ML Model Development Report

APRIL 13

FAST NUCES
Authored by: Arsal Mairaj

Logo Name

Breast Cancer Classification Project Report

Introduction

This project focuses on building a machine learning pipeline to classify breast cancer tumors as Malignant or Benign using the Breast Cancer Wisconsin dataset. The project is divided into multiple tasks, including data preprocessing, model training with various Gradient Descent algorithms, model evaluation, logging with Weights & Biases (W&B), uploading models to Hugging Face, creating a Flask web UI, and setting up a GitHub repository for version control. This report details the implementation, challenges, and outcomes for each task (except Task 7, deployment).

Task 1: Load and Preprocess the Breast Cancer Dataset

Objective

Load the Breast Cancer Wisconsin dataset, preprocess it by scaling the features, and split it into training and validation sets for model training.

Implementation

The Breast Cancer Wisconsin dataset was loaded using scikit-learn's load_breast_cancer function. The dataset contains 569 samples, each with 30 numerical features (e.g., Mean Radius, Mean Texture, etc.) and a binary target variable (0 for Malignant, 1 for Benign).

Steps:

1. **Load the Dataset**: Load the dataset from sklearn.
2. **Split the Data**: The dataset was split into training (80%) and validation (20%) sets using train_test_split with a fixed random seed for reproducibility.
3. **Scale the Features**: Feature scaling was performed using StandardScaler to standardize the features (mean=0, standard deviation=1), which is essential for Gradient Descent-based algorithms like Logistic Regression. The scaler was fitted on the training data and applied to both training and validation sets.
4. **Save the Scaler**: The fitted scaler was saved as scaler.joblib for use in later tasks (e.g., inference in the Flask app).

Outcome

- The dataset was successfully loaded and split: 455 samples for training and 114 for validation.
- Features were scaled, ensuring compatibility with Gradient Descent algorithms.
- The scaler was saved for reuse in inference tasks.

Challenges

- Ensuring the scaler was only fitted on the training data to avoid data leakage into the validation set.
- Saving the scaler properly for later use in the Flask app.

Task 2: Uploading Model to Hugging Face

Objective

Train Logistic Regression models using different Gradient Descent methods (Batch, Stochastic, Mini-batch) and upload the models, scaler, and necessary files to a Hugging Face repository for public access.

Implementation

Three Logistic Regression models were trained using SGDClassifier with loss="log_loss" (equivalent to Logistic Regression) and different Gradient Descent strategies. The models and scaler were then uploaded to a Hugging Face repository.

Steps:

1. **Train the Models**:
    - **Batch Gradient Descent (bgd_model)**: Used SGDClassifier with default settings (equivalent to Batch Gradient Descent).
    - **Stochastic Gradient Descent (sgd_model)**: Used SGDClassifier with a small learning rate.
    - **Mini-batch Gradient Descent with Early Stopping (es_model)**: Used SGDClassifier with early_stopping=True to implement Mini-batch Gradient Descent and early stopping.
2. **Upload to Hugging Face**: The models (bgd_model.joblib, sgd_model.joblib, es_model.joblib) and the scaler (scaler.joblib) were uploaded to the Hugging Face repository arsalmairaj2k/breast-cancer-classification-models using the Hugging Face API.

3. **Documentation**: The Hugging Face repository includes a README documenting the models, scaler, and their usage. The repository URL is: https://huggingface.co/arsalmairaj2k/breast-cancer-classification-models.

Outcome

- Three models (bgd_model, sgd_model, es_model) and the scaler were successfully uploaded to Hugging Face.
- The repository is publicly accessible, and the files can be downloaded for inference (e.g., in the Flask app).
- The README provides clear instructions for using the models and scaler.

Challenges

- Initially, the scaler was removed from the repository, causing errors in the Flask app ("name 'scaler' is not defined"). This was resolved by re-uploading scaler.joblib.
- Ensuring proper authentication with Hugging Face API (required a token, which was set up correctly).

Task 3: Model Evaluation

Objective

Evaluate the performance of the trained models (bgd_model, sgd_model, es_model) on the validation set using appropriate metrics (accuracy, precision, recall, F1-score) and log the results to Weights & Biases (W&B).

Implementation

The models were evaluated on the validation set (X_test_scaled, y_test), and their performance metrics were logged to W&B for visualization and comparison.

Steps:

1. **Evaluate the Models**: The classification_report from scikit-learn was used to compute accuracy, precision, recall, and F1-score for each model on the validation set.
2. **Log to Weights & Biases**: W&B was initialized, and the evaluation metrics were logged for each model. Additionally, training metrics (e.g., loss over iterations) were logged during training.

3. **W&B Dashboard**: The W&B dashboard provides visualizations of the metrics, allowing comparison across the three models. The dashboard is accessible at: https://wandb.ai/arsalmairaj2k-fast-nuces/breast-cancer-classification.

Outcome

- **Performance Metrics**:
  - All models achieved high accuracy (~98%) on the validation set, indicating strong performance.
  - The es_model (Mini-batch Gradient Descent with Early Stopping) performed slightly better due to its ability to stop training when validation performance plateaued.
  - Precision, recall, and F1-scores were balanced for both classes (Malignant and Benign), with minimal false positives/negatives.
- The W&B dashboard provides a clear comparison of the models, showing that es_model is the best candidate for inference.

Challenges

- Ensuring W&B was properly initialized and metrics were logged correctly.
- Interpreting the metrics to choose the best model (es_model) for the Flask app.

Task 4: Training the Models

Objective

Train Logistic Regression models using Batch Gradient Descent, Stochastic Gradient Descent, and Mini-batch Gradient Descent with Early Stopping, and save the models for later use.

Implementation

This task overlaps with Task 2, as the models were trained and saved as part of the Hugging Face upload process. However, here we focus on the training details and hyperparameter choices.

Steps:

1. **Model Training**: The models were trained on the scaled training data (X_train_scaled, y_train) using SGDClassifier. Hyperparameters were tuned to ensure convergence:
   - loss="log_loss": Implements Logistic Regression.
   - penalty="l2": Adds L2 regularization to prevent overfitting.

- alpha=0.0001: Small regularization strength.
- max_iter=1000: Maximum iterations for convergence.
- random_state=42: Ensures reproducibility.
- For es_model, early_stopping=True, validation_fraction=0.1, and n_iter_no_change=5 were used to implement Mini-batch Gradient Descent with early stopping.

2. **Save the Models**: The models were saved as bgd_model.joblib, sgd_model.joblib, and es_model.joblib using joblib.dump, as shown in Task 2.

Outcome

- All three models were successfully trained and saved.
- The es_model benefited from early stopping, converging faster than the other models while maintaining high performance.
- The saved models were used in Task 2 for Hugging Face upload and in Task 5 for the Flask app.

Challenges

- Tuning hyperparameters (e.g., alpha, max_iter) to ensure convergence without overfitting.
- Ensuring the es_model used an appropriate validation fraction and stopping criterion.

Task 5: Web UI Using Flask

Objective

Develop a Flask-based web application to serve inference requests, providing a user interface for users to input the 30 features of the Breast Cancer dataset and receive predictions (Malignant or Benign) along with probabilities.

Implementation

A Flask app was created with a backend (app.py) and a frontend (index.html). The backend downloads the es_model and scaler from Hugging Face, processes user inputs, and returns predictions. The frontend provides a form for users to input the 30 features and displays the prediction results.

Steps:

1. **Project Setup**:
   Created a project directory and set up a virtual environment.

2. **Backend (app.py)**: The Flask app downloads the es_model and scaler from Hugging Face, accepts POST requests with a 30-feature vector, scales the features, makes a prediction, and returns the result as JSON.
3. **Frontend (index.html)**: The frontend is an HTML form with 30 input fields corresponding to the Breast Cancer dataset features. JavaScript handles form submission, sends the data to the /predict endpoint, and displays the prediction and probabilities.
   python app.py
   The UI was accessed at http://127.0.0.1:5000.
4. **Testing**:
   - Tested with the first validation sample: [13.08, 15.71, 85.63, 520.0, 0.1075, 0.127, 0.04568, 0.0311, 0.1967, 0.06811, 0.1852, 0.7477, 1.383, 14.67, 0.004097, 0.01898, 0.01698, 0.00649, 0.01678, 0.002425, 14.5, 20.49, 96.09, 630.5, 0.1312, 0.2776, 0.189, 0.07283, 0.3184, 0.08183].
   - The app correctly predicted "Benign" with a high probability (e.g., 98.00%).

Outcome

- The Flask app successfully loads the es_model and scaler from Hugging Face, processes user inputs, and returns accurate predictions.
- The UI is user-friendly, allowing users to input the 30 features and view the prediction and probabilities.
- Debug logging was added to app.py to troubleshoot issues (e.g., the app hanging on 127.0.0.1:5000).

Challenges

- **Initial Error ("name 'scaler' is not defined")**: This occurred because the scaler was not loaded in app.py. Fixed by adding code to download and load scaler.joblib from Hugging Face.
- **Scaler Removed from Hugging Face**: Initially removed the scaler.joblib file from Hugging Face, causing errors. Resolved by re-uploading the scaler.
- **App Hanging on 127.0.0.1:5000**: The app kept loading without displaying the UI. Debug logging revealed that the app was hanging during the hf_hub_download step due to network issues or slow Hugging Face server response. Suggested fixes included:

- Testing with local files to bypass hf_hub_download.
- Checking for port conflicts (e.g., port 5000 in use).
- Ensuring a stable internet connection.

- **Hugging Face Symbolic Link Warnings**: Encountered warnings about symbolic links when using hf_hub_download. These were resolved by running the app as Administrator or enabling symbolic links on Windows using fsutil behavior set SymlinkEvaluation L2L:1 R2R:1 L2R:1 R2L:1.

Task 6: GitHub Repository

Objective

Create a GitHub repository to store the project code, including the Jupyter notebook, Flask app files, and a README for documentation.

Implementation

A GitHub repository was created to version control the project files and provide a centralized location for the code.

Steps:

1. **Create the Repository**:
   - Created a new repository on GitHub named breast-cancer-classification under the user arsalmairaj2k.
   - URL: https://github.com/arsalmairaj2k/breast-cancer-classification
2. **Project Structure**: The repository includes the following files:

```
breast-cancer-classification/
├── MLFR_CS4090_A3_i211520.ipynb  # Jupyter notebook with all tasks
├── app.py                # Flask app backend
├── templates/
│   └── index.html          # Flask app frontend
├── README.md               # Documentation
└── requirements.txt        # Dependencies
```

3. **Add Files**:
   - The Jupyter notebook (MLFR_CS4090_A3_i211520.ipynb) contains all code for Tasks 1–5.
   - app.py and templates/index.html were added for the Flask app.
   - requirements.txt was created to list dependencies:

4. **README Documentation**: The README.md includes:
    - Project overview and objectives.
    - Instructions to set up and run the Flask app locally.
    - Links to the Hugging Face repository and W&B dashboard.
    - A list of dependencies and installation steps.
    - Example usage of the Flask app with a sample input.

Outcome

- The GitHub repository successfully stores all project files and provides version control.
- The README makes the project accessible to others, with clear instructions for setup and usage.
- The repository ensures reproducibility and collaboration.

Challenges

- Ensuring all necessary files (app.py, index.html, requirements.txt) were included in the repository.
- Writing a comprehensive README that covers setup, usage, and external links (Hugging Face, W&B).

Conclusion

This project successfully implemented a machine learning pipeline for breast cancer classification using the Breast Cancer Wisconsin dataset. Key achievements include:

- Preprocessing the dataset with proper scaling and splitting (Task 1).
- Training and uploading three Logistic Regression models (Batch, Stochastic, Mini-batch with Early Stopping) to Hugging Face (Tasks 2 and 4).
- Evaluating the models and logging metrics to W&B for visualization (Task 3).
- Developing a Flask web UI to serve predictions, with a user-friendly interface for inputting features and viewing results (Task 5).
- Creating a GitHub repository for version control and documentation (Task 6).

The es_model (Mini-batch Gradient Descent with Early Stopping) was selected for inference due to its superior performance, as seen in the W&B dashboard. The Flask app provides an accessible way for users to interact with the model, and the GitHub repository ensures the project is well-documented and reproducible.

Future Work

- Address the Flask app hanging issue by optimizing the hf_hub_download process (e.g., caching files locally).
- Complete Task 7 (deployment) to host the Flask app on a cloud platform for public access.
- Explore additional models (e.g., Random Forest, XGBoost) to potentially improve classification performance.