CS CAPSTONE DESIGN DOCUMENT

DECEMBER 4, 2019

REDUCING PATIENT DOSE FROM DIAGNOSTIC IMAGING USING MACHINE LEARNING

PREPARED FOR

OREGON STATE UNIVERSITY

DR. STEVE REESE _____ _____

Signature *Date*

PREPARED BY

RADIOLOGICAL COUNTING

IAN BROWN _____ _____

Signature *Date*

SEAN GILLEN _____ _____

Signature *Date*

YIHONG LIU _____ _____

Signature *Date*

Abstract

The capstone team will create a system to arrive at a spectroscopy classification result faster than conventional spectrometers through the use of real-time processing of radiation data. The system will be controlled by a manager module that receives data from the spectrometer and sends it to three trained machine learning models, which prompt for more data until a confidence value is reached. The main focus will be to develop the accuracy and speed of the system enough for it to be a compelling demonstration of the algorithm described in a 2019 patent by OSU and Georgetown University researchers. This document is covered under a Non-Disclosure Agreement (NDA) limiting access to its signers, the project's stakeholders, and OSU employees (including Teaching Assistants).

CONTENTS

1 INTRODUCTION

1.1 Purpose

This design document details the technologies to be used during the development of the machine learning algorithms as well as the rationale behind each decision. After reading this document, a developer should know what components are needed for the project as well as the tools that can be used to implement the solution.

1.2 Scope

The scope of this document includes the design decisions and rationale for each aspect of the machine learning project. For each aspect, the desired functionality will be summarized and the methods, techniques, and/or tools that the team has decided to use to achieve said functionality will be discussed.

1.3 Intended Audience

This document is intended for the project client and the senior capstone advising team. Both the client and capstone advisors will use this document to examine and analyze the project team's design choices as well as monitor their progress during development for the whole year.

2 GLOSSARY

- ADC: Analog-to-Digital Converter. An ADC is a piece of hardware that converts analog signals such as sound waves into digital signals. This allows data to be used by software. The ADC used in this project converts a spectrum of radiation counts into a discrete number of counts for every energy channel.
- API: Application Programming Interface. This describes the functions a developer can use with a third-party tool.
- C4.5, C5.0: Decision tree algorithms.
- Pandas: data processing library for the Python programming language.
- Python: interpreted programming language developed in the mid-1990s. It is known for enabling fast prototyping, and today has a large number of libraries available for data processing and machine learning.
- Tensorflow: math library developed by Google which can be used to apply machine learning models, such as a neural network.
- TRIGA: Training, Research, Isotopes, General Atomics. A class of nuclear research reactor designed and manufactured by General Atomics. One such reactor is present in OSU's radiation center, and equipment used in the reactor bay will be used by this project.
- Weka: Waikato Environment for Knowledge Analysis. An open-source graphical tool enabling quick application of different machine learning algorithms.

3 BODY

3.1 Stakeholders

The main stakeholders for this project are the authors of the patent it falls under. They are:

- Dr. Steve Reese, Oregon State University

- Dr. Ophir Frieder, Georgetown University
- Jessica Curtis, Oregon State University

3.2 Design Viewpoints

3.2.1 Context viewpoint

The system will take radiation counting data from an ADC and return a classification to the user's display. The system's user will be one of the project's stakeholders.

System black box components:

- ADC and equipment it interfaces with on its own
- Computer receiving ADC data
- Computer display/application

The system requires other components to operate, such as the radiation detector itself, and radiation source. These will be ignored for the most part in this design, except for how they affect the ADC's operation and output. This is because the system is only meant to change the classification stage, which happens entirely from ADC data.

The user will start the system by opening the application on a computer in the OSU TRIGA reactor bay. The computer will be connected to the ADC, allowing it to receive data. The user will activate the ADC and the external systems to which the ADC connects in order to start the collection of radiation counts. The application developed will detect the start of output from the ADC, and begin the process of classification.

During the classification process, the system will display real-time outputs from the machine learning models with respective confidence values. Once the confidence values reach a user-set value, the application will send a message to the user. As this version of the system will not control the ADC or radiation source, counting will continue until the user switches off the ADC module or this application, or the ADC reaches its preset time threshold.

3.2.2 Composition viewpoint

The system consists of an ADC, serializer, manager, classifier, and some sort of display. The ADC takes signals from a spectrometer and outputs a file containing data on the counts it performed. The serializer then opens this file and interprets its contents. Then, it constructs a list of data structures that represent this information. The manager will invoke one of three previously-constructed classifiers and feed the list of data from the serializer into it. The classifier itself will take in a data set and attempt to classify it as a specific radioactive material. If the accuracy of the classification is high enough, the manager will stop execution and output the result to one or more displays and/or files. If the accuracy is not high enough, the manager will request more data from the serializer.

While the interval between requests for more data and the threshold for classifier accuracy should be configurable, this project will use 10ms and 95% respectively. Additionally, any reasonable machine learning classifier could work in this system. However, this project will limit these to decision tree, naive Bayes, and neural network.

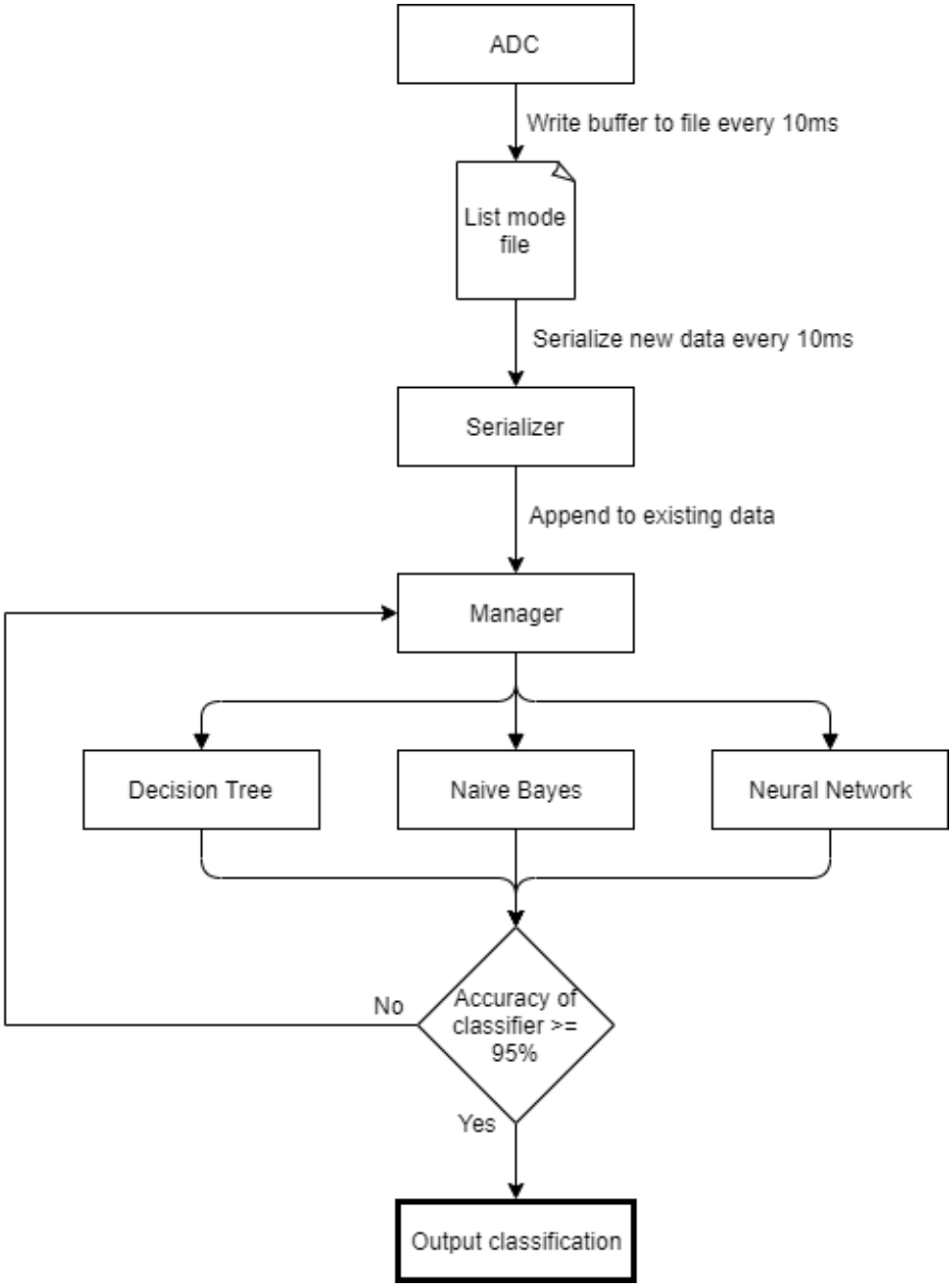

Fig. 1. System diagram

3.2.3 Interaction viewpoint

- `Serializer`
  - `get_data`: Opens the list mode file created by the ADC, serializes the contents to a list of objects, and returns this list. This method will take a byte offset parameter and also return a byte offset. This decision was made so that serialization can be performed incrementally. Every time the method is called, it will start at a location in the file given by the offset parameter, read data for a specified file length, and then

return the serialized data and byte offset at the location of the file in which it stopped. The file name can be parameterized or hardcoded for a slight performance increase if possible.

- `Manager`
  - `main`: The entrypoint of the system. Execution flow will remain here until a classification has been reached with sufficient accuracy. Command line arguments can be passed for configuration changes.
  - `append_data`: This method will perform the call to `Serializer.get_data`. It will also handle updating the working data set with the new data. The byte offset returned by the serializer will also need to be stored in a variable. Then, this can be passed in to `Serializer.get_data` to continue where it left off from the previous call. `Manager.main` will call this method with high frequency, so performance will be important here.
  - `check_classification`: Calls `Classifier.classify` and handles the result. After this call, the accuracy will be compared to the configured threshold. If the accuracy is less than the threshold, control will be passed back to `main` and the loop will continue. If the accuracy is greater than or equal to the threshold, the classification will be passed to one or more external systems, such as a display. A file may also be generated with some results about the overall execution of the system.
- `Classifier`
  - `classify`: Runs classifier on a provided data set. The result of this function should be a classification and a probability representing the estimated accuracy of the classification.

3.3 Design Views

3.3.1 Context view

This system sits between the ADC and the user, so the black box components are only those for the classification part of the process.

3.3.2 Composition view

Since our project will be implemented in Python, the composition was based on the concept of distinct modules. In this context, modules are analogous to classes in the object-oriented programming paradigm. Also, the ADC component and choice of classifiers in the classifier component were previously decided by the patent that the project is based on.

3.3.3 Interaction view

The classifier module will exist as a third-party Python package. Because of this, it is represented as having one main method. The serializer module also has one method, although this will be developed by the group. This is due to the serializer being a fairly simple module. Finally, the idea of the manager is that it will function as a coordinator for the other two modules. It will not export any methods, but it will be the main entrypoint of the system. This module will also be where most of the data being generated resides during runtime.

3.4 Design Rationale

3.4.1 System Interface

Because the main purpose of the project is a proof-of-concept for an algorithm, the interface will not likely be used much by those outside of the development/stakeholder group.

3.4.2 Main Development Language

The group chose to develop the project using the Python programming language. The main benefit to this choice is the simplicity and readability of Python code over other popular languages. Also, since the main goal of the project is to function as a proof of concept, other factors such as performance play less of a part, as long as the code can be understood by the stakeholders. Python also supports several packages such as scikit-learn, TensorFlow, and PyTorch which perform well for applications of machine learning.