CPE 490 590: Machine Learning for Engineering Applications

12 Machine Learning Model Deployment

Rahul Bhadani

Electrical & Computer Engineering, The University of Alabama in Huntsville

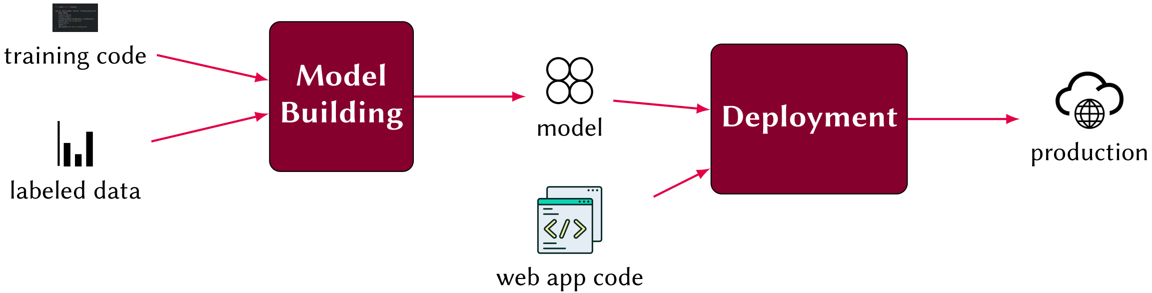
Outline

Motivation

Taking Models into Production

Take your trained model into a different environment where we can make predictions or inferences on the incoming data.

Machine Learning Pipeline



Deployment of a Model

- Real-time or batch predictions
- E.g. Flag for anomaly: fraudalent credit card transaction; unexpected power load in the electric grid

PyTorch Model Creation

Creating a PyTorch Model for IRIS Classification

- Create a simple neural network model using PyTorch
- Train it on the IRIS dataset
- Export the trained model

PyTorch Model - Imports and Data Loading

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import onnx
import onnxruntime as ort
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Convert to PyTorch tensors
X_train_tensor = torch.FloatTensor(X_train)
y_train_tensor = torch.LongTensor(y_train)
X_test_tensor = torch.FloatTensor(X_test)
y_test_tensor = torch.LongTensor(y_test)
```

PyTorch Model - Model Definition and Training

```
class IrisClassifier(nn.Module):
    def __init__(self):
        super(IrisClassifier, self).__init__()
        self.layer1 = nn.Linear(4, 10)
        self.layer2 = nn.Linear(10, 3)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        return x

model = IrisClassifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
epochs = 100
for epoch in range(epochs):
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    if (epoch+1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')
```

PyTorch Model - Evaluation and ONNX Export

```python
# Evaluate the model
with torch.no_grad():
    outputs = model(X_test_tensor)
    _, predicted = torch.max(outputs.data, 1)
    accuracy = (predicted == y_test_tensor).sum().item() / y_test_tensor.size(0)
    print(f'Test Accuracy: {accuracy:.4f}')

# Export the model to ONNX
dummy_input = torch.randn(1, 4)
input_names = ["input"]
output_names = ["output"]

torch.onnx.export(
    model, dummy_input, "iris_classifier.onnx",
    input_names=input_names, output_names=output_names,
    dynamic_axes={"input": {0: "batch_size"}, "output": {0: "batch_size"}},
)
print("Model saved as iris_classifier.onnx")
```

PyTorch Model - Saving Scaler Parameters and Testing ONNX

```python
# Save scaler mean and scale for later use in the Production application
import json
with open("scaler_params.json", "w") as f:
    json.dump({ "mean": scaler.mean_.tolist(), "scale": scaler.scale_.tolist()
    }, f)

print("Scaler parameters saved as scaler_params.json")
# Test the ONNX model
ort_session = ort.InferenceSession("iris_classifier.onnx")
# Prepare input
ort_inputs = {ort_session.get_inputs()[0].name: X_test.astype(np.float32)}
# Run inference
ort_outputs = ort_session.run(None, ort_inputs)
# Calculate accuracy
ort_predicted = np.argmax(ort_outputs[0], axis=1)
ort_accuracy = np.sum(ort_predicted == y_test) / len(y_test)
print(f'ONNX Model Test Accuracy: {ort_accuracy:.4f}')
```

PyTorch Model - Saving Example Data

```python
# Save some example data for testing in Production (unscaled Data)
examples = []
class_names = iris.target_names
for i in range(10):
    idx = np.random.randint(0, len(X))
    examples.append({
        "features": X[idx].tolist(),
        "label": int(y[idx]),
        "class_name": class_names[y[idx]]
    })

with open("example_data.json", "w") as f:
    json.dump(examples, f)

print("Example data saved as example_data.json")
```

Web App Development

Basic Setup

Create a directory

```
mkdir -p ~/iris_classifier_app
cd ~/iris_classifier_app
```

Python Environment and Package Installation

Create a directory

```
# Create virtual environment
python3 -m venv venv

# Activate virtual environment
source venv/bin/activate
pip install flask numpy onnxruntime pandas scikit-learn gunicorn
```

Flask for Model Deployment

https://flask.palletsprojects.com/en/stable/

- Flask is microframework written in Python for web applications.
- Here, micro means simple but extensible.

Basic Layout of Web App

```
~/iris_classifier_app
            |-- app.py
            |-- example_data.json
            |-- iris_classifier.onnx
            |-- scaler_params.json
            |-- static
            |   |-- css
            |   |   `-- styles.css
            |   `-- js
            |       `-- script.js
            `-- templates
                `-- index.html
```

app.py

In Flask, `app.py` typically serves as the main entry point for your web application. It's where you initialize your Flask application instance, define routes, and configure other settings.

`cd ~/iris_classifier_app` and create a while called `app.py`. The content of the `app.py`:

https://github.com/rahulbhadani/iris_classifier_app/blob/main/app.py

Explaining `app.py`

```python
@app.route('/')
def home():
    return render_template('index.html', examples=example_data)
```

- `@app.route('/')` is decorator in Python that associates the URL path / (the root or home page of the application) with the function that follows (`home` in this case).
- `render_template` renders html template file `index.html` and passes the variable `example_data` to the template.

Explaining `app.py`

```python
@app.route('/predict', methods=['POST'])
def make_prediction():
    data = request.get_json()
    features = np.array([data['features']], dtype=np.float32)

    try:
        result = predict(features)
        return jsonify(result)
    except Exception as e:
        return jsonify({"error": str(e)}), 400
```

- `@app.route('/predict', methods=['POST'])` is decorator in Python that associates the URL path `/predict` with the function that follows (`make_prediction` in this case). The `methods=['POST']` argument specifies that this route only accepts POST requests. This is typically used when the client sends data to the server (e.g., form data or JSON).
- `make_prediction` is responsible for handling POST requests to the `/predict` route. It processes the incoming data, makes a prediction, and returns the result.
- `request` is a Flask object that contains the data sent by the client in the request.

Explaining app.py

```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

- This starts the Flask development server with specific configurations.

HTML for the Homepage

https://github.com/rahulbhadani/iris_classifier_app/blob/main/templates/index.html

Template Integration in Flask

HTML uses jinja2 templating convention.

```
<link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
```

- `url_for('static', filename='...')` generates URLs for static files
- Flask automatically looks for templates in the `templates/` directory
- Templates are rendered using `render_template()`

Dynamic Content Generation

Template shows dynamic content generation with Jinja2:

```
{% for example in examples %}
<div class="example-card" data-features="{{ example.features }}">
    <h3>{{ example.class_name }}</h3>
    <ul>
        <li>Sepal Length: {{ "%.1f"|format(example.features[0]) }} cm</li>
        <!-- More features... -->
    </ul>
    <button class="use-example-btn">Use These Values</button>
</div>
{% endfor %}
```

- `% for %` loops through data passed from Python
- `{ variable }` displays values
- `{ "%.1f"|format(value) }` formats numeric values

JavaScript for Client-Side Interaction

The HTML template references a JavaScript file:

```
<script src="{{ url_for('static', filename='js/script.js') }}"></script>
```

This script handles:

- Form submission via AJAX
- Updating prediction results
- Rendering the prediction chart
- Handling “Use These Values” button clicks

Client-Side Chart Visualization

Chart.js is used for visualization:

```
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<!-- ... -->
<div class="chart-container">
    <canvas id="probability-chart"></canvas>
</div>
```

JavaScript would create a bar chart of prediction probabilities

CSS for Styling

`/css/styles.css` is used for styling the HMTL.

Explaining Javascript: DOM Initialization

The script starts by getting references to HTML elements:

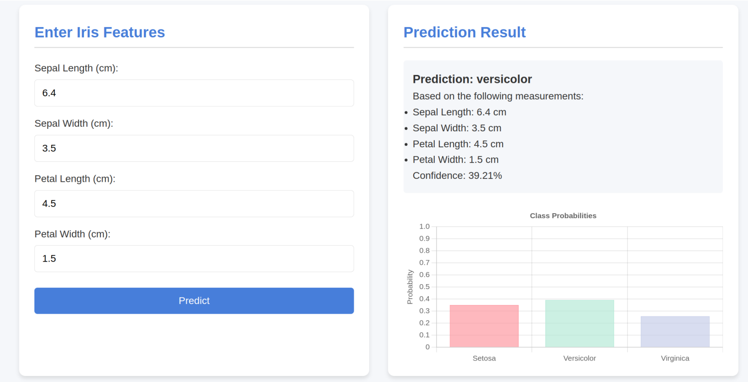
```
document.addEventListener('DOMContentLoaded', function() {
    // Get DOM elements
    const predictionForm = document.getElementById('prediction-form');
    const sepalLengthInput = document.getElementById('sepal-length');
    const sepalWidthInput = document.getElementById('sepal-width');
    const petalLengthInput = document.getElementById('petal-length');
    const petalWidthInput = document.getElementById('petal-width');
    const predictionResultDiv = document.getElementById('prediction-result');
    const exampleButtons = document.querySelectorAll('.use-example-btn');

});
```

- Wraps code in `DOMContentLoaded` event to ensure DOM is fully loaded
- Retrieves references to form elements and result display area
- Gets all example buttons using `querySelectorAll`

DOM = Document Object Model

Chart.js Integration

The script initializes a Chart.js bar chart for displaying prediction probabilities:



Form Submission Handling

The script handles form submission with a function to collect input values:

```
// Handle form submission
predictionForm.addEventListener('submit', function(e) { We.preventDefault();
    // Get input values
    const features = [parseFloat(sepalLengthInput.value), parseFloat(sepalWidthInput.value),
    ↪   parseFloat(petalLengthInput.value),parseFloat(petalWidthInput.value)];

    // Make prediction request
    fetch('/predict', { method: 'POST', headers: { 'Content-Type': 'application/json', }, body: JSON.stringify({ features:
    ↪   features }),
    })
    .then(response => response.json())
    .then(data => {
        // Display result
        displayPredictionResult(data, features);
        // Update chart
        updateProbabilityChart(data.probabilities);
    })
    .catch(error => { predictionResultDiv.innerHTML = `<p>Error: $-error.message"</p>`;
    });
});
```

API Interaction

Key aspects of the API interaction:

- Prevents default form submission with `e.preventDefault()`
- Collects form values and converts to numbers with `parseFloat()`
- Uses `fetch()` API for asynchronous request to the server
- Sends data as JSON with proper headers
- Handles response with Promise chaining
- Calls helper functions to update UI with results
- Includes error handling

Example Data Integration

The script enables users to populate the form with example data:

```
// Handle example buttons
exampleButtons.forEach(button => {
    button.addEventListener('click', function() {
        const features = JSON.parse(this.parentElement.dataset.features);
        // Fill form with example values
        sepalLengthInput.value = features[0];
        sepalWidthInput.value = features[1];
        petalLengthInput.value = features[2];
        petalWidthInput.value = features[3];
        // Submit form
        predictionForm.dispatchEvent(new Event('submit'));
    });
});
```

- Retrieves feature values from HTML data attribute
- Populates form fields with example values
- Triggers form submission programmatically

Displaying Prediction Results

Helper function to update the UI with prediction results:

```
// Function to display prediction result
function displayPredictionResult(data, features) {
    const result = `
        <h3>Prediction: <span class="prediction-class">${data.class_name}</span></h3>
        <p>Based on the following measurements:</p>
        <ul><li>Sepal Length: ${features[0]} cm</li><li>Sepal Width: ${features[1]} cm</li><li>Petal Length: ${features[2]} cm</li><li>Petal Width: ${features[3]} cm</li></ul>
        <p>Confidence: ${(data.probabilities[data.class_name] * 100).toFixed(2)}%</p>
    `;
    predictionResultDiv.innerHTML = result;
}
```

- Uses template literals for clean HTML construction
- Displays prediction result and input features
- Formats confidence percentage with two decimal places

Updating the Chart

Function to update the probability chart with new data:

```
// Function to update probability chart
function updateProbabilityChart(probabilities) {
    const data = [
        probabilities.setosa || 0,
        probabilities.versicolor || 0,
        probabilities.virginica || 0
    ];

    probabilityChart.data.datasets[0].data = data;
    probabilityChart.update();
}
```

- Extracts probability values for each class
- Uses `|| 0` to provide default values if undefined
- Updates chart data
- Calls `chart.update()` to render changes

Data Flow Between Client and Server

- Client collects input data from form
- Data is sent to server via `fetch()` API
- Server processes the request and runs the ML model
- Server sends back prediction results as JSON
- Client updates UI with results

Key JavaScript Concepts Used

- Event Handling:
  - `addEventListener` for form submission and button clicks
  - `DOMContentLoaded` for initialization
- DOM Manipulation:
  - `getElementById` and `querySelectorAll` for element selection
  - `innerHTML` for content updates
- Asynchronous Programming:
  - `fetch()` API for AJAX requests
  - Promise chaining with `.then()`
- Data Visualization:
  - Chart.js for interactive bar charts

Connection to Flask Backend

The JavaScript code interacts with a Flask endpoint

```
fetch('/predict', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify({ features: features }),
})
```

and the Flask route is

```
@app.route('/predict', methods=['POST'])
```

Complete Code

Python Notebook for Creating ONNX trained model

https://github.com/rahulbhadani/CPE490_590_Sp2025/blob/master/Code/Chapter_12_Machine_Learning_Model_Deployment.ipynb

Flask App

https://github.com/rahulbhadani/iris_classifier_app

The End