Laboratorio di Informatica Applicata

Final Project

Prof. Tiberio Uricchio

1 Project Overview

This document outlines the final project requirements for the Laboratorio di Informatica Applicata course. The project is designed to integrate and apply the key concepts covered throughout the course, including Python programming, data analysis, machine learning, version control, and containerization.

1.1 General Objectives

1. Develop a complete data analysis and machine learning system
2. Implement version control using Git and GitHub
3. Create a deployable solution using Docker containers
4. Apply best practices in software development
5. Demonstrate proficiency in Python and its key libraries for data science

2 Project Requirements

2.1 Core Components

The project must include the following components:

2.1.1 Data Analysis Component

- Load and preprocess a dataset (chosen from suggested options or approved by the instructor)
- Perform exploratory data analysis using Pandas and NumPy (or other tools if needed)
- Create informative visualizations using Matplotlib and Seaborn (or other tools if needed)
- Document insights discovered through the analysis

2.1.2 Machine Learning Component

- Implement a machine learning model using scikit-learn (or more than one if needed)
- Train, tune, and evaluate the models using appropriate metrics
- Compare model performance and explain the results
- Optional advanced component: implement a neural network using PyTorch

2.1.3 Web Interface Component

- Create a simple web interface and a REST/JSON API using Flask
- Implement functionality to input data for prediction
- Display prediction results and relevant visualizations
- Include documentation explaining the models and analysis
- (Optional) Add proper unit tests

2.1.4 Version Control

- Maintain a well-structured Git repository on GitHub
- Use meaningful commit messages following best practices
- Include a comprehensive README.md file
- Create an appropriate .gitignore file

2.1.5 Containerization

- Create a Dockerfile for the application
- Create a docker-compose.yml file if multiple services are required
- Ensure the application runs consistently in the containerized environment
- Document container usage and deployment instructions

3 Suggested Applications

Students may choose from the following applications or propose an alternative (subject to approval by the instructor). Each project should aim to incorporate data analysis, machine learning, a Flask web interface/API, Git version control, and Docker containerization.

3.1 Housing Price Prediction

- **Description:** Develop a model to predict housing prices based on various features such as location, size, number of rooms, amenities, and other relevant attributes.
- **Dataset:** Kaggle Housing Prices Competition (primary), or create a custom dataset.
- **Task:** Regression
- **Difficulty:** Easy
- **Additional ideas:**
  - Scrape data from local real estate websites (e.g., `immobiliare.it`, `idealista.it`) to build and analyze a custom, localized dataset.
  - Implement feature engineering to create new insightful variables (e.g., price per square meter, age of house, proximity to amenities).
  - Compare the performance of different regression models (Linear Regression, Random Forest, Gradient Boosting, SVR).

3.2 Customer Churn Analysis

- **Description:** Predict which customers are likely to stop using a service (churn) based on their behavior patterns, demographics, and service usage history.
- **Dataset:** Telco Customer Churn
- **Task:** Binary Classification
- **Difficulty:** Easy
- **Additional ideas:**
  - Use a Large Language Model (LLM) to generate synthetic customer profiles with varying churn likelihoods to test model robustness or augment the training data.
  - Compare the performance of classical ML classification algorithms against an LLM-based approach (e.g., prompting an LLM to classify churn likelihood based on a customer profile).
  - Identify key features driving churn and provide actionable insights for retention strategies (e.g., using SHAP or LIME for model interpretability).

3.3 Stock Price Movement Prediction

- **Description:** Analyze historical stock data to predict future price movements or trends for specific stocks or market indices. This could involve predicting the price at a future point or a buy/sell/hold signal.
- **Dataset:** Build your own dataset using financial data APIs like Yahoo Finance (accessible via the `yfinance` library) or other financial data providers (e.g., Alpha Vantage, IEX Cloud).
- **Task:** Regression or Classification (Time Series Forecasting)

- **Difficulty:** Medium
- **Additional ideas:**
  - Incorporate sentiment analysis from financial news headlines (e.g., using NewsAPI) or social media (e.g., Twitter/X, StockTwits) as additional features.
  - Implement and evaluate a simple backtesting framework to assess the profitability of trading strategies based on your model's predictions.
  - Explore time-series specific models like ARIMA, Prophet, or LSTMs (optional advanced component).

3.4 Medical Diagnosis Aid

- **Description:** Develop a system to predict the likelihood of a specific disease (e.g., diabetes, heart disease) based on patient medical records and diagnostic measurements.
- **Dataset:** PIMA Indians Diabetes Database, Heart Disease UCI
- **Task:** Binary/Multi-class Classification
- **Difficulty:** Easy/Medium
- **Additional ideas:**
  - Investigate the use of LLMs for diagnostic assistance by providing patient data in a structured prompt and evaluating the LLM's diagnostic suggestion. Compare its performance and reasoning to your trained ML model.
  - Focus on model interpretability (e.g., using SHAP or LIME) to understand which features are most influential in predictions.
  - Address potential biases in the dataset and discuss their implications on fairness and equity in healthcare.

3.5 Image Classification with Transfer Learning

- **Description:** Build an image classifier for a specific domain (e.g., types of animals, fashion items, plant species, architectural styles) by leveraging pre-trained deep learning models.
- **Dataset:** CIFAR-10, Fashion MNIST, Caltech-101, or a custom dataset (e.g., images scraped from the web).
- **Task:** Multi-class Classification
- **Difficulty:** Medium
- **Additional ideas:**
  - Implement various data augmentation techniques (rotation, zoom, flips, color jitter, mixup, cutmix) to improve model robustness and performance.
  - Compare the performance of different pre-trained architectures (e.g., ResNet, VGG, MobileNet, EfficientNet, ViT) when fine-tuned on your chosen dataset.

  - Visualize feature maps or attention maps from different layers of the CNN/Transformer to understand what the model is learning.
  - (Advanced) Attempt to train a smaller CNN from scratch and compare its performance to transfer learning approaches.

3.6 Smart Document Organizer & Analyzer

- **Description:** Build a system to automatically analyze, categorize, and suggest new names or metadata (like tags, summaries) for PDF or text documents.
- **Dataset:** Create a diverse test set of various document types (e.g., academic papers, invoices, personal letters, technical reports, news articles).
- **Task:** Text Classification, Information Extraction, NLP, LLM Application
- **Difficulty:** Medium
- **Additional ideas:**
  - Compare the effectiveness of different LLMs (e.g., via APIs like OpenAI, Claude, or local models using Ollama) for content understanding, summarization, and metadata suggestion.
  - Implement a feature to extract key entities (people, organizations, dates, technical terms) from documents using spaCy or an LLM.
  - Develop a user interface that allows batch processing of documents and editing of suggested metadata.

3.7 UNIPI Academic Information Assistant

- **Description:** Create an AI-powered assistant that can answer queries about UNIPI academic information by retrieving data from the university website. This could utilize a Model Context Protocol (MCP) server architecture to augment an LLM's capabilities or a RAG (Retrieval Augmented Generation) pipeline.
- **Task:** Information Retrieval, LLM Augmentation, Web Scraping, Question Answering
- **Difficulty:** Medium/Hard
- **Additional ideas:**
  - Implement a system to retrieve and present information like course schedules, professor office hours, exam dates, or program descriptions.
  - Build a conversational interface (e.g., using Flask for the backend and a simple HTML/JS frontend) that can maintain context across multiple student queries.
  - (Advanced) Explore creating a vector database from UNIPI website content for efficient RAG.
- **Resources:** MCP Fast agents, Claude API, Beautiful Soup or Scrapy (for web scraping), LangChain or LlamaIndex (for RAG pipelines), Example MCP servers.

3.8 3D Artwork Viewer from 2D Images

- **Description:** Using an image segmentation model, segment artworks (paintings, photographs) into distinct regions. Allow users to define or estimate depth for these segments and render a pseudo-3D (2.5D) representation on a web page.
- **Dataset:** Public domain artwork images (e.g., from museum websites like The Met, Rijksmuseum, or Wikimedia Commons).
- **Task:** Image Segmentation, 3D Rendering (simplified), Web Visualization
- **Difficulty:** Medium/Hard
- **Additional ideas:**
  - Compare different segmentation models (e.g., Segment Anything Model (SAM), or MLLM-based segmentation).
  - Implement interactive controls for users to adjust depth parameters for different segments or to "paint" depth maps.
  - (Advanced) Explore basic 3D rendering techniques using WebGL (e.g., via `Three.js`) or look into creating anaglyph 3D images for viewing with red-cyan glasses.
- **Resources:** `OpenCV`, `Pillow`, `Three.js` (JavaScript library for 3D graphics).

3.9 Multimodal LLM for Image Segmentation

- **Description:** Explore the emergent capability of Multimodal Large Language Models (MLLMs) to perform image segmentation by providing an image and a text prompt describing the object(s) to segment. Compare this approach with a traditional, specialized segmentation model.
- **Dataset:** A diverse set of images for segmentation tasks (e.g., COCO dataset samples, PASCAL VOC, or custom images). Ground truth masks for a small test set would be beneficial for quantitative comparison.
- **Task:** Image Segmentation, MLLM Application, Comparative Analysis
- **Difficulty:** Medium
- **Additional ideas:**
  - Create a benchmark to compare segmentation quality (e.g., Intersection over Union - IoU scores) and speed between an MLLM API and a dedicated model like SAM.
  - Analyze failure cases and scenarios where MLLMs excel or struggle compared to specialized models (e.g., complex scenes, fine-grained details, ambiguous prompts).
  - Develop a web interface that allows users to upload an image, provide a text prompt, and see segmentation results from both methods side-by-side.
- **Resources:** Simon Willison's blog post on Gemini for segmentation, Moondream AI, APIs from OpenAI (GPT-4V), Google (Gemini Pro Vision).

3.10 Optical Music Recognition (OMR) System

- **Description:** Implement a system to recognize musical notes and symbols from images of sheet music. The core task is to identify notes, rests, clefs, key signatures, time signatures, etc.
- **Dataset:** Public OMR datasets (e.g., CVC-MUSCIMA, MUSCIMA++, DeepScores).
- **Task:** Object Detection, Image Processing, (potentially Sequence Modeling)
- **Difficulty:** Medium/Hard
- **Additional ideas:**
  - Convert recognized notation into a machine-readable format like MusicXML or MIDI, enabling playback or further digital manipulation.
  - Implement a feature to correct common recognition errors through a user interface.
  - Focus on handling handwritten sheet music or scores with complex layouts, which present additional challenges.
- **Resources:** Curated list of OMR Datasets and code, `OpenCV`, `TensorFlow Object Detection API` or `YOLO`.

3.11 Document Knowledge Graph Extractor

- **Description:** Automatically extract named entities (people, organizations, locations, concepts, dates) and the relationships between them from a given text document to construct and visualize a knowledge graph.
- **Dataset:** Text documents of your choice (e.g., news articles, historical texts, technical manuals, Wikipedia articles).
- **Task:** Named Entity Recognition (NER), Relationship Extraction, Knowledge Graph Construction, NLP, Graph Visualization
- **Difficulty:** Medium
- **Additional ideas:**
  - Compare the performance of traditional NLP NER/RE tools (e.g., `spaCy`, `NLTK` with custom rules) versus LLM-based entity and relation extraction (e.g., prompting an LLM to output structured data).
  - Implement interactive graph visualizations using libraries like `pyvis`, `Cytoscape.js`, or `Plotly Dash` within your Flask app.
  - Allow users to query the knowledge graph (e.g., "What organizations are mentioned in relation to [Person X]?").
- **Resources:** `spaCy`, `NLTK`, `NetworkX` (for graph manipulation), `pyvis`.

3.12 AI-Powered GeoGuesser

- **Description:** Develop a system that attempts to identify the geographic location (country, city, region, or even specific landmark) depicted in an uploaded image, mimicking the popular GeoGuessr game.
- **Dataset:** No specific training dataset required if primarily using pre-trained Vision Language Models (VLMs) or MLLMs. For evaluation, a set of diverse images with known geolocations will be needed.
- **Task:** Image Analysis, LLM Application (Visual Question Answering / Image Captioning with geographic focus), Geolocation
- **Difficulty:** Easy/Medium
- **Additional ideas:**
  - Experiment with different prompting strategies for VLMs/MLLMs to elicit better location clues (e.g., "Describe this image with a focus on architectural styles, vegetation, language on signs, and any other geographic indicators," or "What country and city might this be in and why?").
  - Integrate a mapping API (e.g., `Leaflet.js` with OpenStreetMap, or Google Maps API) to display the predicted location and potentially the actual location for comparison.
  - Create a scoring system based on the distance between the predicted and actual location.
- **Resources:** GeoGuessr (for inspiration), APIs from OpenAI (GPT-4V), Google (Gemini Pro Vision), `geopy` library for distance calculations.

3.13 Fine-Grained Fruit/Plant Recognition

- **Description:** Implement an object detection and classification system specifically for identifying different varieties of a particular fruit (e.g., apples, grapes) or plant species, where subtle visual differences are key.
- **Dataset:** MinneApple, CitDet, PlantVillage Dataset, or a custom dataset of high-resolution images.
- **Task:** Object Detection, Fine-Grained Visual Classification (FGVC)
- **Difficulty:** Medium
- **Additional ideas:**
  - Focus on data augmentation techniques crucial for fine-grained tasks (e.g., careful cropping, color space manipulation, advanced augmentation libraries).
  - Evaluate the model's ability to distinguish between very similar-looking varieties and analyze common misclassifications using confusion matrices and visualization of misclassified samples.
  - Implement a confidence score for the detected fruit/plant type and variety.

- **Resources:** MinneApple Paper, CitDet Project Page.

3.14 Interactive Hand Gesture Control System

- **Description:** Develop a system that recognizes specific hand gestures from a webcam feed and maps them to predefined actions or commands (e.g., control presentation slides, play/pause media, volume control, simple game controls).
- **Dataset:** Pre-trained hand pose estimation models (like from MediaPipe) are often used as a base. For gesture classification, a small custom dataset of gesture sequences (videos or series of landmarks) might need to be created by the student.
- **Task:** Hand Pose Estimation, Gesture Recognition, Real-time Application
- **Difficulty:** Medium/Hard
- **Additional ideas:**
  - Use a library like `MediaPipe Hands` for robust hand tracking and landmark detection in real-time.
  - Define a set of custom static or dynamic gestures and train a simple classifier (e.g., k-NN, SVM, or a small neural network like an LSTM for dynamic gestures) on features derived from hand landmarks.
  - Integrate the gesture recognition with your Flask application to control elements on a webpage or interact with other applications via simulated key presses.
- **Resources:** MediaPipe Hands, Example of hand recognition software, `OpenCV`.

3.15 Audio Event Detection and Classification

- **Description:** Create a system that identifies and classifies various sound events from audio recordings or a live microphone feed (e.g., glass breaking, dog barking, siren, car horn, speech, music).
- **Dataset:** AudioSet (large, for pre-trained models or subsets for fine-tuning), ESC-50 (smaller, for custom training/fine-tuning), Urbansound8K.
- **Task:** Audio Classification, Sound Event Detection (SED)
- **Difficulty:** Medium
- **Additional ideas:**
  - Use pre-trained audio models (e.g., YAMNet, PANNs, AST) for feature extraction or direct classification.
  - Implement real-time processing and visualization of detected sound events and their confidence scores over time (e.g., a scrolling timeline or a live dashboard in the Flask app).
  - (Advanced) Allow users to upload audio files for analysis or process live audio from the microphone (requires browser permissions).

- **Resources:** `Librosa` (for audio processing and feature extraction like MFCCs), `TensorFlow Hub` for Audio Models, `Hugging Face Transformers` for audio models.

3.16 Extractive & Abstractive Text Summarization Service

- **Description:** Build a web service that takes a long piece of text (e.g., news article, blog post, research paper abstract) as input and generates a concise summary. Implement and compare both extractive and abstractive summarization techniques.
- **Dataset:** For extractive, no specific training data needed for algorithms like TextRank. For abstractive, use pre-trained models or fine-tune on datasets like CNN/DailyMail or XSum.
- **Task:** NLP, Text Summarization
- **Difficulty:** Medium
- **Additional ideas:**
  - Implement an extractive method (e.g., TF-IDF based, TextRank/LexRank) and an abstractive method (e.g., using a pre-trained transformer model like BART, T5, PEGASUS, or an LLM API).
  - Allow users to specify desired summary length (e.g., number of sentences or percentage of original text).
  - Evaluate summaries using ROUGE scores against reference summaries if available for a test set, or perform qualitative evaluation.
- **Resources:** `NLTK`, `spaCy` (for extractive methods), `Hugging Face Transformers` library for summarization models.

3.17 AI-Powered Code Review Assistant (Basic)

- **Description:** Create a tool that uses an LLM to analyze Python code snippets, identify potential basic bugs (e.g., unused variables, simple logical errors), suggest style improvements based on PEP 8, or generate docstrings.
- **Dataset:** No specific training dataset needed; relies on LLM capabilities. Test with various Python code snippets.
- **Task:** LLM Application, Code Analysis, NLP
- **Difficulty:** Medium
- **Additional ideas:**
  - Integrate with a Flask web interface where users can paste code and get feedback.
  - Experiment with different LLMs (OpenAI, Claude, local models like CodeLlama via Ollama) and prompting techniques for optimal results.
  - Focus on a specific type of code review feedback, e.g., security vulnerabilities (very basic level) or performance tips.

- **Resources:** LLM APIs, `Ollama` for local models, `pylint` or `flake8` (to compare LLM suggestions with traditional linters).

3.18 Real-time Anomaly Detection in Sensor Data Streams

- **Description:** Develop a system to detect anomalies in real-time or near real-time streaming data from sensors (e.g., temperature, pressure, vibration from IoT devices, or simulated data).
- **Dataset:** Generate synthetic time-series data with injected anomalies, or use public datasets like Numenta Anomaly Benchmark (NAB).
- **Task:** Anomaly Detection, Time Series Analysis
- **Difficulty:** Medium/Hard
- **Additional ideas:**
  - Implement and compare statistical methods (e.g., Z-score, Moving Averages) with unsupervised machine learning approaches.
  - Develop a Flask application that visualizes the data stream and highlights detected anomalies in real-time (or simulated real-time).
  - Explore methods for setting dynamic thresholds for anomaly detection.
- **Resources:** `scikit-learn` (for Isolation Forest, One-Class SVM), `statsmodels` (for time series analysis), `PyOD` (Python Outlier Detection library).

3.19 Personalized News Feed Aggregator & Classifier

- **Description:** Build a system that fetches news articles from various sources based on user-defined keywords or topics, classifies them into predefined categories (e.g., politics, technology, sports), and presents a personalized feed.
- **Dataset:** Use news APIs (e.g., NewsAPI.org, GNews API) to fetch articles. For classification, you might use a pre-trained text classifier or fine-tune one on a dataset like AG News or 20 Newsgroups.
- **Task:** Web Scraping/API Integration, Text Classification, NLP, Personalization
- **Difficulty:** Medium/Hard
- **Additional ideas:**
  - Implement a feature for users to "like" or "dislike" articles to refine their preferences over time (simple collaborative filtering or content-based filtering for personalization).
  - Use an LLM to summarize articles or generate topical tags.
  - Develop a Flask web interface that displays the aggregated and classified news feed, filterable by category or source.
- **Resources:** `requests`, `Beautiful Soup` (if scraping), `feedparser` (for RSS feeds), `scikit-learn` (for text classification), `Hugging Face Transformers`.

3.20 AI Recipe Generator from Ingredients

- **Description:** Create an application that takes a list of available ingredients from a user and suggests or generates recipes they can make.
- **Dataset:** Scrape recipe websites (e.g., Allrecipes, Epicurious) or use existing recipe datasets like RecipeNLG or Recipe1M+.
- **Task:** Information Retrieval, NLP, LLM Application (Generative Task)
- **Difficulty:** Medium/Hard
- **Additional ideas:**
  - Use an LLM to generate novel recipe ideas or adapt existing recipes based on the given ingredients and user preferences (e.g., cuisine type, dietary restrictions).
  - Implement a feature to rank suggested recipes based on ingredient match, user ratings (if available), or estimated cooking time/difficulty.
  - Develop a Flask interface for users to input ingredients and view generated/suggested recipes with instructions.
- **Resources:** Web scraping libraries, LLM APIs, recipe dataset sources.

3.21 Plant Disease Detection from Leaf Images

- **Description:** Develop an image classification system to identify common diseases in plants based on images of their leaves.
- **Dataset:** PlantVillage Dataset, or other publicly available plant disease image datasets.
- **Task:** Image Classification (Multi-class)
- **Difficulty:** Medium
- **Additional ideas:**
  - Use transfer learning with pre-trained CNN architectures (e.g., MobileNetV2, EfficientNet for potentially mobile-friendly deployment).
  - Implement data augmentation techniques specifically suited for leaf images (e.g., variations in lighting, rotation, scale).
  - Develop a simple Flask web application where users can upload an image of a plant leaf and get a prediction of the disease, along with confidence scores.
- **Resources:** `TensorFlow/Keras` or `PyTorch` for model building.

3.22 Sports Game Outcome Predictor

- **Description:** Build a model to predict the outcome (e.g., win/loss, score difference) of sports games (e.g., football, basketball, soccer) based on historical game data, team statistics, and player performance.

- **Dataset:** Publicly available sports statistics websites or APIs (e.g., ESPN data, Kaggle sports datasets, specialized sports stats providers).
- **Task:** Classification or Regression
- **Difficulty:** Medium
- **Additional ideas:**
  - Perform extensive feature engineering to create relevant predictors (e.g., team form, head-to-head statistics, player availability, home/away advantage).
  - Compare different machine learning models (e.g., Logistic Regression, Random Forest, Gradient Boosting, Neural Networks) for prediction accuracy.
  - Develop a Flask interface to input upcoming game details and display the model's prediction.
- **Resources:** `Pandas` for data manipulation, `scikit-learn` for modeling.

4 General Evaluation Criteria

The project part will be evaluated based on the following general criteria:

4.1 Code Quality (15%)

- Clean, well-organized, and documented code
- Proper use of Python language features and idioms
- Adherence to Python style guidelines
- Effective error handling and edge cases
- Modular and reusable code structure

4.2 Data Analysis (20%)

- Depth and thoroughness of exploratory data analysis
- Quality and relevance of visualizations
- Insights extracted from the data
- Appropriate data preprocessing and feature engineering
- Handling of missing data and outliers

4.3 Machine Learning (20%)

- Appropriate model selection for the problem
- Proper model evaluation and hyperparameter tuning
- Performance of the predictive models

- Understanding of model strengths and limitations
- Comparison of different approaches

4.4 Web Interface (20%)

- Functionality and user experience
- Integration with the machine learning models
- Documentation and clarity
- Design and user interface considerations
- Error handling and edge cases

4.5 Version Control (10%)

- Quality of Git usage (commits, branches, etc.)
- Completeness of documentation
- Repository organization
- Use of issues and project management tools (optional)

4.6 Containerization (15%)

- Proper Docker configuration
- Ease of deployment
- Container performance and reliability
- Multi-service orchestration (if applicable)

5 Deliverables

Students must submit:

1. **GitHub Repository URL** containing all code and documentation
2. **Project Report** (in PDF format) explaining:
    - Problem statement and objectives
    - Data analysis approach and findings
    - Machine learning methodology and results
    - Technical implementation details
    - Challenges encountered and solutions
    - Eventual Future improvements
3. **Docker Container** instructions for running the application
4. **Presentation** short, max 5 minute long, slides to be presented at the oral

6 Help and code review

If you are stuck or need help, you can always mail the teacher or book an appointment. The teacher is also available to perform code reviews.