Computer Science Capstone

CS 486 - Requirements Engineering

SmartState

Project Title: Event-driven MachinE Learning, Intelligent Assessor (EMELIA)

Software Design

Overview:

The purpose of this document is to present this project at a more technical level, in addition to showcasing our software design and project architecture. This document will describe the components and modules that will define our machine learning system.

Team Members:

David Rodriguez

Andrew Hurst

Reed Hayashikawa

Jianxuan Yao

Jesse Rodriguez

Clients:

Jon Lewis

Aaron Childers

General Dynamics Mission Systems

Capstone Mentor:

Fabio Marcos De Abreu Santos

Northern Arizona University

School of Informatics, Computing, and Cyber Systems

Version: 2.0

Date: 2.12.2020

1. Introduction

General Dynamics Missions Systems is a global aerospace and defense company that develops solutions in all areas such as land, air, and cyber domains. General Dynamics develops and maintains an advanced command, control and direction-finding communications system to execute search and rescue missions. The current communications system used by General Dynamics relies on a ticket system to generate reports regarding system or hardware failures. These tickets contain data such as a hex code representing the error, a unique ID, location for where the error occurred, and many more.

Each category such as the ones previously mentioned must be classified according to the type to assist system engineers at General Dynamics. System engineers manually parse through the tickets in order to classify the type of error reported. This process is inefficient and reduces our client's ability to respond to system failures. In order to properly address the problem at General Dynamics, we will create an Event Driven MachinE Learning Intelligent Assessor (EMELIA) that will effectively classify these reports.

With the current issue of engineers at General Dynamics manually categorizing these "tickets", we are working closely with our clients Aaron Childers and Jon Lewis to build a system that will automate this process to be more efficient. Aaron Childers and Jon Lewis are two systems engineers that will oversee our project. Aaron and Jon are part of a larger team of engineers that oversee all of the complex communications systems.

This project consisted of functional requirements, non-functional requirements, and environmental constraints that must be met in order to be successful. Functional requirements listed for this project include but not limited to, the system must be able to take in large volumes of data at a single time, the system must utilize a supervised learning model, and the system must classify ticket data using predefined categories. Some non-functional requirements include, the

system should be able to classify data at 90% rate of accuracy, the system must be maintainable and scalable, and the system must be able to integrate successfully with the existing framework. Lastly, the environmental constraints that need to be followed include software version requirements and components that must be used in order for this project to be successful. Some of these constraints include, Python 3.6 needs to be used to develop this system, Tensorflow 1.13.1 will be utilized in order to create and execute our model, and Keras 2.2.4 will assist us in implementing the functionality of the learning model. Thus, the purpose of this document is to make implementation plans explicit and allow us to recognize any problems at the early stages of development.

2. Implementation Overview

Our system will accept the input of a CSV file format, representing several "tickets". We will then retrieve this data by implementing a program that reads parameters through the command line interface. The platform we chose is anaconda, which helps us keep the same version, reduce error generation, reduce the impact of errors, and help our team troubleshoot. Then through pandas to accept various file formats, and in Python, they are converted into dataframe objects, a data structure with highly centralized data organization.

The structure of these tickets will be similar, but may vary depending on the number of categories or categories within each ticket. However, there are several categories that are consistent throughout the series. We will use Python to create a program that will help us simplify the algorithm to reduce the workload. The program will parse each ticket and extract the data, store it in a series of structures for easier access, and process it in the machine learning model. We will train 80% of the tickets to get the error rate we need, and then gradually reduce step size to reduce the value of error rate. Finally, we analyze the overfitting and underfitting through the charts from training data, so as to reduce the value of error rate.

Then we will use our machine learning algorithm to collect and analyze data. This algorithm takes tensorflow as the framework, because tensorflow is an open-source machine learning platform. The data flow chart is used for numerical calculation. The interface is simple and consistent, which provides clear feedback for program errors. It consists of advanced APIs and a flexible ecosystem. It is easy to operate and to integrate into our system. Tensorflow provides resources such as a high-level API such as keras to implement the Python language, which allows faster prototyping and production.

Then, the program will continue to test the results shown in the model evaluation step through scikit learn, which contains a set of neural network models, and can use supervised learning to classify the data. Supervised learning is the process of learning the function that maps the input to the output based on the sample input-output pair. If the result does not meet our minimum accuracy of 90%, the program will return to the data preparation / extraction step for further evaluation and repeat the following steps until the minimum accuracy of 90% is achieved.

Assuming the program has completed all the steps, the next step is to return the results to the customer. We will integrate a solution to output the metrics to the console and to a new CSV file. After allocating memory to the results, the CSV file is updated for user access.

3. Architectural Overview

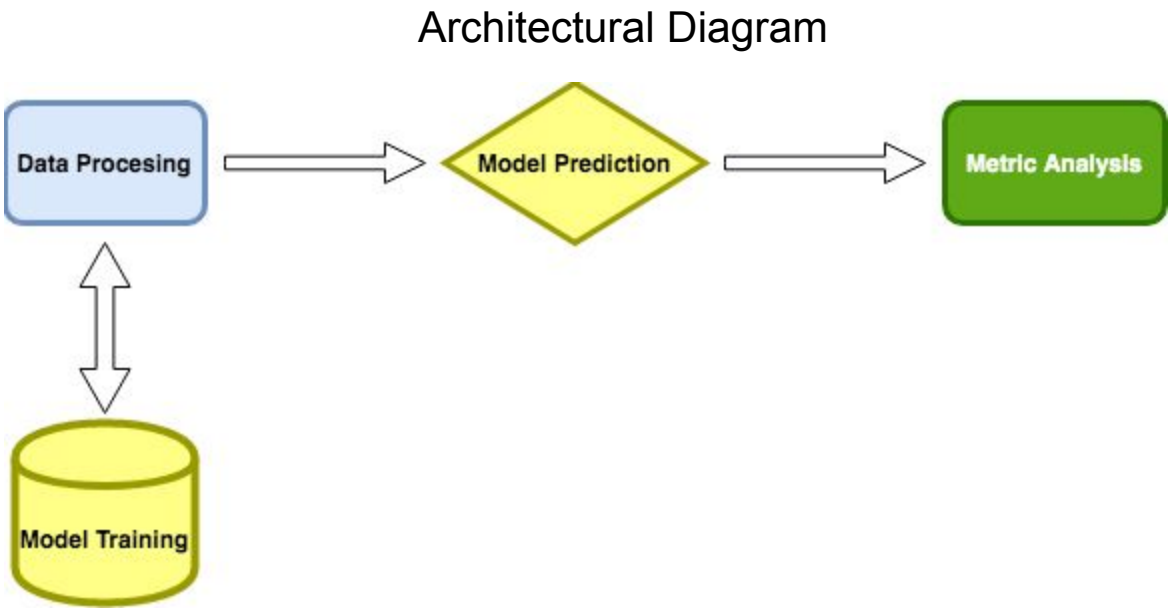

Figure 1: Overview of EMELIA

3.1 Discussion of Architecture

3.1.1 Key Module Features

Understanding EMELIA requires a basic understanding of the key responsibilities and features of each module. The system is composed of four major components: Data Processing, Model Training, Prediction Model, and Prediction Metrics. The combination of all these components produces an accurate predicting system for General Dynamics Error Reporting Ticketing system. Each module plays an important role in that process.

The data processing module essentially formats the chosen tickets and the relative information in such a way for the Neural Network Prediction Model to compute. However, the data formatting will vary slightly for input for the prediction model and training the model. The Prediction Model will take the formatted data and use machine learning to predict the corresponding classifications for the data. The Metrics Analysis Model will take the output from the Neural

Network and consolidate it in a readable fashion, producing output files and provides metrics on the system's performance. The last component, the Training Module, will not be essential to the process every time the program runs. It will only be used for initially training the Neural Network and retraining when new features to predicting are introduced.

3.1.2 Control Flow and Communication

Control of the program will be dictated from a command-line environment. Arguments will be passed from the terminal to the main driver program. The EMELIA architecture will be structured in a procedural fashion; Each module's functionality will be called from the driver and data will flow through there between modules.

3.1.3 Architectural Influences

The architectural influences shaping the software are a high percentage of procedural architecture, as previously mentioned. This is a result of our programming requiring more of a pipeline of processing structure. However, the Neural Network model will be organized as an object allowing the user to store, save, and recreate. Overall the system will implement procedural methods with small object-oriented influences.

4. Module and Interface Descriptions

This section will provide detailed descriptions of the modules and interfaces that will be utilized in creating EMELIA. The interface of this system will only include exporting of data and/or functionality and importing of data and/or functionality depending on the responsibility of the module. There will be no graphical interface for this system. All commands will be ran via a command line. The purpose of this "port system" allows each module to access data or functions in a sequential manner that will make troubleshooting simple.

Included with every module is a detailed description of the correlation that the module has with the other components of the project, interface mockups, and overview of the coding practices needed to have a successful implementation. Before discussing each module, we have developed the following system diagram for the entire process that our project will undergo from data processing to output of the data.

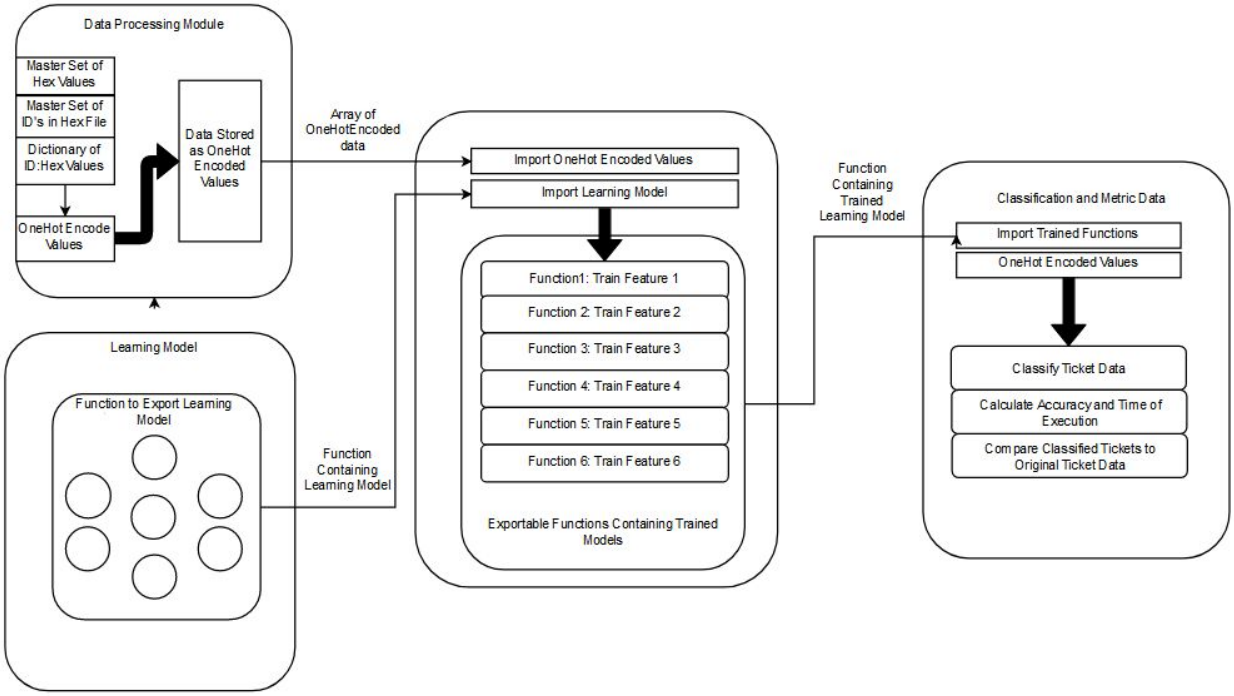

Figure 2: System diagram for EMELIA

4.1 Data Processing

4.1.1 Component Responsibilities

This component will be responsible for acquiring all necessary ticket data and alarm data needed to pass to the neural network layers located in a different module of the system. This module will configure the data properly using basic coding practices in order for it to be passed to the neural network. Our system will need to be able to receive input data in the format of a CSV file representing several “tickets”. We will then retrieve this data by implementing a program to read

in the arguments through the command-line interface. Below is a diagram of the two csv files and the relationship that exists with the contents of the files.

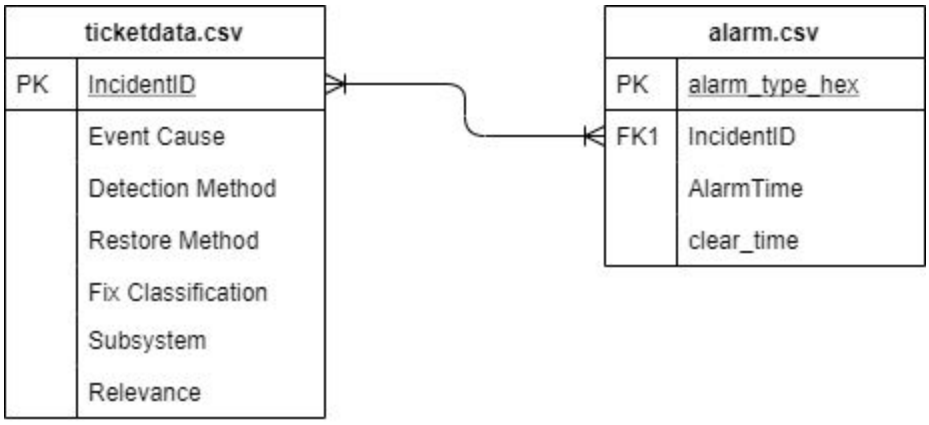

Figure 3: Relational diagram for ticket data and alarm data

Figure 4: Unique hex and ID values stored in sets

Since the ticket and will be in two csv files this module will have to parse through each “ticket” and join the two tables of data to make all incident identifiers for tickets and alarm hex codes available in a single file. These tickets will be similar in structure but may differentiate by the number of classifications or category values as seen in Figure 3. With that being said, this module will need to create data structures containing both ticket and alarm data that can be used within this module along with the other modules. In order for this structure to be accessed in the future, this module will store OneHotEncoded ticket data in an array. The program will need to utilize function calls to OneHot encoded ticket data that will serve as input to the learning model.

The following figure is an example of the array structure that will be implemented to store corresponding ticket data:

```
[
        [ID1, [0x23456], [Label1, Label2, Label3, Label4, Label5, Label6]],
        [ID2, [0x23457], [Label1, Label2, Label3, Label4, Label5, Label6]]
        [ID3, [0x23458], [Label1, Label2, Label3, Label4, Label5, Label6]],
        [ID4, [0x23459], [Label1, Label2, Label3, Label4, Label5, Label6]],
        [ID5, [0x23450], [Label1, Label2, Label3, Label4, Label5, Label6]],
        [ID6, [0x23451], [Label1, Label2, Label3, Label4, Label5, Label6]],
        .
        .
        .
        [Nth ID, [Nth Hex Values], [Nth Label Values]]
]
```

Figure 5: Unique ID's with Corresponding Hex Codes & Labels

The purpose of this array will be to store all related data in an ordered fashion, while remaining accessible to developers. The values in this nested 2-Dimensional array will allow the developers to OneHot encode its contents, and make comparisons to prediction data in later modules.

4.1.2 Program Module Relationship

This module has a direct relationship with other components and advances data to downstream modules. During this processing component, a master list of data will be created that will be used for relational mapping in other modules. In order for the training model, which is the next module in the system, to be accurate the data needs to be configured correctly and thoroughly examined to prevent loss of training data and reduce bias in the data provided to the neural network model.

4.1.3 Program Interface

This module will interface with the other modules in the system by moving data down the pipeline. Data will be stored in an array and exported to the next module for use. Since this

module is the root of all functionality for the other modules, it is crucial that we implement this module so that it organizes data in a way that accurately reflects each ticket. This is to ensure the data can be used for the training module and be successfully displayed for the output metrics.

4.2 Training Model

The neural network model will be used to train on the test data provided by the client. The learning model will be contained within a single function, inside a module, to be exported for use by each of the features that will need to be classified for the ticket system.

4.2.1 Component Responsibilities

Construction of the learning model is the primary responsibility of this component. The learning model will be constructed within a single function. Each of the models will be part of a larger pipeline that feeds data into a learning model.

The number of dense layers will not be dynamically created. Since the learning model will be "tuned" to create the highest accuracy results, the user(s) will not be able to control the number of dense layers or specify a different activation function for the model. The activation function will be a "softmax" function in order to generate confidence value prediction data as an output of the model.

4.2.2 Program Module Relationship

This module will be exported to the model classification module within the system. This module is only meant to construct the learning model. By modularizing the system in this way, the learning model can be adjusted by future users and engineers without negative side effects or need for reconstruction. The implementation for this module provides the highest amount of integration and the least amount of coupling to the ticket data and the resulting classification of ticket data.

4.2.3 Program Interface

The interface for this module will be easy to implement and use by the design team. Since Python is the programming language used in this project, the team will need to import the file containing the learning model and make a single function call to a function containing the learning model. This design will be most scalable for future development and allows engineers to import and run the learning model for classification or perform operations in the module without the learning model.

4.3 Model Classification

4.3.1 Component Responsibilities

This component contains the learning model that will be stored for each feature that will be classified regarding each ticket. The model will be trained and store the state for each of the neural network models which are specific for each feature. The state of the learning model will be saved to a .md5 file or .hd5 file. Each model will be wrapped inside a function and will be passed ticket feature data via a parameter. By bundling each of the trained neural networks in a function, these trained networks can be exported to the output module for comparison to the test data.

4.3.2 Program Module Relationship

The module will be responsible for importing the neural network function created in the previous module. The function that is imported will be used on six separate occasions to train on each feature of ticket data. The ticket data will be imported from the data processing module as an array. The array will contain OneHot encoded values that will be used as input for training. Data processing will be a prerequisite for this step to successfully train on the data that is provided. This module will be the third step in the pipeline of data. The file will construct and export its functionality to be used downstream. It simply exports all the functions used to train the neural network learning models to the next available module.

4.3.3 Program Interface

This program will export all functions to be used by the output module. The functions that contain the trained learning models will be explicitly named so that developers can differentiate between them in the driver file. The functions from this module are not codependent and may be used in any order within the driver file.

4.4 Output Metrics

4.4.1 Component Responsibilities

This module is responsible for reporting the time needed to perform classification of ticket data. The accuracy of the classification model will need to be reported to the user from this module. This module will contain a function that will call the function containing the learning model, and will pass the ticket data as a parameter to the function. By exporting all required functions to a single module, the team can accomplish the following goals:

- Make successive calls to each of the functions that contain the trained neural networks
    - Successive calls will make more accurately record the time taken to complete program execution
- The file that contains OneHot encoded values will be made available to functions that require input
    - In this case, the column containing all relevant ticket data will be able to provide the input necessary to run the functions
- Predicted data by the learning model can be stored in this file
    - This will make comparison of predicted ticket data to test ticket data more convenient for the development team
    - A simple function can be used to compare the values from the predicted ticket data to the test data → The goal is to automate the comparison in order to generate an accuracy (confidence of predicted values) reading

4.4.2 Program Module Relationship

This module will import the ticket data from the data processing module and the trained model from the model classification module. This will be the last stage of the pipeline in the system. The classified data and the metrics related to system performance and accuracy will be printed to a log file. This log file will inform the client of the speed and the quality of the learning model to classify ticket data. Since the classified ticket data will be in the log file, the client can manually review the classification data to prove accuracy of the system.

4.4.3 Program Interface

This module will utilize the interface functionality from the other modules in the system. This module will not contain an interface to be used in other modules. The reason this module will not use an interface is that it is the final step in the data pipeline for this system. All data should be prepared and assorted into a variety of data structures. The neural network will have been constructed by system developers in the training module. Once the data has been passed to the trained model, the trained model will be imported for use by the output module. All available interface data and functionality will be used in this module. The system will perform predictions utilizing test data that has remained unseen during the training stage. The success of EMELIA will be determined by the output of this module.

5. Implementation Plan

This section will focus on the full schedule for the development of Emelia for the Spring semester of 2020. It will be divided into the main phases outlined in the the Gantt chart displayed in Figure #6.

5.1 Planning Phase

5.1.1 Refactor Previous Prototype

As a team, we reviewed and refactored our previous prototype from the Fall 2019 semester to be more compatible with our client's data.

5.1.2 Plan Software Design

During the preparation/planning phase the team planned out the overall structure of the project by breaking down the individual components and brainstorming how we would implement each part.

5.1.2 Initial Interview with Client

Our team also scheduled a physical meeting with clients Jon and Aaron. Lastly, we designated tasks to each team member as outlined in the Gantt chart in Figure #6 according to each team member's individual strengths as programmers.

5.2 Phase One: Data Formatting

5.2.1 Export Alarm Data to Master Dictionary

In order to properly prepare the raw data for training the model, David will create a master dictionary that contains all the Incident ID numbers and their corresponding hex codes.

5.2.2 Create Data Structure for Classification

Reed will create a simple data structure that contains ticket ID numbers, their related hex values, and the possible classification features for a given ticket.

5.2.3 Create Functions for One Hot Encoding Data

Andrew will be responsible for creating a complex function that performs One Hot Encoding to any data structure that is passed into it. One Hot Encoding transforms any given data into its binary representation that the learning model can understand.

5.2.4 Create Dictionary for One Hot Encoding Data

After creating the function for One Hot Encoding the data, Jesse will create a separate dictionary that will hold each of the possible hex values with their corresponding One Hot Encoded value.

5.2.5 Construct Test Matrix

Yao will create the test matrix that contains the ticket ID, hex codes, and classification values.

5.2.6 Prepare Data for Training Model

Once these tasks are completed, the team will conduct additional testing to complete the Data Formatting Phase of the development of EMELIA.

5.3 Phase Two: Learning Model

5.3.1 Input Data for Training Model

Reed will focus on training the learning model by importing the train data and labels for the learning model and then input a portion of that data once the Neural Network has been constructed and properly trained.

5.3.2 Create Neural Network Structure

David will utilize the previous prototype in order to create the Neural Network to train the data.

5.3.3 Testing and Refactoring of Neural Network

Once the Neural Network is created, Yao will be responsible for testing and refactoring the hidden layers to improve the overall functionality of the model.

5.3.4 Create a Way to Save the State of the Model

Reed will be responsible for conducting additional research in order to save the state of our model after the training is complete.

5.4 Phase Three: Output Metrics

5.4.1 Format Output Metrics

Jesse will be responsible for formatting the output metrics into a format that is usable to our clients.

5.4.2 Compare Predictive Value to Test Value

Andrew will be responsible for inputting the test data into the learning model and comparing the predictive values to the test values to determine the accuracy of the learning model.

5.4.3 Driver Program Structure

David will work on the Driver program structure to perform a sequence of operations to control the functionality of EMELIA.

5.4.4 Unit Testing

Yao will be responsible for testing the learning model to ensure that the behavior is what we expect it to be.

5.4.5 Finalize Complete Prototype

Lastly, the whole team will work together to complete the base prototype. The team is confident that this implementation plan will help the development of the project.

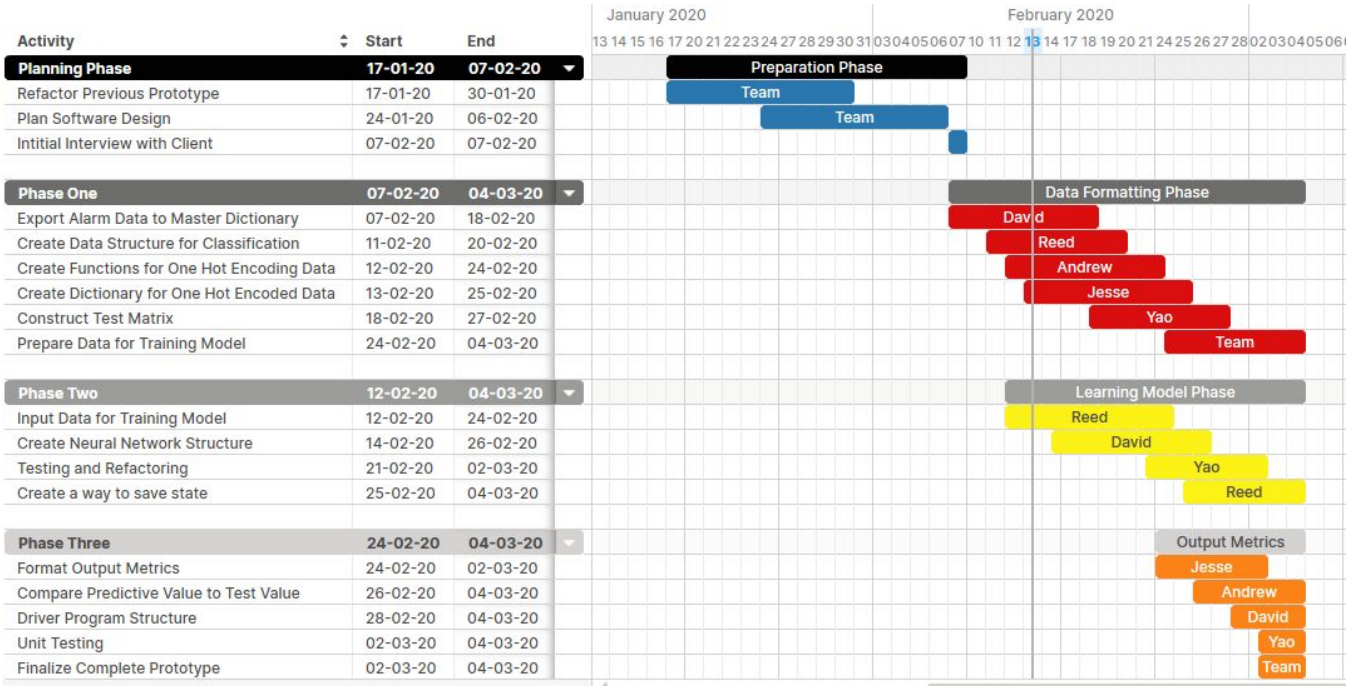

Figure 6: Schedule for Development of EMELIA

6. Conclusion

In summary, our team's software Emelia will be utilized to increase the efficiency of our client's existing ticket system. Currently, Rescue 21 utilizes hundreds of communication towers throughout the United States. The maintenance for such a large infrastructure is handled by a team of software engineers who spend at least 20 minutes manually classifying individual tickets, which can be exacerbated by incorrectly identifying tickets. With a more efficient ticket classification system, our client will be able to observe long term data trends more efficiently to identify problem areas quickly and improve overall system maintenance.

This document details the planned architecture for our software EMELIA and how it will be implemented. The Implementation Overview outlines the relationship between our software and the dependencies that it needs in order to be used correctly. The overall structure of our software will include a series of functions that process ticket data, train the learning model, and come up with predictions for how each ticket will be classified. This design will help our team develop our software to present to our clients Aaron Childers and Jon Lewis.

Despite the upcoming challenges regarding implementation for this system, we are confident that our software will fulfill our clients' requirements. We are also confident that we will overcome any technological difficulties in the upcoming development phases of our project. We hope to have our main phases of development completed within the first week of March.