Title: Machine Learning for Assessing Soil Liquefaction Risk in Seismic Zone

Machine Learning Project Documentation

Deployment

1. Overview

The deployment phase involves making the trained machine learning model available for real-world or production use. This includes saving the model in a format that can be easily loaded, setting up an environment where the model can accept input and produce predictions, and integrating security and monitoring mechanisms. The goal is to ensure that the model can be reliably accessed and used by end-users or other systems while maintaining performance and security.

Model_13, which demonstrated the best overall performance due to its strong metrics across validation and test sets, was chosen for deployment. This model benefited significantly from L2 regularization, which enhanced its generalization capabilities. The deployment process involved setting up an interactive web interface using Gradio, a user-friendly library for building machine learning model interfaces.

Deployment Steps: (Code 1)

1. **Model Saving and Loading:**
    - Model_13 was saved as an HDF5 file (model_13.h5), which includes the architecture, weights, and training configuration. This allows for easy loading and reuse of the model in different environments.
2. **Custom Objects Handling:**
    - The model included a custom activation function (LeakyReLU). This required registering custom objects globally to ensure they are recognized when loading the model.
3. **Scaler Initialization:**
    - The StandardScaler used for normalizing the input data during training was initialized. It is crucial to fit the scaler with the same statistics used during training to ensure consistency.
4. **Gradio Interface Creation:**
    - Gradio was used to create an interactive interface for users to input the required features and receive predictions from the model. This interface was set up to handle inputs for z(m), dw(m), svo(kPa), s'vo(kPa), Mw, amax(g), and qc1Ncs, and return a binary classification of "Liquefaction" or "No Liquefaction".(see Figure 1)

Integration with Other Systems:

The Gradio interface can be integrated into web applications or shared with stakeholders as a standalone application. This facilitates easy access and usability for non-technical users who need to make liquefaction predictions based on the model.

2. Model Serialization

Process:

- **Model Saving:** The trained model, Model_13, was saved using TensorFlow/Keras's model.save() function, which stores the model in the HDF5 format (model_13.h5). This format includes the model architecture, weights, and optimizer state, making it easy to reload the model for inference.
- **Custom Objects Handling:** The model uses a custom activation function (LeakyReLU), which required registering custom objects during the saving and loading process to ensure they are properly recognized.

Considerations:

- **Efficiency:** The HDF5 format is efficient for storing complex models, providing a balance between file size and speed of loading.
- **Portability:** HDF5 is widely supported and allows the model to be moved across different environments without compatibility issues.

3. Model Serving

Process:

- **Platform Choice:** The model is served using Gradio, a library for creating web-based interfaces for machine learning models. Gradio is chosen for its simplicity and ability to quickly deploy interactive interfaces.
- **Environment Setup:** The deployment environment includes loading the serialized model and initializing the necessary preprocessing steps, such as data normalization using StandardScaler.

Deployment Platform:

- **Local Deployment:** The initial deployment is done locally, but Gradio interfaces can be easily deployed on cloud platforms like AWS, GCP, or Heroku for broader access.

4. API Integration

Process:

- **API Endpoints:** The Gradio interface serves as an API endpoint where users can input data and receive predictions. Each input field corresponds to a feature required by the model.
- **Input/Output Formats:** The inputs are taken as numerical values representing the features (`z(m)`, `dw(m)`, `svo(kPa)`, `s'vo(kPa)`, `Mw`, `amax(g)`, `qc1Ncs`). The output is a textual prediction ("Liquefaction" or "No Liquefaction").

Implementation:

- **Interface Creation:** Gradio creates a simple web interface with the defined input and output formats.

```python
import gradio as gr
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
import pandas as pd
from keras.layers import LeakyReLU
from keras.models import load_model

# Define the custom objects
custom_objects = {
    'LeakyReLU': LeakyReLU
}

# Register custom objects globally
keras.utils.get_custom_objects().update(custom_objects)

# Load the trained model with custom objects
model = keras.models.load_model('model_13.h5', custom_objects=custom_objects)

# Initialize the scaler (you may need to fit it again if you didn't save it)
# Assuming the scaler was fitted on training data and we have the same training data or stats
scaler = StandardScaler()
# Placeholder data for fitting the scaler (Use your actual training data here)
data = pd.DataFrame({
    'z(m)': [0], 'dw(m)': [0], 'svo(kPa)': [0], 's\'vo(kPa)': [0],
    'Mw': [0], 'amax(g)': [0], 'qc1Ncs': [0]
})
scaler.fit(data)  # Fit the scaler with your actual training data

def predict_liquefaction(z, dw, svo, svo_prime, Mw, amax, qc1Ncs):
    # Create a DataFrame with the input data
    input_data = pd.DataFrame({
        'z(m)': [z], 'dw(m)': [dw], 'svo(kPa)': [svo], 's\'vo(kPa)': [svo_prime],
        'Mw': [Mw], 'amax(g)': [amax], 'qc1Ncs': [qc1Ncs]
    })

    # Normalize the input data using the fitted scaler
    input_scaled = scaler.transform(input_data)

    # Make a prediction using the model
    prediction = model.predict(input_scaled)

    # Since the output is a probability, we can set a threshold (e.g., 0.5) for binary classification
    prediction_label = 'Liquefaction' if prediction >= 0.5 else 'No Liquefaction'

    return prediction_label

# Create the Gradio interface
interface = gr.Interface(
    fn=predict_liquefaction,
    inputs=[
        gr.inputs.Number(label="z(m)"),
        gr.inputs.Number(label="dw(m)"),
        gr.inputs.Number(label="svo(kPa)"),
        gr.inputs.Number(label="s'vo(kPa)"),
        gr.inputs.Number(label="Mw"),
        gr.inputs.Number(label="amax(g)"),
        gr.inputs.Number(label="qc1Ncs")
    ],
    outputs=gr.outputs.Textbox(label="Prediction"),
    title="Liquefaction Prediction",
    description="Predict liquefaction based on input parameters."
)

# Launch the interface
interface.launch()
```

Code 1 deployment with gradio

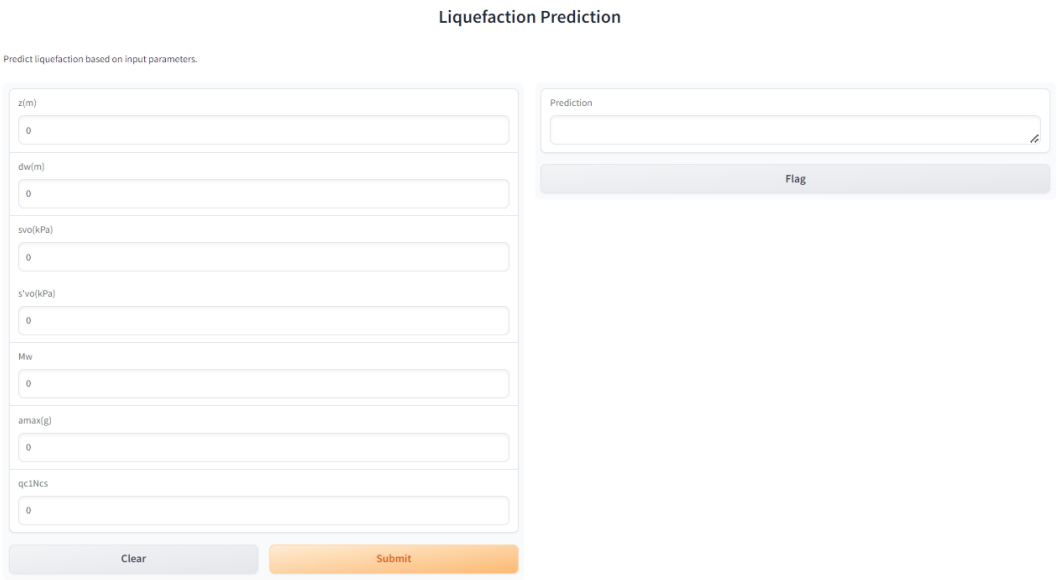

Figure 1 user interface for liquefaction prediction