PHASE-1_SUBMISSION_CLOUD-COMPUTING_GROUP-3

ML MODELS WITH IBM WATSON

SUGUNA COLLEGE OF ENGINEERING (7150)

TEAM MEMBERS:

MUSTAFA GOLWALA

KARTHICK BALAJI.R

MOHAN RAJ.R

Project Title: ML Models with IBM Watson

Problem Statement: Become a wizard of predictive analytics with IBM Cloud Watson Studio. Train machine learning models to predict outcomes in real-time. Deploy the models as web services and integrate them into your applications. Unlock the magic of data-driven insights and make informed decisions like never before!

Project Steps

Phase 1: Problem Definition and Design Thinking

Problem Definition: The project involves training a machine learning model using IBM Cloud Watson Studio and deploying it as a web service. The goal is to become proficient in predictive analytics by creating a model that can predict outcomes in real-time. The project encompasses defining the predictive use case, selecting a suitable dataset, training a machine learning model, deploying the model as a web service, and integrating it into applications.

i)PREDICTIVE USE CASE:

1. Empathize: Begin by understanding the problem deeply. Interview stakeholders, gather historical data, and get a sense of their pain points related to predictive analytics. For example, talk to the marketing team to understand their challenges with predicting customer churn.

2. Define: Clearly define the problem you want to address with predictive analytics. For instance, if you're focused on customer churn, the problem might be: "How can we reduce customer churn by proactively identifying at-risk customers?"

3. Ideate: Generate a wide range of ideas for predictive solutions. Consider various machine learning algorithms and data sources. Think about what data points might be indicative of churn, such as customer behaviour, usage patterns, and feedback.

4. Prototype: Create initial prototypes of your predictive model using Watson Studio. Experiment with different algorithms and feature sets. Test the model on historical data to evaluate its performance.

5. Test: Assess the performance of your prototype model. Use metrics like accuracy, precision, and recall gauging, its effectiveness in predicting customer churn. Gather feedback from stakeholders and make necessary improvements.

6. Iterate: Based on the feedback and test results, refine the model and iterate through the prototyping and testing phases. Continue until you have a model that reliably predicts customer churn.

7. Implement: Once you have a well-performing predictive model, integrate it into your business processes. Ensure that the predictions can be used in a practical and meaningful way to reduce churn.

8. Monitor: Continuously monitor the predictive model's performance in a real-world setting. Set up alerts for significant changes in customer churn patterns and refine the model as necessary.

9. Feedback Loop: Maintain an ongoing feedback loop with stakeholders and end-users. Gather insights on the model's impact on reducing churn and adjust as needed.

10. Scale: If the predictive analytics solution proves successful in reducing churn, consider scaling it to address other predictive use cases, such as product demand forecasting.

ii)DATASET SELECTION:

Dataset: Customer Churn Data

Source: You can obtain customer churn data from various sources, including:

1. Kaggle: Kaggle offers a wide range of datasets, including customer churn data. You can search for datasets related to telecom or subscription-based services, which often have customer churn information.

2. Your Organization: If you're working on a project within a company, consider using your own customer data. This will be highly relevant and specific to your business.

3. Synthetic Data: If real data is not available or can't be used due to privacy concerns, you can generate synthetic customer churn data. Tools like Faker or libraries in Python can help you create such data.

iii)MODEL TRAINING:

Algorithm: Random Forest

Description: Random Forest is an ensemble learning method that combines multiple decision trees to create a robust and accurate predictive model. It's particularly effective for classification tasks, such as predicting customer churn. Some key benefits of using Random Forest include its ability to handle large datasets, deal with missing values, and reduce the risk of overfitting.

```
# Load and preprocess your dataset

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Train a Random Forest model
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

iv)MODEL DEPLOYMENT:

1. Prepare the Model: Ensure that you have a trained and well-validated machine learning model in Watson Studio. You should have saved this model and all the necessary dependencies.

2. Create a Deployment Space: In Watson Studio, navigate to your project and create a deployment space if you haven't already. Deployment spaces help organize your deployed models and services.

3. Deploy the Model:

- From your project, go to the "Models" section.

- Select the model you want to deploy.

- Click on "Deploy" or a similar button, depending on the Watson Studio interface.

- Follow the deployment wizard, which will guide you through the deployment process.

4. Configure Deployment:

- Choose the runtime environment for your deployment. You can select options like IBM Cloud, Kubernetes, or AutoAI.

- Specify the resources you want to allocate for the deployment, including the number of nodes and CPU/memory settings.

- Configure any necessary authentication and security settings.

5. Create an Endpoint: Once the deployment is complete, you'll receive an API endpoint URL. This is the URL that you'll use to interact with your model through web services.

6. Test the Endpoint: Before using the deployed model in production, it's a good practice to test the endpoint to ensure it's working correctly. You can do this from Watson Studio or using API testing tools.

7. Integrate into Applications: You can now integrate the API endpoint into your applications or services. This allows you to make predictions in real-time by sending data to the deployed model.

8. Monitor and Maintain: Regularly monitor the deployed model's performance and retrain it if necessary. Set up alerts for any anomalies or issues.

9. Scale as Needed: Depending on the usage and demand, you can scale the deployment resources up or down to ensure optimal performance.

10. Document the API: Provide documentation on how to use the API, including input data format, expected output, and any necessary authentication details.

```
from ibm_watson_machine_learning import APIClient

wml_credentials = {
    "apikey": "your_api_key",
    "url": "https://us-south.ml.cloud.ibm.com"
}
wml_client = APIClient(wml_credentials)
Code Suggestions

space_id = "your_space_id"

deployment = wml_client.deployments.create(model_id="your_model_id",name="your_deployment_name",space_id=space_id)
```

v)INTEGRATION:

1. Obtain the API Endpoint: When you deploy your model using IBM Cloud Watson Studio, you'll receive an API endpoint URL. This is the address where you'll send data for predictions.

2.API Access: Ensure that your application or system has access to the internet and can make HTTP requests to the API endpoint. If there are authentication requirements, such as API keys or tokens, make sure your application includes these in the requests.

3.Data Preparation: Prepare the input data that you want to send to the model for prediction. The data should be structured according to the model's input requirements, including data format, data types, and any necessary preprocessing.

4. Send HTTP Requests: Use programming libraries or tools to send HTTP POST requests to the model's API endpoint. This involves including the input data in the request body.

5. Receive Predictions: After sending the request, you'll receive a response from the model's API. This response will contain the predictions generated by the model based on the input data.

6. Parse and Use Predictions: Extract and parse the predictions from the API response in your application. Depending on your use case, you might store the predictions, display them to users, or use them for decision-making within your application.

7. Error Handling: Implement error-handling mechanisms to handle cases where the API request fails or returns errors. This can include checking for response status codes and handling exceptions gracefully.

8. Performance Optimization: Optimize the integration for performance by batching multiple requests, caching predictions if necessary, and monitoring the response times from the deployed model.

9. Security: Ensure that data sent to and received from the model is handled securely. Use encryption and follow best practices for securing API interactions.

10. Monitoring and Maintenance: Regularly monitor the integration to ensure that it continues to work as expected. Set up alerts for any issues with the deployed model or API endpoint.

11. Documentation: Document the integration process for your team and any future developers. Include information on how to use the API, expected input and output formats, and any specific considerations.

```python
import requests
import json

# Define your input data (modify for your specific data)
input_data = {
    "feature1": value1,
    "feature2": value2,
    # Add all the required features
}
```
Code Suggestions
```python

# Define the API endpoint URL
endpoint_url = "your_model_api_endpoint_url"

# Send a POST request to the model
response = requests.post(endpoint_url, json=input_data)
```
Code Suggestions
```python

# Parse the predictions from the response
if response.status_code == 200:
    predictions = json.loads(response.text)
    # Use the predictions in your application
else:
    print("Error: Failed to get predictions")
```