Bioinformatics Research Project Log

Chandrashekar CR
Supervisor: Dr. Eran Elhaik
Lund University

April 10, 2025

Contents

March 31, 2025

Tasks for the Day

- Understood the mGPS algorithm from the R code and implemented the preprocessing steps.
- Set up the working environment and installed all required libraries.
- Began tracking the project.

Notes and Observations

- Utilizing the `ai_env` environment from previous projects and accessing resources on the bioinformatics server.
- Initial challenges in understanding the R code are anticipated to decrease with further engagement.

April 1, 2025

Tasks for the Day

- Understood and implemented Recursive Feature Elimination followed by the XGBoost machine learning algorithm with the correct hierarchical steps.
- Initialized Git for version control and pushed code to GitHub.
- Gained a deeper understanding of cross-validation principles.

Notes and Observations

- Achieved a general understanding of the workflow and added comments to the MetaSUB preprocessing script in R.
- Acquired knowledge regarding the importance of cross-validation, although implementation is pending.

April 2, 2025

Tasks for the Day

- Acquired information from Vignesh regarding access to the LUNARC server.
- Determined that reimplementing the exact XGBoost model is unnecessary; the focus is on understanding the input data preprocessing.

Notes and Observations

- Git repository initialized for the project, tracking all files except data and research papers.
- The `metasub_global_git.csv` file contains the Geographically Informative Taxa (GITs) required for the XGBoost model.
- The primary objective is to comprehend the preprocessing of input data for XGBoost.
- Key questions identified: What is the shape of the input data? What are the prediction targets?

April 3, 2025

Tasks for the Day

- Implemented basic neural network architectures in PyTorch. Hyperparameter tuning indicates that 200 GITs are sufficient for accurate predictions, despite the dataset containing n data points.
- Integrated Ray parallel processing to optimize hyperparameters, aiming to reduce the estimated 4-5 hour search time.
- Preprocessed data into numerical format by converting categorical variables (continents and cities) using one-hot encoding.
- Deferred the implementation of stratified K-fold cross-validation for later.

Notes and Observations

- Successfully logged into the LUNARC server's login node, but GPU access and utilization for neural network training require further investigation.
- Authentication and login to LUNARC are complete; however, assistance is needed to understand:
  - The location of allocated storage.
  - The process of submitting jobs using SBATCH.
  - The fundamentals of working on High-Performance Computing (HPC).
- Following Recursive Feature Elimination (RFE), the final dataset has a shape of 4070×204, with 200 features and 4 target variables.
- Each data point comprises 200 features (GITs) representing the relative sequence abundance (RSA) of microorganisms. The 4 target variables are continent, city, latitude, and longitude.
- The dataset includes samples from 40 unique cities across 7 continents.
- Categorical variables (continent and city) were encoded using `sklearn`'s `LabelEncoder`, while latitude and longitude were standardized using `StandardScaler`.
- Initial consideration of stratified cross-validation was temporarily replaced with `train_test_split` for initial model development. Stratified cross-validation will be revisited for enhanced model performance.

Neural Network Architecture

- The initial model is a simple feedforward neural network, inspired by the hierarchical structure of the previous XGBoost study.
- The first version includes an input layer (200 nodes), two hidden layers (400 nodes each), a smaller hidden layer (2 nodes), and an output layer (7 nodes for the 7 continents).
- The plan is to initially train this model to predict the continent. Subsequently, the predicted continent probabilities will be concatenated with the original 200 features as input for a second neural network (similar architecture) to predict the city.
- The optimal handling of latitude and longitude values within the neural network remains an open question.
- Long training times due to CPU-based computation on the bioinformatics server are a significant challenge. GPU access on the LUNARC cluster is required.

April 4, 2025

Tasks for the Day

- Finalized the presentation for the weekly lab meeting.
- Focused on obtaining GPU access and understanding HPC architecture.
- Initiated preprocessing for marine and soil datasets, aiming for modular script design applicable to all datasets.

Notes and Observations

Explored various neural network implementations and gained initial understanding of HPC principles.

April 7, 2025

Tasks for the Day

- Developed a functional neural network to predict all target variables (continent, city, latitude, and longitude).
- Completed preprocessing steps for marine and soil datasets.
- Began modularizing code for efficient execution on the HPC.

Notes and Observations

- Developed a working script for MetaSUB data processing, with pending error handling and file format validation.
- Created a working script to extract relevant features using Recursive Feature Elimination with a Random Forest base model.
- Initiated work on the HPC, understanding basic operations and starting to modularize scripts for GPU compute nodes.

April 8, 2025

Notes and Observations

- Started building multiple neural network models.
- Began learning batch scripting using SLURM.

April 10, 2025

Notes and Observations

- Exploring alternative scaling methods for latitude and longitude, including conversion to radians and trigonometric transformation into a two-dimensional space.

April 11, 2025

Tasks for the day

- These are some of my thoughts. The currrent approach is always making a new neural network model from scratch for each dataset. A final model should be made that utilizes all the iterations done and must work for all type of datasets regardless of the layers of predictions.
- For example, the MetaSUB dataset contains informaiton on the continent, city, latitude and longitude. Whereas the marine dataset contain information only the sea, latitude and longitude. There should be a way that can handle these cases instead of definfing a newtwork from scratch.
- Finish the logic for the latitude and longitude neural network model.
- Compare the three models with metasub dataset on accuracy, confusion matrix, plot on world map.

April 14, 2025

Notes and Observations

- **Combined Models Generally Outperform Separate Models:** Models 2 (nn_model_combined.py) and 4 (nn_combined_model_lat_long.py), which employ combined or hierarchical architectures, tend to show better performance, particularly on the latitude and longitude/XYZ prediction tasks, compared to Model 1 (nn_model.py) and Model 3 (nn_model_lat_long.py) which use separate networks.
- **XYZ Coordinate Transformation Seems Beneficial:** Models 3 and 4, which incorporate the transformation of latitude and longitude into XYZ coordinates, demonstrate significantly lower Mean Absolute Errors for these predictions compared to Model 1, which predicts latitude and longitude directly.
- **Trade-offs Between Classification and Regression:** There isn't one single model that dominates across all tasks. Some models show higher accuracy for continent and city prediction, while others excel in latitude and longitude prediction.
- **Overfitting:** In several instances, the training accuracy is notably higher than the test accuracy, suggesting some degree of overfitting across the models. This is a common challenge in machine learning and something to consider for future improvements.

Model-by-Model Comparison

- **Model 1: nn_model.py (Right Top Corner)**
  - **Continent Prediction:** Strong test accuracy (89.56%) with balanced precision, recall, and F1-score.
  - **Cities Prediction:** Lower test accuracy (78.75%) compared to continent prediction.
  - **Latitude and Longitude Prediction:** High MAEs (0.3964 and 0.3445). Maps show significant spread between predictions and true locations.
  - **Architecture:** Uses separate networks for each task, which may limit learning of shared representations.
- **Model 2: nn_model_combined.py (Left Bottom Corner)**
  - **Continent Prediction:** Slightly lower test accuracy (88.33%) than Model 1.
  - **Cities Prediction:** Improved test accuracy (81.82%) with higher precision and F1-score.
  - **Latitude and Longitude Prediction:** Lowest MAEs (0.2230 and 0.1876). Very accurate predictions, as confirmed by maps.

  - **Architecture:** Hierarchical structure facilitates learning between tasks.

- **Model 3: nn_model_lat_long.py (Right Bottom Corner)**
  - **Continent Prediction:** High test accuracy (89.31%).
  - **Cities Prediction:** Lowest test accuracy (75.92%) among all models.
  - **Latitude and Longitude Prediction:** Very high MAEs (8.5982 and 21.2971). Severe scatter on maps.
  - **Architecture:** Uses XYZ internally but has poor coordinate prediction, possibly due to flawed transformations or output layers.

- **Model 4: nn_combined_model_lat_long.py (Left Top Corner)**
  - **Continent Prediction:** Slightly lower test accuracy (87.35%).
  - **Cities Prediction:** Highest test accuracy (82.92%) among all models.
  - **Latitude and Longitude Prediction:** Low MAEs (4.5070 and 14.4377), but not as low as Model 2.
  - **Architecture:** Combines hierarchical structure with XYZ usage, effective for cities and coordinates.

Comparative Analysis and Best Model

- **Best for Coordinates:** Model 2 clearly outperforms others in latitude and longitude prediction.
- **Best for Cities:** Model 4 achieves the highest accuracy.
- **Continent Accuracy:** All models perform similarly well.
- **Overall Best Model:** Model 2 (nn_model_combined.py) is the most well-rounded, offering competitive classification accuracy and the best coordinate prediction performance.

Opportunities for Improvement

- **Reduce Overfitting:** Apply techniques like dropout, L1/L2 regularization, or early stopping.
- **Fix Model 3:** Investigate XYZ conversion and back-transformation.
- **Refine Hierarchical Designs:** Experiment with alternative fusion strategies or attention mechanisms.
- **Ensemble Strategies:** Combining predictions may improve performance across tasks.

June 4, 2025

Performing recursive feature selection on the tax_metasub_data.csv we get 300 important features. s

June 5, 2025

Frequently Forgotten Things

- **processed_metasub.csv:** This is all the dataset which contains all the information about different species. This is obtained after processing the raw metasub data. We remove certain cities that are porrly represented and perfrom a bit pre-processing. This dataset should be considered as the starting point for writing any code.
- **metasub_global_git.csv:** This is the dataset that contains the Geographically Informative Taxa (GITs) that are used for training the XGBoost model.

July 19, 2025

I am just keep a track of the things that I have done so far. This part consists of the scripts that I have written and the models that I have created.

/home/chandru/binp37/scripts/ensemble/

This is the root directory for the ensemble modeling scripts.

- `main.py`: This is the main entry point for the ensemble pipeline. It imports all the different models from the subdirectories, handles data loading and preprocessing (specifically for a hierarchical prediction task), and likely orchestrates the training and evaluation of the ensemble.

/home/chandru/binp37/scripts/ensemble/catboost_ensemble/

This folder contains scripts related to the CatBoost model.

- `catboost_classification.py`: Implements a `CatBoostTuner` class for training and tuning a CatBoost classifier. It uses Optuna for hyperparameter optimization and includes methods for training, evaluation, and running the full pipeline.

/home/chandru/binp37/scripts/ensemble/ft_transformer/

This folder is for the FT-Transformer model, a state-of-the-art architecture for tabular data.

- `ft_transformer_classification.py`: Implements a classifier using the FT-Transformer model. It includes `FTClassifier` for training/evaluation and `FTTransformerTuner` for hyperparameter tuning with Optuna.

/home/chandru/binp37/scripts/ensemble/grownet/

This folder contains scripts for GrowNet, a gradient boosting framework using neural networks.

- `grownet_classification.py`: Implements a `GrowNetClassifier` for classification tasks. It uses a series of small MLPs trained sequentially. It also includes a `GrowNetTuner` for hyperparameter optimization.
- `grownet_regressor.py`: Implements a `GrowNetRegressor` for regression tasks, following the same boosting principle as the classifier. It also has a corresponding `GrowNetTuner`.

/home/chandru/binp37/scripts/ensemble/lightgbm_ensemble_model/

This folder is for the LightGBM model.

- `lightgbm_classification.py`: Implements a `LightGBMTuner` class for training and tuning a LightGBM classifier. It uses Optuna for hyperparameter search and provides functions to train and evaluate the model.

/home/chandru/binp37/scripts/ensemble/random_forest/

This folder is for the Random Forest model.

- `randomforest_classification.py`: Contains a `RandomForestTuner` class that defines a complete pipeline (`run_complete_pipeline`) for tuning, training, and evaluating a Random Forest classifier.

/home/chandru/binp37/scripts/ensemble/simple_nn/

This folder contains scripts for simple, fully-connected neural networks.

- `nn_classification.py`: Implements an `NNTuner` class to find the best hyperparameters for a neural network classifier using Optuna.
- `nn_regression.py`: Contains an `NNTuner` class for tuning a neural network regressor, optimizing for a regression metric.

/home/chandru/binp37/scripts/ensemble/xgboost_ensemble/

This folder is dedicated to XGBoost models.

- `xgboost_classification.py`: Implements an `XGBoostTuner` class for hyperparameter tuning and training of an XGBoost classifier using Optuna.
- `xgboost_regression.py`: Defines a `XGBoostRegressorTuner` class, which currently specifies default parameters for an XGBoost regressor.

/home/chandru/binp37/scripts/grownet/

These scripts implement different versions of the GrowNet model, a gradient boosting framework that uses neural networks as weak learners.

`hirarchical_grownet.py`

This script implements a specialized, hierarchical version of the GrowNet model tailored for a multi-task prediction problem. Its primary goal is to simultaneously predict a hierarchy of geographical labels: continent, city, and geographical coordinates (x, y, z).

- **Hierarchical Structure:** The model architecture is explicitly hierarchical. The prediction for continents is used as an input feature for predicting cities, and both continent and city predictions are used to predict the final coordinates.
- **Multi-Task Learning:** It uses a combined loss function to train all three tasks concurrently.
- **Learnable Uncertainty:** The script employs learnable uncertainty weights (`log_sigma`) to automatically balance the contribution of the loss from each task (continent classification, city classification, and coordinate regression). This helps prevent one task from dominating the training process.
- **Custom Boosting:** It uses a custom gradient boosting approach where each new weak learner is trained to correct the residuals of the combined ensemble.

`grownet_classification.py`

This script provides a more standard implementation of the GrowNet model for multi-class classification tasks. It follows the core principles of stage-wise training of weak learners.

- **Stage-wise Training:** The model is built by sequentially adding weak learners (`WeakLearner`). The first learner is trained on the original features, while subsequent learners are trained on the original features concatenated with the cumulative predictions from the previous stages.
- **Corrective Step:** After all weak learners are trained individually, a global "corrective step" is performed to fine-tune all model parameters together, allowing the weak learners to adjust to each other.
- **Simpler Boosting:** This implementation's boosting mechanism is based on passing the cumulative output to the next learner, rather than explicitly calculating and fitting on gradients and Hessians.
- **Trainer Class:** It includes a `GrowNetTrainer` class that encapsulates the stage-wise training and corrective step logic.

`grownet_classification_revised.py`

This script is a revised, more robust, and theoretically grounded implementation of GrowNet for classification. It more closely follows the principles of traditional gradient boosting algorithms like XGBoost, but with neural networks.

- **Gradient-based Boosting:** Unlike the previous script, this version explicitly computes first and second-order gradients (gradients and Hessians) of the loss function. Each new weak learner is trained to fit these gradients, which is a more direct application of the gradient boosting framework.
- **Global Corrective Network:** It introduces a separate, global corrective network that is trained at the end to fine-tune the entire ensemble's predictions, offering a final refinement step.
- **Wrapper Class:** It features a high-level `MicrobiomeGrowNetClassifier` wrapper class that simplifies the entire workflow, including data splitting, training, evaluation, and plotting results.
- **XGBoost Comparison:** The script includes a function to directly train and compare the performance of the GrowNet model against a standard XGBoost classifier on the same dataset, providing a useful performance benchmark.