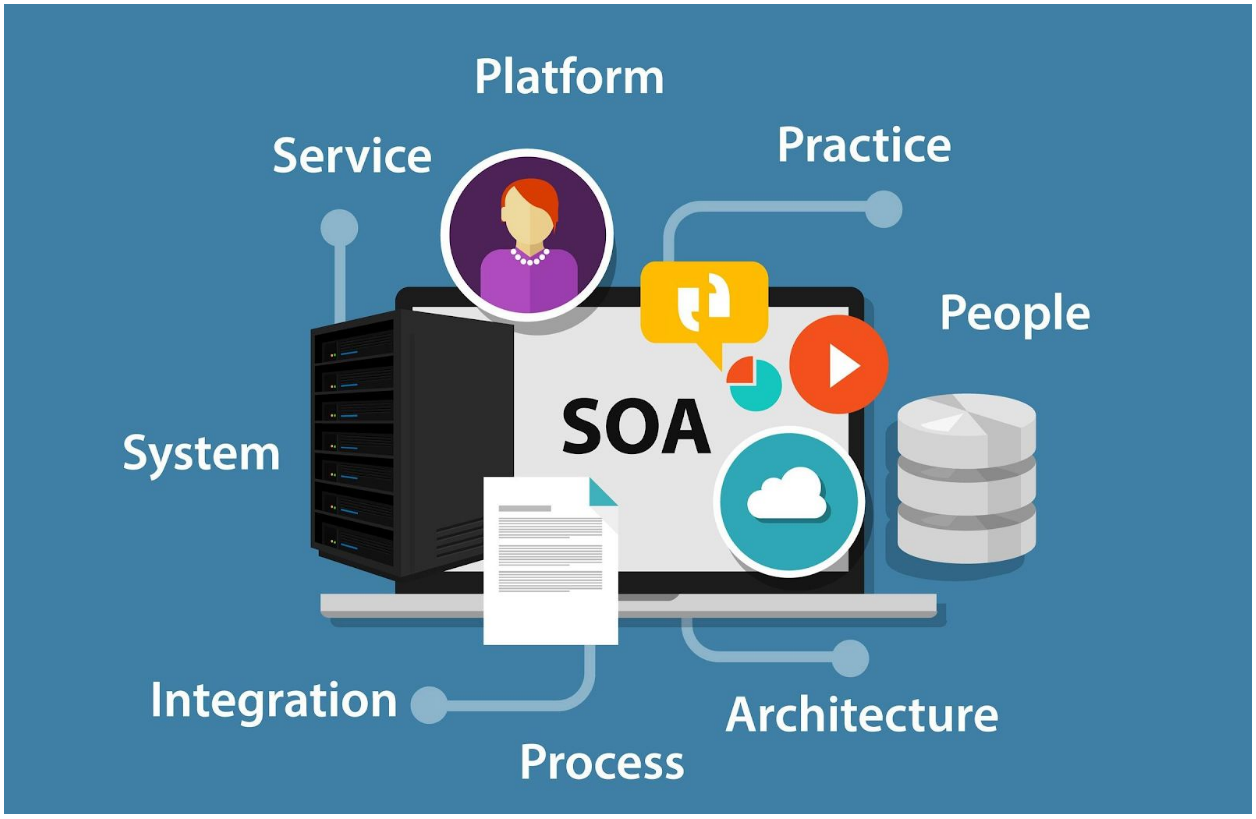
Software Architecture Documentation



Team: Yuki

Team Member: Chaoying Liu, Xin Li, Yuncong Wu, Xiaoyun Ye & Zhiyu Li

REVISION HISTORY

Date	Author	Description
Jun. 08. 2019	Zhiyu Li	The initial organization of the document created. Added the first draft.
Jun. 14. 2019	Zhiyu Li	Some modification based on the suggestions of Jeff (client) and Cliff (mentor).
Jun. 15. 2019	Xiaoyun Ye	Add ARM introduction
Jun. 15. 2019	Xin Li	Add Data Read Class UML
Jun.15.2018	Chaoying Liu	Add Classification Module of Detailed Functionality Design
Jun. 17. 2019	Yunchong Wu	Add Class description of the Data Reader
Jun. 31. 2019	Zhiyu Li	Final Modification
Aug. 6. 2019	Zhiyu Li	Upload

INTRODUCTION

Background of the Project

Nowadays, new computer security threats are increasingly emerging. In particular ARM-based architectures present new and attractive targets for compromise.

- Adversaries are attacking IoT and embedded systems built with ARM processors
- Defenders need to re-calibrate x86-centric low-level code analysis toolchains from the ground up
- Tool developers need a better understanding of new code analysis challenges by measuring inconsistencies among emerging binary analysis solutions covering ARM
- Researchers get a foundation to build new code analytics for defense systems in response to new threats against IoT and embedded

The goal of our project is to develop a system to identify and prioritize features from ARM binaries that are meaningful to ARM-based malware analysis.

About this document

This document contains the software architecture for the Feature Analysis System project, covering the following aspects:

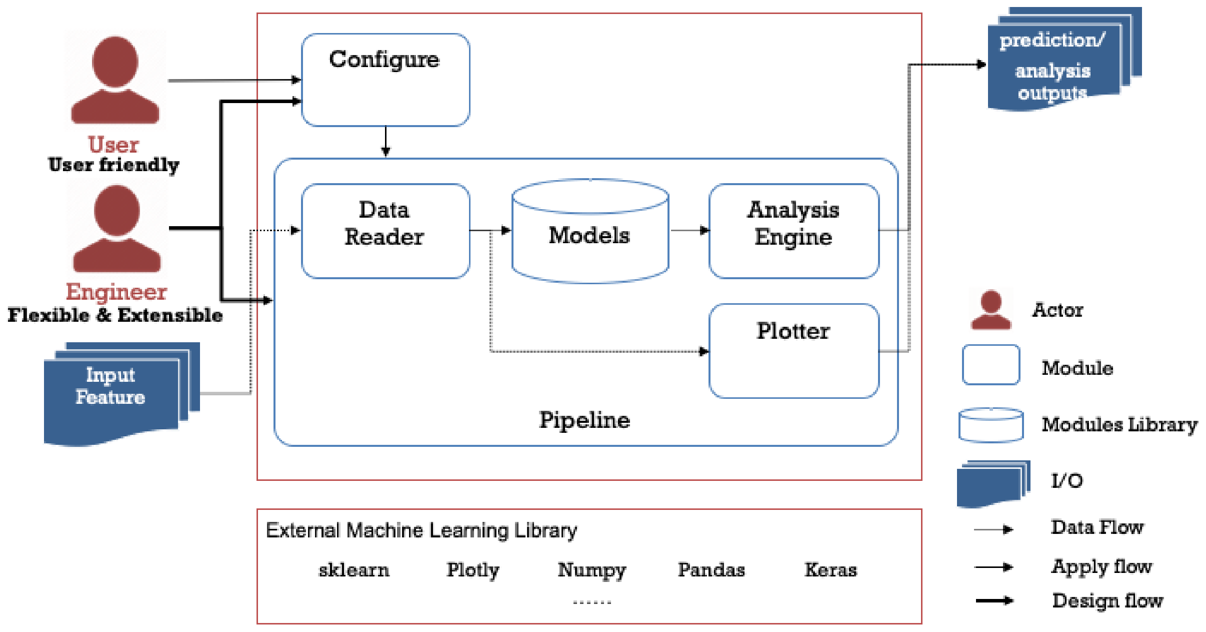
1. **Overall System Design:** A description of the project, the context, and business case for the system being developed.
2. **Implementation Architecture:** A description of the implemented architecture using the component and connector, module and allocation.
3. **Detailed Functionality Design:** A description of the detailed design of important aspects of the system functionality. This is a primer of the detailed design techniques used by the development team and of the dynamic model of the system representing the flow of logic within the system.
4. **Progress and Tracking:** The priority, plan and progress tracking of system design and implementation.
5. **Extensibility:** Architectural alternatives for the possible extensions to the system, and how well the alternatives accommodate these extensions.

Intended audience

The report is intended for the following audience:

1. Stakeholders (direct or indirect) of the project,
2. Users that might want to get an insight into or analyze the architecture and design of the project, and
3. Engineers that have an interest in documentation of the architectures and design of software systems, or want to expand and extend the program.

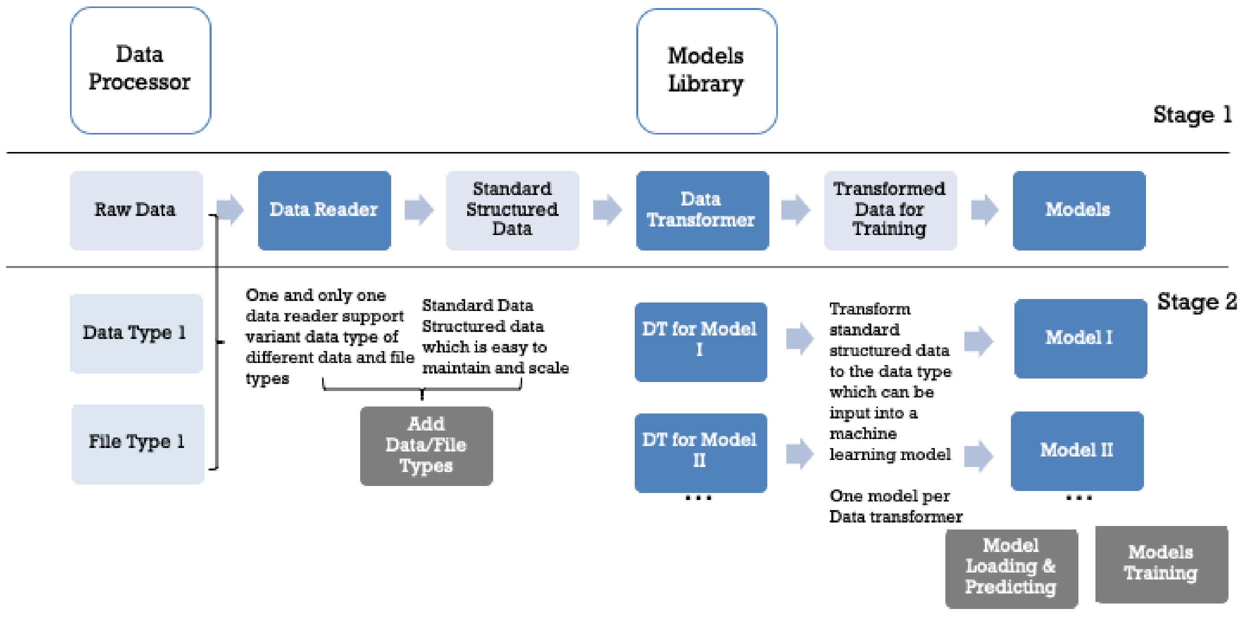
OVERALL SYSTEM DESIGN

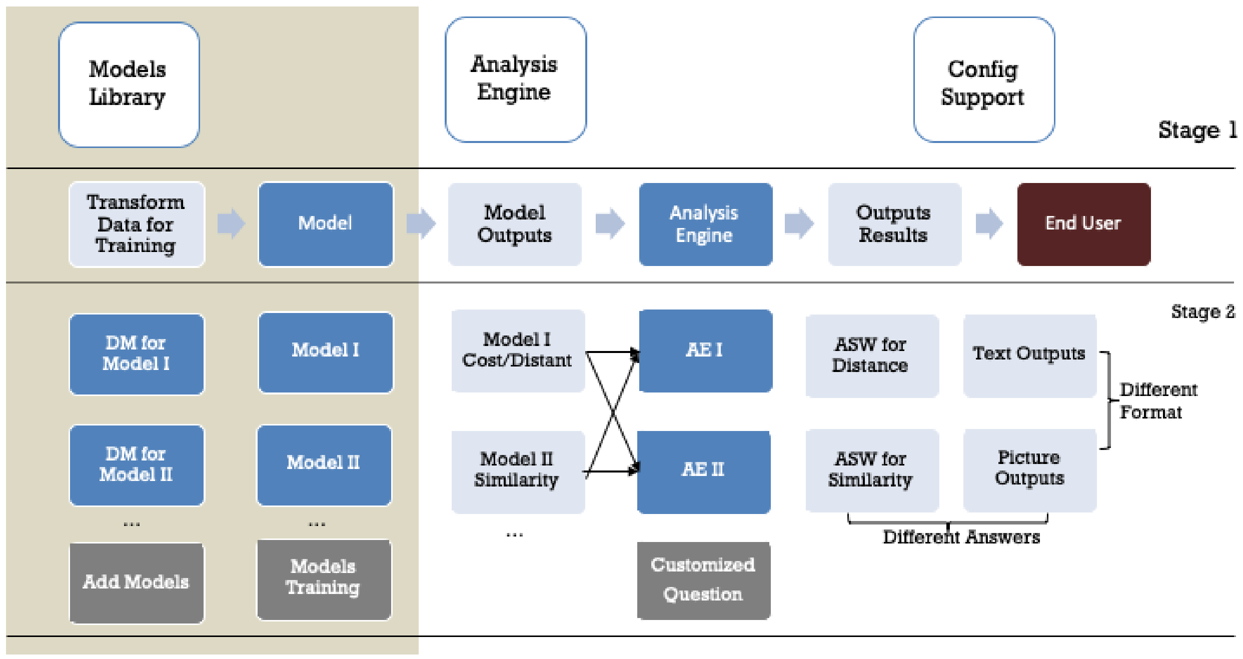


There will be two stages of this project:

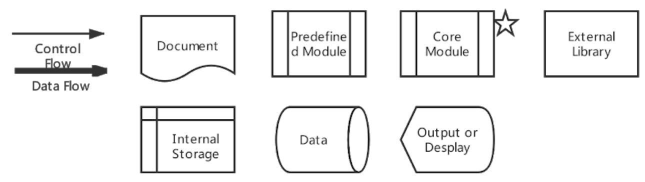
1. First Stage: End to end platform for user
    a. One single model
    b. Certain question(s)
    c. Input feature
    d. Prediction/analysis outputs

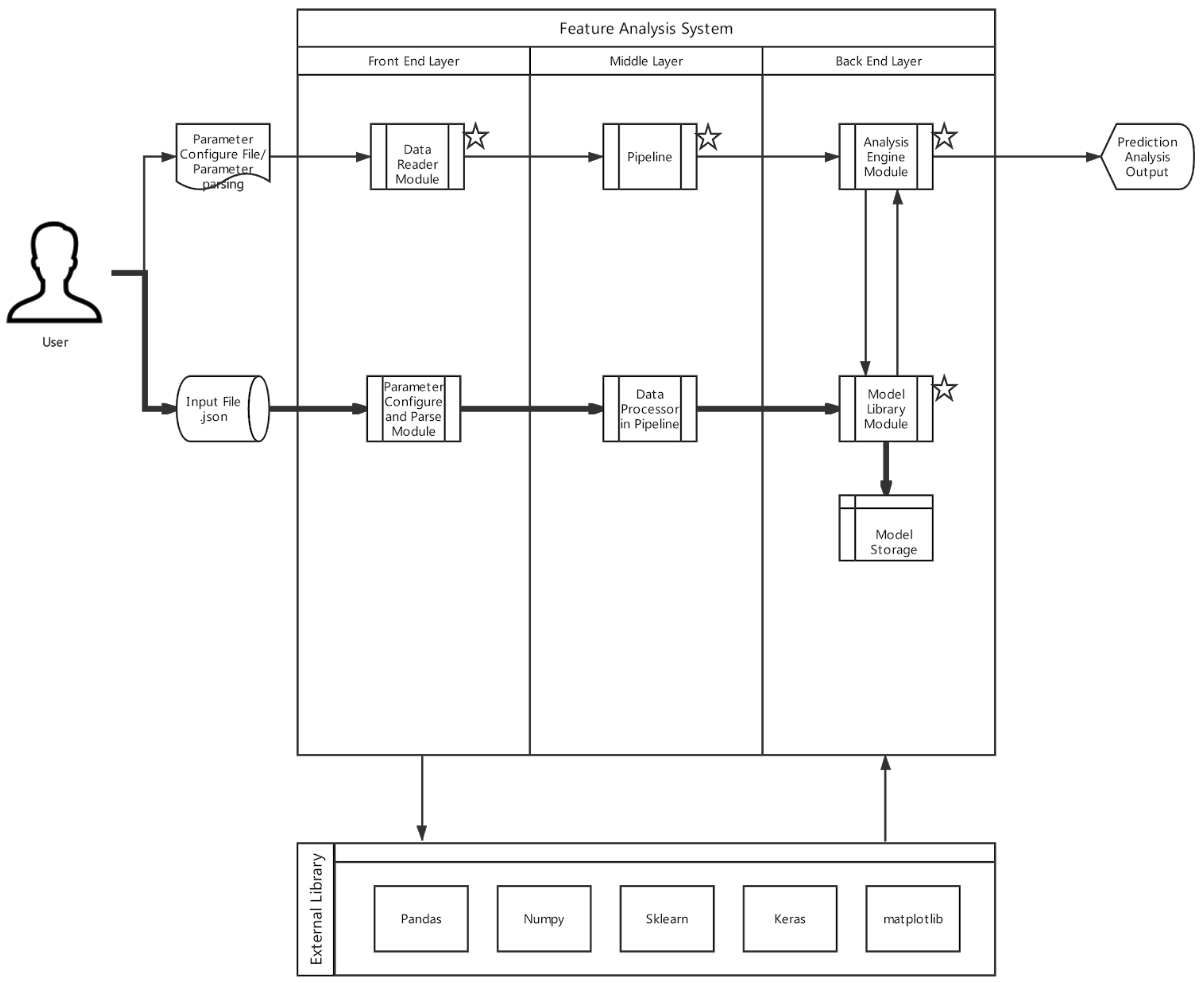
2. Second Stage: Extensible platform for the engineer
    a. Multiple Customized model
    b. Multiple Customized questions
    c. Input features with different data and file types
    d. Customized prediction/analysis outputs w.r.t. questions


Data Processor
Models Library
Stage 1
Raw Data
Data Reader
Standard Structured Data
Data Transformer
Transformed Data for Training
Models
Stage 2
Data Type 1
File Type 1
One and only one data reader support variant data type of different data and file types
Standard Data Structured data which is easy to maintain and scale
Add Data/File Types
DT for Model I
DT for Model II
Transform standard structured data to the data type which can be input into a machine learning model
One model per Data transformer
Model I
Model II
Model Loading & Predicting
Models Training

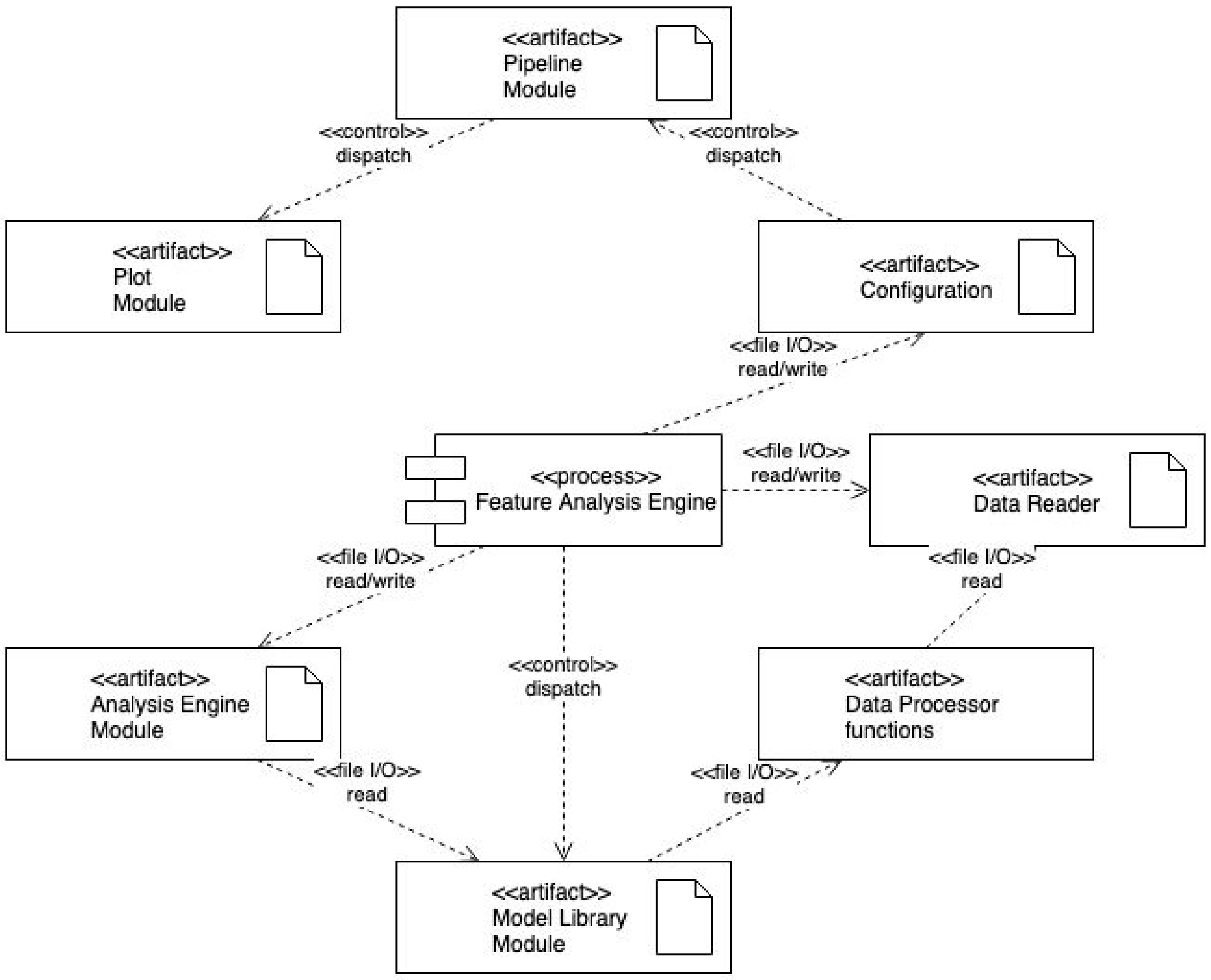


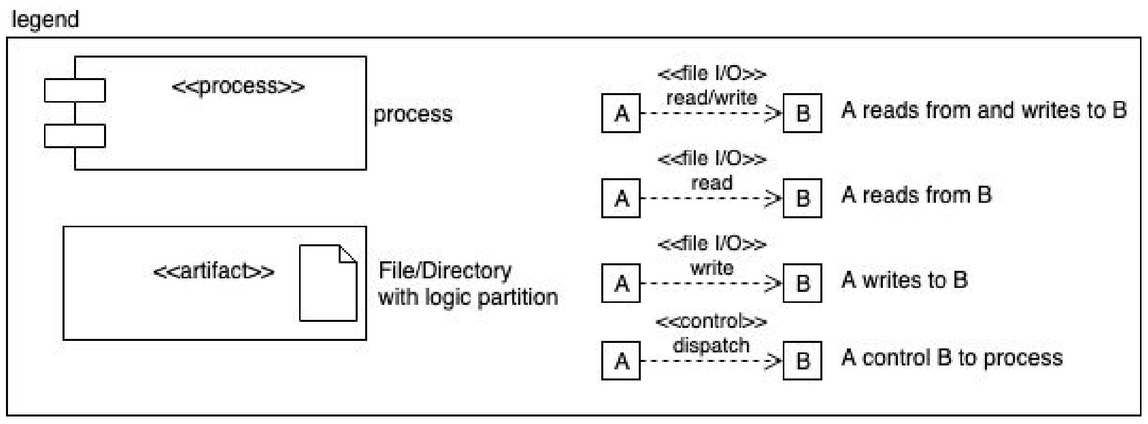
CONTEXT DIAGRAM





COMPONENT & CONNECTOR DECOMPOSITION LEVEL I





Detailed Functionality Design

1. Data Reader Module

Description:

This module is used to read all formats of input data, transfer them into other format or read data from other DataReader and transfer them into different types of data format.

Class description:

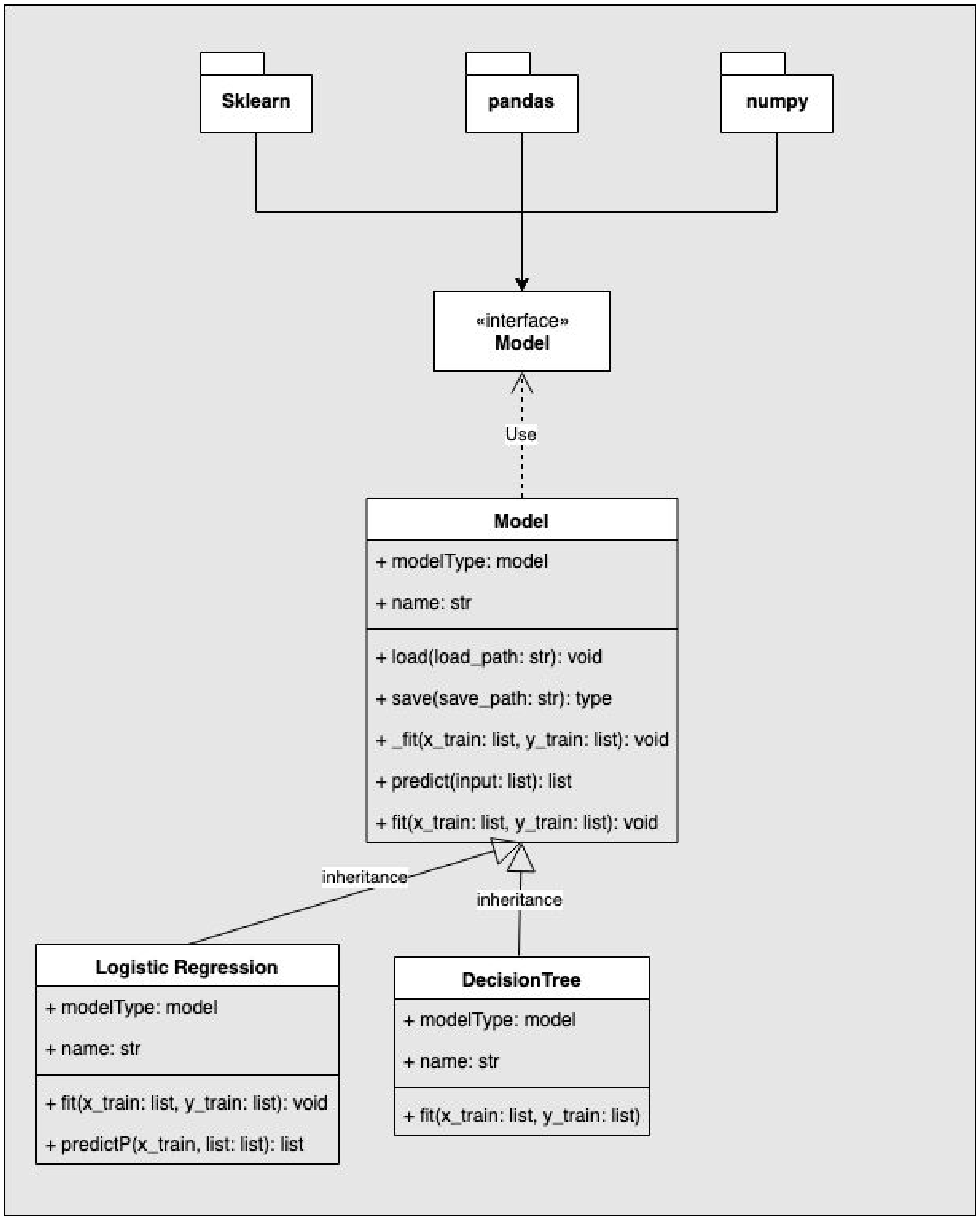
DataReader should extend class MLPipeLine.DataReader and implement 6 methods to make it works:

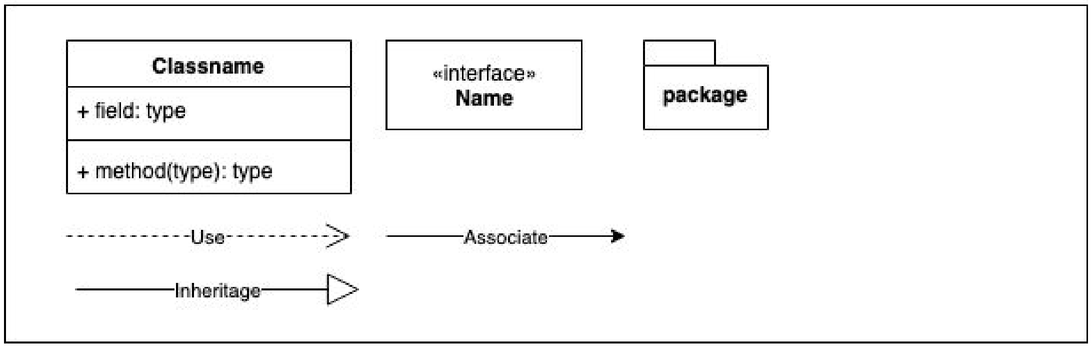
1. readFromPath(path)
   Read data from file path.
2. readFromData(data)
   Read data from other DataReader.getData()
3. getBatch(batch_size)
   Return a new DataReader containing only batch_size of data
4. resetBatch()
   Reset the iterater of data
5. __iter__() / __next__()
   DataReader should support iterator to iterate data.
6. getData()
   Return data for next DataReader

2. **Model Library Module**

   Description:

   This module is used to support model training, addition, saving and loading.


Sklearn
pandas
numpy
«interface»
Model
Use
Model
+ modelType: model
+ name: str
+ load(load_path: str): void
+ save(save_path: str): type
+ _fit(x_train: list, y_train: list): void
+ predict(input: list): list
+ fit(x_train: list, y_train: list): void
inheritance
inheritance
Logistic Regression
+ modelType: model
+ name: str
+ fit(x_train: list, y_train: list): void
+ predictP(x_train, list: list): list
DecisionTree
+ modelType: model
+ name: str
+ fit(x_train: list, y_train: list)


Classname
+ field: type
+ method(type): type
«interface»
Name
package
Use
Associate
Inheritage

model_zoo.py
The file includes the following major function:
Model:
Models class serves as the **common interface** of all the models. There are 6 common functions in this class:
load: load data from the pre-trained model
save: save the trained model to the local directory
_fit: internal function for the training process of the model
fit: abstract fit function to be overridden by specific models
predict: predict the result given the input testing dataset

Specific Models:
Specific models are referred to as the actual machine models like Logistic Regression Model or Decision Tree in the picture.
Specific models are required to
fit: specify the model class and call the train function of the model
predict_p: predict the probabilities of the dataset if supported

Extensibility and Tradeoff

Add model:

If we want to add a new specific model, we should establish a new class extending the model class and override the fit function. The fit function should specify the model type (i.e. model class in sklearn) and the training parameters like the learning rate of regression or max depth of decision tree.

Given the model class implemented as the interface for each model, it is extensive friendly to add new model into the system. Basically, only the model name, the model type should be specified in the field and fit function should be overridden in the methods.

Add new feature type

If we want to add new models, sometimes the data requires further transformation before feeding into the machine learning model. Since the transformation is quite customized, it is extremely hard to summarize common modules.

Therefore, new functions should be implemented in the pipeline before input into the model to be trained.

3. **Analysis Engine Module**
   Description:
   This module is used to do analysis from give predict result.
   Class Description:
   analyser.py
   The file includes the following major function:
   similarity: perform similarity analysis between 2 pieces of data.
   __call__: special method triggered when the instance of a class is called.

   Extensibility and Tradeoff

   Add new analysis function:
   If we want to add new analysis function, we should specify the functions, and add the functions into the __call__ function which can trigger the new functions when the analysis class is called.

4. **PipeLine**
   Description:
   In this class, the data will be processed step by step as a pipeline

   Pipeline.py
   The file includes the following major function:
   __init__: Initial pipeline class
   _validate_steps: validate the steps in the pipeline to check whether it is allowed
   run: process the data step by step
   _iter: an inner iterator function to iterate the steps
   named_steps: get name step dictionary
   _get_item_by_index: an inner function allows obtaining item from key or index reference
   __getitem__: is used to implement calls like class[key/index] returning the corresponding step
   fit: is used to train the model with pipeline, the data would be preprocessed before input into the model

__call__: special method triggered when the instance of a class is called. It is the same as **run**

Extensibility and Tradeoff

Design new pipeline:

The Pipeline class allows the engineers to design arbitrary steps including model training, analysis, and plot. The pipeline would simply run each step to input the data and get the result after processed in the step.

Some regulations should be followed:

1. Model class, analysis class, functions, methods, and numpy arrays are allowed in the initialization of pipeline
2. For model training, model class should be in the last step, since it is meaningless to add new functions afterwards;
3. Analysis class should be put in the last step, since analysis class would print the results without return value
4. Plot class should be put in the last step, since plot class would show the pictures without return value

All of the above rules are checked in the_validate_steps function

Add new type of step:

If we want to add a new type of step, the step function should be specified. And _validate_steps function should be updated if new regulations should be updated.

5. **Data Process Module**

Description:

There is no python file for data process module. We achieve full flexibility by allowing user to add custom function or method (function of a class) in the pipeline.

Extensibility and Tradeoff

Add new data process module:

arbitrary numbers or types of function are allowed to use in the pipeline, if the function specify the input and the return value (output).

6. **Plot Module**

   Description:

   This module is used to draw some pictures for the outputs.

   plot.py
   plot class:
   __init__: initial the plot class
   draw: draw the picture given input data. Plot x vs y if x and y are both 1 dimensional array, or plot first 2 dimensions of y if y has multiple dimensions.
   __call__: special method triggered when the instance of a class is called.

   Distribution: draw the distribution picture - a distplot picture with x: value, y: count
   Correlations: draw the correlation picture - a heat map with correlation/similarity between columns

   Extensibility and Tradeoff

   Add new plots:

   New plots can be added after specifying the plot type, parameters, data to show.

7. **Configure Module**

   Description:
   A short-cut Module allowing engineering to build customized command line parser.

   Config.py

Config class
A class to decode and manage parsing parameters.

parse: parse parameters into the class
state_dict: get all config parameters by checking with attributes of the class
to_dict: serializes this instance to a Python dictionary
to_json_string: Serializes this instance to a JSON string.
to_json_file: save this instance to a json file.

Extensibility and Tradeoff

Design customized command-line:

Use ArgumentParser library to add new arguments, parse the parameters to config class, and finally specify the process steps.

No extra codes for interface between command line and augments.

Extra convenience is provided to save and load parameters to and from a json file by using Config class.

QUALITY ATTRIBUTE

Quality Attribute	Performance
Concern	Response Time
Scenarios	**Stimulus:** end-user actions like model prediction, training, analysis. **Source of stimulus:** end-user **Environment:** The actions are predefined in the system. **Artifact:** model library module **Responses:** The results for involved scenarios are displayed. **Responses Measure:** The time (based on the scale of data)

Quality Attribute	Robustness
Concern	Robustness under the changed feature format, model addition.
Scenarios	**Stimulus:** end-user actions like the model selection **Source of stimulus:** end-user **Environment:** The models are predefined in the system. **Artifact:** model library module **Responses:** The function of the model is corrected processed or return an error message if the parameters are not allowed. **Responses Measure:** Smooth functionality and mo system crash. **Stimulus:** end-user actions like changing the input data format, or inputting a piece of data with the different data format **Source of stimulus:** end-user **Environment:** The input data format are predefined in the system. **Artifact:** Data Reader **Responses:** All the differences are identified by the program, the function is corrected process or return an error message if the formats are not allowed. **Responses Measure:** Smooth functionality and no system crash. **Stimulus:** end-user actions like inputting a piece of data with empty or NaN data. **Source of stimulus:** end-user **Environment:** The input data format are predefined in the system. **Artifact:** Data Reader **Responses:** The empty and Nan data will be automatically filled by 0 (digits) or “” (string) **Responses Measure:** Smooth functionality and no system crash.

Quality Attribute	Extensibility
Concern	Extensibility of Program
Scenarios	**Stimulus:** end-user actions like the model addition **Source of stimulus:** end-user **Environment:** The model common interface is predefined in the system. **Artifact:** model library module **Responses:** the model interface can be inherited by the new model class, saving modification time. **Responses Measure:** The lines of code to add a new model. **Stimulus:** end-user actions like the data type addition **Source of stimulus:** end-user **Environment:** The data common interface is predefined in the system. **Artifact:** Data Reader/Transformer **Responses:** the data type are specified in the program, all the data type should follow the rules/API. **Responses Measure:** The lines of code to add a new data type.