```
except requests.exceptions.ConnectionError as errc:
logging.error(f"Error Connecting: {errc}")
except requests.exceptions.Timeout as errt:
logging.error(f"Timeout Error: {errt}")
except requests.exceptions.RequestException as err:
logging.error(f"Something went wrong: {err}")
while True:
input_data = {"input": "sample_input_data"}
start_time = time.time()
response = make_api_request(input_data)
response_time = time.time() - start_time
logging.info(f"API Response Time: {response_time} seconds")
if response is not None and "error" in response:
logging.error(f"API Error: {response['error']}")
```

Deploy the Model and Deploy as Web Service:

Deploying a machine learning model as a web service involves multiple steps, including loading the model, creating an API, and handling incoming requests. Here, I'll provide you with a basic example using Flask, a popular Python web framework, to deploy a machine learning model as a web service.

Program:

```
pip install Flask
from flask import Flask, request, jsonify
import joblib
app = Flask(__name__)
model = joblib.load("model.pkl")
try:
data = request.get_json(force=True)
features = data['input']
prediction = model.predict([features])[0]
return jsonify(prediction=prediction)
except Exception as e:
return jsonify(error=str(e))
if __name__ == '__main__':
app.run(port=5000, debug=True)
```

Integrate the Deployed Model into Applications:

Integrating a deployed machine learning model into applications using an API endpoint involves a series of steps.

Get API Endpoint:

Once the model is deployed, you'll get an API endpoint. This endpoint is the URL where your application can send requests to make predictions.

Program:

```
import requests
def get_api_endpoint():
response =
requests.get("https://api.example.com/get_endpoint")
if response.status_code == 200:
api_endpoint = response.json().get("endpoint")
return api_endpoint
```

```
else:
print("Failed to retrieve API endpoint.")
return None
def make_predictions(api_endpoint, input_data):
try:
response = requests.post(api_endpoint, json={"input":
input_data})
predictions = response.json()
print("Predictions:", predictions)
except Exception as e:
print("Error:", e)
api_endpoint = get_api_endpoint()
input_data = {"feature1": 0.8, "feature2": 1.2, "feature3":
0.5}
if api_endpoint:
make_predictions(api_endpoint, input_data)
```

API Documentation:

API Documentation refers to a set of guidelines, instructions, and information provided by developers or organizations that describe how to interact with their application programming interface (API).

Program:

```
import requests
API_ENDPOINT = 'YOUR_API_ENDPOINT'
input_data =
{
'feature1': 0.2,
'feature2': 0.8,
}
try:
response = requests.post(API_ENDPOINT,
json=input_data)
if response.status_code == 200:
result = response.json()
print('Model prediction:', result)
else:
print('Error:', response.status_code)
except Exception as e:
print('An error occurred:', str(e))
```

Integrate into Applications:

Integrating into applications using a provided API endpoint refers to the process of incorporating external services, functionalities, or data into your own software application by leveraging the capabilities

exposed through an Application Programming Interface (API) endpoint.

Program:

```
import requests
API_ENDPOINT = 'YOUR_API_ENDPOINT'
input_data =
{
'feature1': 0.2,
'feature2': 0.8,
}
try:
response = requests.post(API_ENDPOINT,
json=input_data)
if response.status_code == 200:
result = response.json()

print('Model prediction:', result)
else:
print('Error:', response.status_code)
except Exception as e:
print('An error occurred:', str(e))
```

Conclusion:

Deploying machine learning models as web services on cloud platforms and integrating them into applications through API endpoints empower businesses with powerful tools to drive innovation, improve efficiency, and deliver exceptional user experiences. By harnessing the capabilities of cloud-based machine learning services, businesses can stay agile, make data-driven decisions, and remain competitive in the rapidly evolving digital landscape.