Introduction to Machine Learning Project

Project Overview

For this project, you will be building and evaluating a series of machine learning models that can predict (classify) for a given piece of software whether it is malware or goodware, identify the best performing model and put it into production as part of a Web application, and use a CI/CD pipeline including automated testing for its gated deployment. You can complete this project either individually or as a group of no more than three people.

Learning Outcomes

When completed successfully, this project will enable you to:

- Correctly preprocess and analyze structured data.
- Train, tune, and evaluate ML models using cross-validation and test sets.
- Package a trained ML model for inference in production.
- Build and deploy a Web application serving ML predictions.
- Implement a CI/CD pipeline with automated testing.
- Demonstrate and communicate ML results effectively.

Project Description

Malware refers to malicious code, such as viruses, keyloggers or backdoor programs and goodware refers to normal, benign software that provides some useful, user-desired function. Your task is known as **machine learning-based static malware detection**,

referred to as static as this is carried out based on features of the (static) executable (.exe) file, which allows prediction of potential harmful software operation prior to running it—which is desirable!

Alternatively, if you wish to build a predictive model based on a different dataset and corresponding predictive problem, you can submit a proposal for an alternative predictive problem and corresponding dataset to be approved by a Quantic faculty member at projects+msse@quantic.edu. **If approved**, you can instead work with your own selected dataset and predictive problem for this project.

You will make use of Python and its machine learning-related packages such as scikit-learn, PyTorch, and other Python libraries and associated data manipulation packages for this project. While you can fully hand code this project if you wish, you are highly encouraged to utilize leading AI code generation models/AI IDEs to assist in rapidly producing your solution, being sure to describe in broad terms how you made use of them. Here are some examples of very useful AI tools you may wish to consider. You will be graded on the quality and functionality of the application and how well it meets the requirements—no given proportion of the code is required to be hand coded.

The dataset can be found here. It is in .csv format and contains approximately 50,000 instances. It has 27 input attributes, along with a target attribute (Column L, called "Label") with a value of 0 indicating goodware and 1 indicating malware. These executable files are of type Portable Executable (PE), the very common executable format used for .exe files on the Windows operating system.

Step-by-Step Instructions

1. Dataset & Problem Definition
    - Confirm dataset and target variable. Define success metrics (primary: AUC, secondary: accuracy).
    - Ensure 20% test set is held out before data pre-processing or feature engineering to prevent leakage.
2. Environment & Reproducibility
    - Use a virtual environment (e.g., venv, conda etc.).
    - Pin dependencies in requirements.txt or environment.yml (conda).
    - Provide scripts (e.g., train.py, eval.py) to reproduce results.
    - Set fixed random seeds for splits and cross-validation.
3. Data Understanding & Preparation
    - Conduct exploratory data analysis (EDA): inspect distributions, class balance, missing data.

- Document any issues and how they are handled.
- Apply preprocessing only on training data (fit on train folds, transform validation/test).

4. Train/Validation/Test Protocol
    - Split dataset into 80% training and 20% hold-out test set (stratified according to class proportions).
    - Within the training set, perform stratified 10-fold cross-validation (CV) for model selection and hyperparameter tuning.
    - Keep the test set untouched until the final evaluation.
5. Preprocessing & Feature Engineering
    - Apply scaling, encoding, and imputation as needed, ensuring transformations are fit on training only.
    - Preprocessing steps (e.g., scaling, imputation) must be fit only on training folds during cross-validation, and then applied to the corresponding validation/test folds.
    - Explore feature selection or dimensionality reduction and justify choices.
6. Model Training & Evaluation
    - Required baseline models: Logistic Regression, Decision Tree, Random Forest, and one PyTorch MLP.
    - Additional models: evaluate at least 3 further high-performing models, spanning at least two different algorithm families (e.g., XGBoost, LightGBM, CatBoost).
    - Record cross-validation AUC and accuracy (mean ± std dev) for all models.
    - After comparing results, the top-performing model from CV will be selected for final evaluation on the hold-out test set.
    - Record the results of the final production model on this test set.
7. Web Application Development
    - Package the chosen production model and integrate it into a Web application built with Flask.
    - The Web app must include:
        - A UI form for manual feature entry (can use pre-filled demo row from the malware dataset to demo - to avoid the need to enter many feature values manually) that then displays the prediction (malware or goodware)
        - An upload option for a file containing numerous instances, for which the predictions are displayed for each.
        - If the uploaded file contains class labels, the app runs an evaluation and displays AUC, accuracy, and confusion matrix. This can be demoed with the 20% hold-out test file, to demonstrate performance metrics.

8. Web Application Deployment
    - Deploy the application to a host with a free-tier (e.g., Render, Railway, etc.).
    - Ensure the app is publicly accessible via a shareable URL.
9. CI/CD Pipeline
    - Implement a basic pipeline (e.g., GitHub Actions).
    - At minimum, your CI/CD pipeline must run automated tests before deploying new code. This can be triggered either (a) on pull requests before merging, or (b) on pushes to the main branch.
    - Deployment must only occur if tests pass. If you push directly to main and tests fail, the code will still be on main (just not deployed).
10. Automated Testing
    - Unit tests: preprocessing and model wrapper functions.
    - Integration tests: check /predict API endpoint with a sample payload.
    - Post-deploy smoke test: GET /health endpoint to confirm deployment.

Submission Guidelines

Your submission should be a single PDF document containing two links: 1) a link to your recorded demo presentation and 2) a link to your GitHub repository containing the following:

- You **must add** “quantic-grader” as a collaborator to your private repo. Add this collaborator using the repo’s Settings > Collaborators > Add people.
- Must contain all source code and CI/CD configuration.
- Must include a deployed.md file containing a URL link to the live deployed Web application.
- Must include an evaluation-and-design.md file providing CV results for all models and final hold-out test set evaluation and explaining any design decisions, data pre-processing or feature engineering done.
- A brief ai-tooling.md document mentioning AI tools used for code generation and software engineering and how they were used (i.e., what worked well, what didn’t).
- Your recorded screen-share video should demo the use of the Web application providing access to the production model at its public URL for inference, also explaining automatic testing done during the CI/CD pipeline and showing the operation of the CI/CD pipeline.
    - All group members must speak and be present on camera.
    - The demonstration/presentation should be between 5 and 10 minutes long.
    - You do not need to show your ID in this demo.

To submit your project, please click on the "Submit Project" button on your dashboard and follow the steps provided. If you are submitting your project as a group, **please**

ensure only ONE member submits on behalf of the group. Please reach out to msse+projects@quantic.edu if you have any questions. Project grading typically takes about 3-4 weeks to complete after the submission due date. There is no score penalty for projects submitted after the due date, however grading may be delayed.

Plagiarism Policy

Here at Quantic, we believe that learning is best accomplished by "doing"—this ethos underpinned the design of our active learning platform, and it likewise informs our approach to the completion of projects and presentations for our degree programs. We expect that all of our graduates will be able to deploy the concepts and skills they've learned over the course of their degree, whether in the workplace or in pursuit of personal goals, and so it is in our students' best interest that these assignments be completed solely through their own efforts with academic integrity.

Quantic takes academic integrity very seriously—we define plagiarism as: "Knowingly representing the work of others as one's own, engaging in any acts of plagiarism, or referencing the works of others without appropriate citation." This includes both misusing or not using proper citations for the works referenced, and submitting someone else's work as your own. Quantic monitors all submissions for instances of plagiarism and all plagiarism, even unintentional, is considered a conduct violation. If you're still not sure about what constitutes plagiarism, check out this two-minute presentation by our librarian, Kristina. It is important to be conscientious when citing your sources. When in doubt, **cite**! Kristina outlines the basics of best citation practices in this one-minute video. You can also find more about our plagiarism policy here.

Project Rubric

Scores 2 and above are considered passing. Students who receive a 1 or 0 will not get credit for the assignment and must revise and resubmit to receive a passing grade.

Score	Description
5	• Addresses ALL of the project requirements, but not limited to: ○ Implements baseline models plus ≥3 additional models with complete CV and test evaluation. ○ Report includes CV table (AUC/accuracy ± sd), and final hold-out test set performance metrics. ○ Web application is fully functional: UI including file upload work. Metrics displayed if the test set is uploaded. ○ Successful public deployment to Render, Railway, Fly.io etc. (or other choice). ○ CI/CD pipeline fully functional with tests on PR or push to main with auto-deploy if tests pass. ○ Substantial automated unit + integration + smoke tests implemented. ○ Clear, effective demo presentation that shows both the UI functionality and CI/CD pipeline operation.
4	• Addresses MOST of the project requirements, but not limited to: ○ All baseline models and ≥3 additional models evaluated; minor omissions. ○ Mostly reproducible environment (small gaps) ○ Report includes CV table and test set metrics with minor clarity issues. ○ Web application functional, but some UI/file upload features incomplete. ○ Public deployment works with minor issues. ○ CI/CD present; some limitations in tests or deploy automation. ○ Demo presentation provided, covers most aspects.

3	• Addresses SOME of the project requirements, but not limited to:
	  ○ Four baseline models trained; fewer than 3 additional models OR shallow evaluation.
	  ○ Partial reproducibility (some pins missing).
	  ○ Report includes CV results but lacks depth or completeness.
	  ○ Web application partially working; test file upload feature missing or incomplete.
	  ○ Deployment available but with some issues.
	  ○ CI/CD incomplete (tests or deploy missing).
	  ○ Demo unclear or missing demo of functionality.

2	• Addresses FEW of the project requirements, but not limited to:
	  ○ Fewer than four baseline models; minimal additional models.
	  ○ Poor reproducibility practices.
	  ○ Sparse report, missing results tables.
	  ○ Minimal or non-functional Web application
	  ○ Deployment has significant issues.
	  ○ CI/CD lacking or minimal.
	  ○ Weak or limited demo.

1	• Addresses the project but MOST of the project requirements are missing, but not limited to:
	  ○ Baselines not implemented correctly; few results.
	  ○ No reproducibility.
	  ○ Report largely absent.
	  ○ No working Web app.
	  ○ No deployment.
	  ○ No CI/CD or tests.
	  ○ No demo.

0	• The student either did not complete the assignment, plagiarized all or part of the assignment, or completely failed to address the project requirements.