

National College of Ireland

Project Submission Sheet

Student Name: SAYALI VISHWAS BHAWALKAR, ANUJ SHASHIKANT BHOGLE
.....

Student ID: X23321750, X23288426
.....

Programme: MSc. IN ARTIFICIAL INTELLIGENCE
.....

Year: 2024-2025
.....

Module: ENGINEERING AND EVALUATING ARTIFICIAL INTELLIGENCE
.....

Lecturer: PROF. ABDUL SHAHID

Submission Due Date: 26/03/2025.....

Project Title: MULTI-LABEL EMAIL CLASSIFICATION.....

Word Count: 1630.....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the references section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature: SAYALI VISHWAS BHAWALKAR, ANUJ SHASHIKANT BHOGLE
.....

Date: 26/03/2025.....

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. Projects should be submitted to your Programme Coordinator.
3. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
4. You must ensure that all projects are submitted to your Programme Coordinator on or before the required submission date. **Late submissions will incur penalties.**
5. All projects must be submitted and passed in order to successfully complete the year. **Any project/assignment not submitted will be marked as a fail.**

Office Use Only

Signature:

Date:

Penalty Applied (if applicable):	
----------------------------------	--

AI Acknowledgement Supplement

[Insert Module Name]

[Insert Title of your assignment]

Your Name/Student Number	Course	Date

This section is a supplement to the main assignment, to be used if AI was used in any capacity in the creation of your assignment; if you have queries about how to do this, please contact your lecturer. For an example of how to fill these sections out, please click [here](#).

AI Acknowledgment

This section acknowledges the AI tools that were utilized in the process of completing this assignment.

Tool Name	Brief Description	Link to tool

Description of AI Usage

This section provides a more detailed description of how the AI tools were used in the assignment. It includes information about the prompts given to the AI tool, the responses received, and how these responses were utilized or modified in the assignment. **One table should be used for each tool used.**

[Insert Tool Name]	
[Insert Description of use]	
[Insert Sample prompt]	[Insert Sample response]

Evidence of AI Usage

This section includes evidence of significant prompts and responses used or generated through the AI tool. It should provide a clear understanding of the extent to which the AI tool was used in the assignment. Evidence may be attached via screenshots or text.

Additional Evidence:

[Place evidence here]

Additional Evidence:

[Place evidence here]

Engineering and Evaluating Artificial Intelligence

Abstract- In this project for the Engineering and Evaluating Artificial Intelligence module, we designed and implemented two recent architectural strategy, Chained Multi-Output Classification and Hierarchical Modeling, to expand a modular AI based email classification system into a multi label classification framework. This work was conducted using the Extreme Programming (XP) principles in order to demonstrate the modularity, separation of concerns, abstraction, and maintainable code encapsulation. This project delivered functional AI models not only by collaborative development, version controlled and focused evaluation, but it also added to applied machine learning architectural best practices.

INTRODUCTION

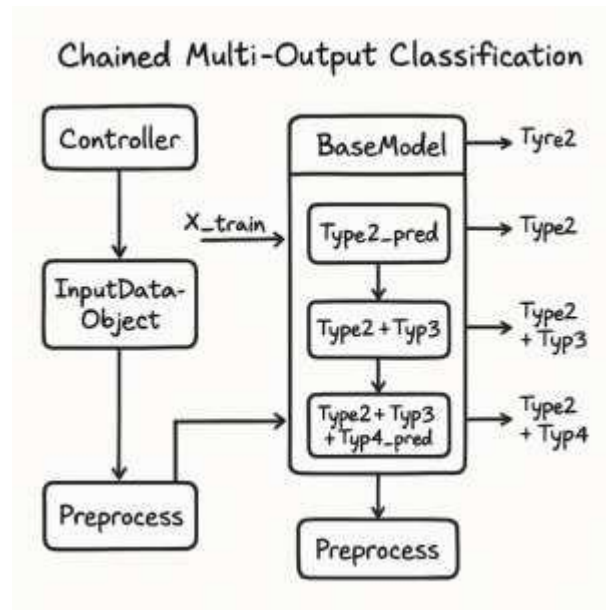
The problem space revolves around Email classification, where a single email can belong to more than one categories (Type2, Type3 and Type4) and thus requires involve a multi label classification approach. Multi-label classification systems, on the other hand, distinguish themselves from multi class classification, where each instance should be classified to one of the classes, by required to predict multiple labels for a given input. It also complicates the fact that labels are interdependent, which cannot be effectively captured with naïve independent classifiers. Thus, the goal was to use such dictionaries in order to introduce structured architectures capable to leverage those dependencies and produce more accurate and interpretable predictions.

All the project was modularized into ordinary components that could be replaced (in line with XP practices). Collaborative code development was done with a strict version control policy enabled in GitHub, feature branches, and the addition of good commit messages. This ensured traceability and accountability for contributions. In the implementation, the structure was based on five main components including the classification (Chained Multi Output Classifier and Hierarchical Classifier), preprocessing logic (preprocess.py), configuration settings (Config.py), and a command line (main.py).

DESIGN DECISION 1: CHAINED MULTI-OUTPUT CLASSIFICATION ARCHITECTURE

The Chained Multi Output classification approach utilizes a sequential or chained style in order for the prediction pipeline to model label dependencies. Let the core idea be as below: It provides a causal or conditional architecture of how labels are coupled; such that downstream predictions are determined by upstream decisions.

SKETCH DIAGRAM 1: CHAINED MULTI OUTPUT CLASSIFICATION



This architecture has been built using a class Chained Multi Output Classifier that inherits a generic class Base Model. The chain of models is each an instance of Random Forest Classifier from scikit-learn. y_{type2} is predicted by the input features X with the input features X picked up as input and the generated y_{type2} was returned to the first classifier. On top of current feature set, X , the second classifier is trained on an extended feature set consisting of both X and $Type2$ prediction, which targets y_{type3} . Thirdly the 3rd classifier is given $X, pred_type\ 2, pred_type\ 3$, and is used to estimate y_{type4} .

This way of chaining makes sure that label type dependencies are honored. While it does introduce error propagation as an aberration upon which an incorrect early prediction could damage later labels. However, this risk can provide interpretability and an intuitiveness of the label interactions.

In this architecture, key data element includes $Type2+Type3+Type4$ concatenated feature, original input matrix X_{train} and individual target labels (y_{type2} , y_{type3} , y_{type4}). The instance wise predictions are stored, and accuracy for each input condition is conditioned where latter label predictions are only valid if the previous predictions are accurate.

Function is called between the controller and model classes in order to carefully encapsulates connectors between components. Structured data are passed without preprocessing, modeling, and evaluation.) The modularity of the system to extend, replace or refactor it is well facilitated.

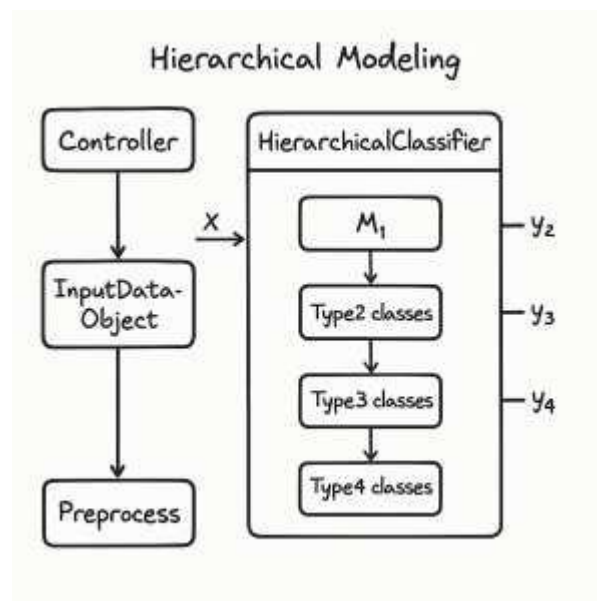
ARCHITECTURE DETAILS: CHAINED MULTI OUTPUT

Element Type	Details
Components	Controller, BaseModel, ChainedMultiOutputClassifier, preprocess.py, Config.py
Connectors	Controller calls preprocessing and model training/predict methods. Data passed as objects.
Data Elements	X_{train} , y_{type2} , y_{type3} , y_{type4} , concatenated labels like $Type2+Type3$, accuracy per instance

DESIGN DECISION 2: HIERARCHICAL MODELING ARCHITECTURE

With the Hierarchical Modeling strategy, the problem is tackled by means of a decision tree like architecture. The structure of this is a classification task in multiple stages. Next, the dataset is applied with the previous prediction and filtered at each step to build a new model from that subset. The first model predicts Type2. Then, for each unique type of Type2 classifier, a type of Type2 to type of Type3 classifier is trained. Secondly, for any combination of Type2 & Type3, another classifier is created for them with an attempt to predict Type4.

SKETCH DIAGRAM 2: HIERARCHICAL MODEL



Working within this tree-based architecture, learning that is more localized along each label path can lead to improved performance by reducing class imbalance and learning specialized decision boundaries. To give a very specific example, if the classifier was trained only on examples in which Type2 is equal to the 'Complaint' label, it will become more context specific in the way that it understands what Type3 label means in that case.

For this architecture, the Hierarchical Classifier class is created and inherits Base Model class. The classifier maintains an internally dictionary of models, each key is a class path (type2_class,), (type2_class, type3_class), etc. First, filtering a subset of data relevant to each key and training each model, produces a fine-grained predictive model.

This architecture has the connectors in the form of the output of one model becoming the filtering criterion for the following stage. For example, the data are in turn sub set according to predictions for the Type3 stage, and this proceeds down the hierarchy. To fulfill this design, one lays a large emphasis on encapsulation and loose coupling of the layers.

This model contains the same input features X , the target variables, however, y_{type3} and y_{type4} are not entire arrays but rather subsets corresponding to their respective hierarchical path. It performs evaluation instance-wise, but only if previous predictions are correct in a manner that mimics a progressive evaluation model.

Although the hierarchical approach provides the best performance at specialized paths, model management and training time increase as the number of unique class paths increases. However, here, it is an essential consideration of the tradeoff between performance and scalability.

ARCHITECTURE DETAILS – HIERARCHICAL MODELING

Element Type	Details
Components	Controller, Base Model, Hierarchical Classifier, preprocess.py, Config.py
Connectors	Output from model at Type2 used to filter data and drive input for Type3 models; similar for Type4.
Data Elements	X, y_type2, y_type3, y_type4, filtered sets per class path, instance-level evaluation accuracy

CUSTOM EVALUATION STRATEGY

The custom conditional accuracy evaluation framework was used to assess both architectures. A prediction for a deeper label type (e.g., Type3 or Type4) can be only deemed valid if all its predecessor's predictions are also valid. This method provides a notion of cascaded responsibility, that models that are more accurate at each level get a reward and a less accurate one incurs a penalty for an early misstep.

In the case of chained architecture, this meant that Type3 predictions were only evaluated when the type to which the prediction was mapped matched with the actual label for both Type2 and Type4. This progressive validation logic was applied to hierarchical models based on the filtering paths. A real-world application of such a custom metric makes more sense, because a correct final label has no usefulness if the earlier context was incorrectly classified.

IMPLEMENTATION AND INTEGRATION

The codebase was broken down into modular form. It first defined some base model interface, Base Model, that had essential methods such as train(), predict() and evaluate(). This base class was extended by both Chained Multi Output Classifier and Hierarchical Classifier. With such strategy we could abstract the model and reuse the code across many model implementations, under a very consistent interface.

Data was loaded, cleaned, and label was extracted in a dedicated preprocess.py script. Constants and model parameters such as test size, random seeds, or model hyperparameters for example were stored into config.py; meanwhile, main.py was the controller script which orchestrated the whole process from data loading to evaluation and the visualization.

Two architectural extensions were introduced, namely chained_multioutput.py and hierarchical_modeling.py. Although these modules were kept in XP driven modular interface, they remained compliant with it. The base estimator is Scikit-learn's Random Forest Classifier as commonly used because it is robust and easy to interpret. Stratified splits were used to train and test the models to have a representative label distribution.

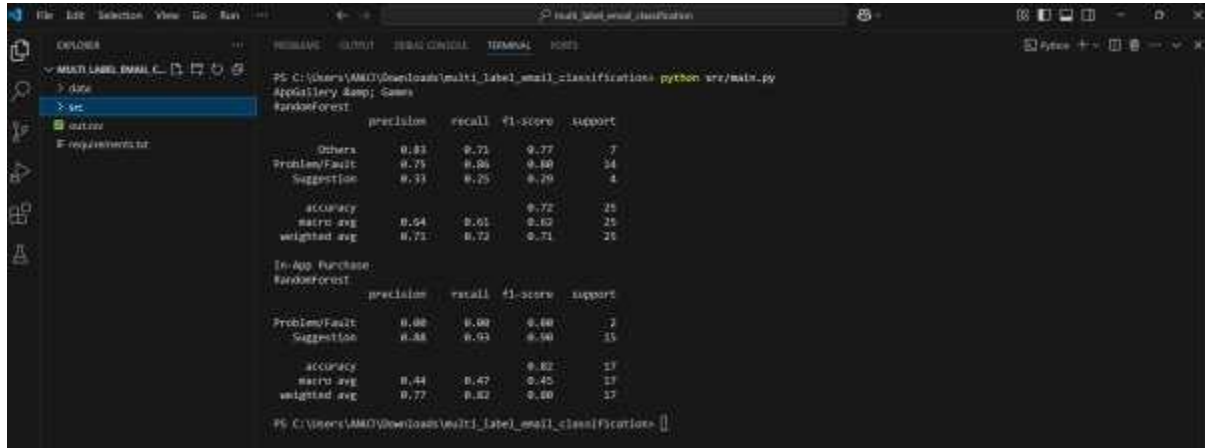
OBSERVATIONS AND RESULTS

However, the chained model was able to capture inter label dependencies well, but it was vulnerable to error propagation. After one of the predictions of Type2 was incorrect, most other label prediction also drifted off. Nevertheless, the chaining was helpful to the model in some strongly dependent paths with label dependencies, thus demonstrating that this strategy can be useful with highly dependent labels.

Subsequently, the hierarchical model proved more resilient to changes in the label variability. It was able to learn localized class specific nuances better by training localized models. But

its training time increased and the memory consumption became larger than expected since managing a multiple classifier instances. Results were evaluated to demonstrate the improvement in the accuracy for the labels on complex paths and especially with multi labels combinations with low support.

Both strategies outperform a simple one-vs-rest approach in terms of label-wise precision and instance-based accuracy. This success adds additional support to the use of architecture aware modeling in multi label problems.



```
PS C:\Users\ANUJ\Downloads\multi_label_email_classifications> python src/main.py
AppGallery & Games
RandomForest
precision    recall  f1-score   support

   Others      0.83      0.73      0.77         7
 Problem/Fault  0.75      0.86      0.80        34
  Suggestion    0.33      0.25      0.29         4

 accuracy      0.72         26
 macro avg      0.64         26
 weighted avg    0.71         26

In-App Purchase
RandomForest
precision    recall  f1-score   support

 Problem/Fault  0.68      0.68      0.68         7
  Suggestion    0.58      0.53      0.56        15

 accuracy      0.82         13
 macro avg      0.44         13
 weighted avg    0.77         13

PS C:\Users\ANUJ\Downloads\multi_label_email_classifications>
```

CONCLUSION

Demonstration of a practical project of how a sensible architecture choice can turn a basic classification system into a robust, and modifiable and scalable neural machine learning pipeline. We overcame the challenges of multi label email classification in the structured and interpretable light, with the use of Chained Multi Output Classification and Hierarchical Modeling.

The project operates in the Extreme Programming with respect to modular interfaces, encapsulated design, and agile collaboration. The tradeoffs between the two architectures are between elegant dependency modeling with simpler implementation versus specialization at the cost of complexity.

GITHUB LINK:

<https://github.com/AnujBhogle/X23288426---CA1-Multilabel-email-classification/upload/main>