Iteration 1

Deadline: December 5th

CSE3063F20P1_RAD_GRP12 _iteration1.pdf

Contents

1 Introduction

1.1 Purpose

The modern world we are in right now is always looking for ways to categorize things, to put things into clusters, to box things. All of this is done to make dealing with differences easier. An example of this, is labeling sentences or words. Companies have long started collecting data and labeling them, then use the results of analyzing them into making their services or products better.

Our project will be a system that can be used to label data by assigning a predetermined labels/classifiers (Ex: positive, negative..etc.) to a group of instances (Ex: comments, text...etc.)

Our audience will be any system that wants to use the same process of labeling/classifying, For example : An online newspaper can use this system to label their news as sports, politics, finance...etc. OR to label customer comments in an e-commerce website as Positive or Negative.

1.2 Scope

This software product which we are developing is used to provide a user-friendly desktop application product that will extract the sentiment feelings of a user towards certain services and products.
In this iteration, our development phase will be focused on making a working prototype of this software.

The goals of our project:

- Providing accurate data labeling results
- Having the ability to use this tool with any type of dataset that fits the user's needs
- User-friendly/easy-to-use desktop UI.

1.3 References

1. IEEE Computer Society. Software Engineering Standards Committee, & IEEE-SA Standards Board. (1998). *IEEE Recommended Practice for Software Requirements Specifications* (Vol. 830, No. 1998). IEEE.
2. https://classroom.google.com/u/0/c/MTgxNjU0NDIyNDBa
3. https://classroom.google.com/u/0/c/MTgxNjU0NDIyNDBa/m/Mzg4NzE2MTc2OTNa/details
4. https://www.toptal.com/
5. https://www.wikipedia.org

1.4 Overview

An overview of everything in the document could be summarized as the following:

- We organized the structure of this document to give the reader an understanding of the problem, then how we are going to fix, then go into detail of the product itself.
- What comes after this section is all concerning the product. We give an overall description of factors that might affect the software solution we are working on, then we talk about its functions, the constraints that would apply on those functions, and finally the assumptions we made based on the user characteristics we have.
- We also talk about the specific requirements including, but not limited to, functional requirements, Performance Requirements, Design Constraints, Software System Attributes, and some Other Requirements.

2 Overall Description

2.1 Product Perspective

This software is neither independent nor self-contained at this point in time. It will most probably in most use cases be a part of a bigger system. It can be used in various fields to analyze, and categorize pieces of information/news/reviews/ and so on. This software solution, however, needs to of course be related to some constraints.
For instance:

- User interfaces: the software is intended to be used to help humans manually label sentences/piece of information or news, this puts a constraint on the user interfaces as we need it to be as simple as possible and function in the most seamless way since it will be used to label (maybe) thousands of lines of data.
- Memory: As mentioned, the software will of course consume memory. However, a constraint on this amount of memory needs to be set as this piece of software will be a component of a bigger software. Because if not put, it might take up too much and make the system crash.

The constraints vary, however, they will become more clear with each iteration and with each time we increase the requirements and make them more concrete.

2.2 Product Functions

Our software is quite simple. The software starts first by loading a .JSON file into its internal structure. Let's call that file a *dataset* for simplicity's sake. Then, *for this iteration only*, the software will go ahead and randomly label each instance provided in that dataset with one of many predefined/pregiven labels. Please note that the labels will also be included in the dataset file. After that, the software will continue to log each action it is taking. That is, logging what instance it labeled, and what label it gave it. After all of this is done, the software generates a .JSON file as an output. It includes the dataset definition, the instances, the labels, the label assignments, and user information.

Within the next iterations, however, the *flow* might change. That's why this section of the document might get updated as well.

2.3 User Characteristics

Our software could be used by any user, providing that they know what the dataset contains. For instance, a dataset that will include sentences that will require sentiment analysis, the label-er should know what to expect. Another example would include named-entity recognition in which multiple phrases or words can be categorized. Our system, with its simple interface, will make the task of labeling much easier, which is an important process nowadays. The users do not need any pre-training to be able to use our program. They only need to input a dataset in the expected format and start using our software right away.

2.4 Constraints

- Our program only accepts input datasets of the format .JSON.
- The dataset should match a specific structure that tells the program of the type of the dataset, maximum labels it can have, and possible labels.

2.5 Assumptions and Dependencies

- How big the datasets are.
- What kind of datasets we are going to be labeling.
- Fixing a mistake in labeling on a large scale will be very costly in terms of time.
- Requirements of our software might change, since the client might need to multi-classify an instance. Meaning, both give it a sentiment-analysis-label, *and* categorize it.

2.6 Apportioning of Requirements

- User Interface

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The customer doesn't require a user interface at the moment. However, the requirements will increase with time, and a user interface is expected to be implemented. So this section of the document is to be rewritten.

3.1.2 Hardware Interfaces

For this iteration the software only requires an operating system with a framework able to run Java programs. For starters, we will limit the devices that will use the software to Computers/Laptops only.

3.1.3 Software Interfaces

For this iteration the following software products are required:

1. JSON Simple:
    - Name: JSON Simple
    - Version number: 1.1
    - Source: http://www.java2s.com/Code/Jar/j/Downloadjsonsimple11jar.htm
    - Purpose: This package is used to parse JSON files.
2. JUnit:
    - Name: JUnit
    - Version number: 5
    - Source: https://junit.org/junit5/
    - Purpose: This package is used to design and manage unit tests in java.
3. Gson:
    - Name: Gson
    - Version number: 2.8.6
    - Source: https://mvnrepository.com/artifact/com.google.code.gson/gson/2.8.6
    - Purpose: This package is used to create ordered and easily readable JSON outputs.

3.1.4 Communication Interfaces

Within this iteration, the software will run offline so there is no need for any communication interface.

3.2 Functional Requirements

3.2.1 Data Labeling System

Data Labeling System is the central class in our object-oriented modelling. It is responsible for creating instances of other classes inside itself and call their corresponding operations when necessary. Overall, Data Labeling System is a “Doing Responsibility” class.

3.2.1.1 Functional Requirement

The first operation which is done by this class is to parse the configuration file, and use the information available in that file to create User and Dataset instances.

3.2.1.2 Functional Requirement

Once all available users and datasets are parsed and stored, this class initiates the process of data labeling. That is, for every user and for every instance available in a dataset, it assigns some label(s) to the corresponding instance.

3.2.1.3 Functional Requirement

After the data labeling process has been finished successfully, this class starts the exporting of labeled data into a JSON file. It reads all the necessary data from a list which was filled during the data labeling phase.

3.2.2 User

User is a class used as a creation tool for users given in the datasets that are loaded to our software. It provides an id attribute, a name attribute, and a type attribute for each user. In general, this class is a “having responsibility”-class. Its responsibility is to carry the user’s type, name, and id.

3.2.2.1 Functional Requirement

This class’s only functional requirement is to store the id, the name, and the type of the users.

3.2.3 Dataset

Dataset is a class which is used as an abstract for a real-world dataset. It includes the necessary attributes which defines a dataset. Moreover, it also keeps a list of all labels and instances it contains. Overall, this class is a “having responsibility”-class. Its responsibility is to store dataset attributes and lists of labels and instances.

3.2.3.1 Functional Requirements

The only operation which is done by this class is to parse dataset related information, process it, and add it to the corresponding fields.

3.2.4 Instance

Instance is a class used as a creation tool for instances of the data pieces given in the datasets given by the user. It provides an id attribute, and a text attribute for each instance. In general, this class is a “having responsibility” class. Its responsibility is to carry the instance’s id and its text.

3.2.4.1 Functional Requirement

This class’s only functional requirement is to store the id and the text of the instance/piece of information that is being labeled.

3.2.5 Label

Label is a class used as a creation tool for instances of the labels provided in the datasets given by the user. It provides an id attribute, and a text attribute for each label. In general, this class is a “having responsibility” class. Its responsibility is to carry the label’s id and its text.

3.2.5.1 Functional Requirement

This class’s only functional requirement is to store the id and the text of the label.

3.2.6 Label Assignment

Label Assignment is a class used as a Label assigning tool for Instances provided in the datasets given by the user.

3.2.6.1 Functional Requirements

The first responsibility is done by the method assignLabels which invokes the Labeling mechanism object passed to it, to label the given instance.

3.2.6.2 Functional Requirements

It holds the following information of a labeling process: a User that does the assigning, a Label to be assigned, and an Instance to do the assignment on.

3.2.7 Labeling Mechanism

Labeling Mechanism is an interface that includes one method that needs to be implemented. The method is called labelInstance and it will be implemented in each class depending on how that certain mechanism works.

3.2.7.1 Functional Requirement

The method labelInstance will take in an instance, an ArrayList of possible labels to choose from, and an integer defining the maximum number of labels allowed per instance. Then, it will return another ArrayList that includes all the labels generated for that instance.

3.2.8 Random Labeling Mechanism

Random Labeling Mechanism is a class that implements the Labeling Mechanism interface. It implements the labelInstance method and adds a new one called getRandomElement.

3.2.8.1 Functional Requirement

This class chooses random labels (based on the maximum number of labels allowed) from the list of available labels and returns an ArrayList of label(s) for each Instance.

3.2.9 Log

Log is a class used for getting a logger object. Upon the creation of an object, a logger is instantiated and attached to a specified file, which is formatted in a readable way. The class contains one method, getLogger, which returns the instantiated logger.

3.2.9.1 Functional Requirements

The class's only responsibility is to instantiate and return a logger.

3.3 Design Constraints

A user has to follow a specific dataset structure which limits the user from inputting a dataset in any other structure.

The input dataset in our product must be in JSON format.

3.4 Other Requirements

3.4.1 Non-functional Requirements

3.4.1.1 Usability

The product should be easy to use. The user should get the labeled instances only by uploading the data into our system.

3.4.1.2 Reliability

The product will be developed using a random labeling mechanism. So in this iteration, there is no certain reliable percentage that can be measured.

3.4.1.3 Performance

The performance of a product depends on how big the data is. The bigger the dataset, the more time the model takes. Overall the model calculation and response time should be as little as possible.

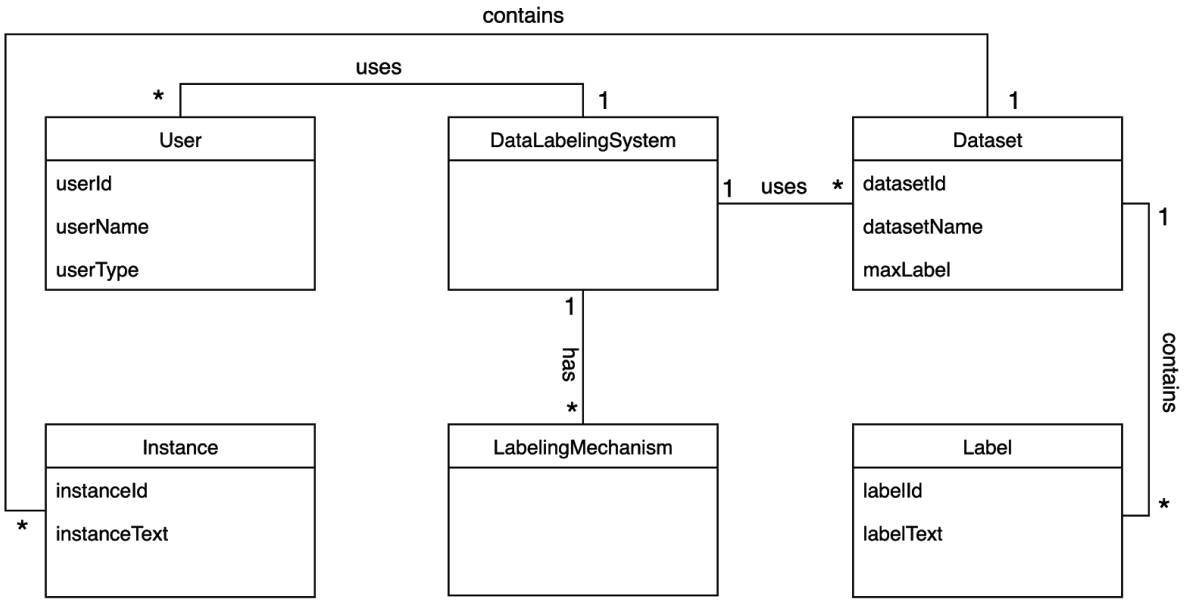
3.4.1.4 Portability

Our product works on all platforms that have terminals and runs java.

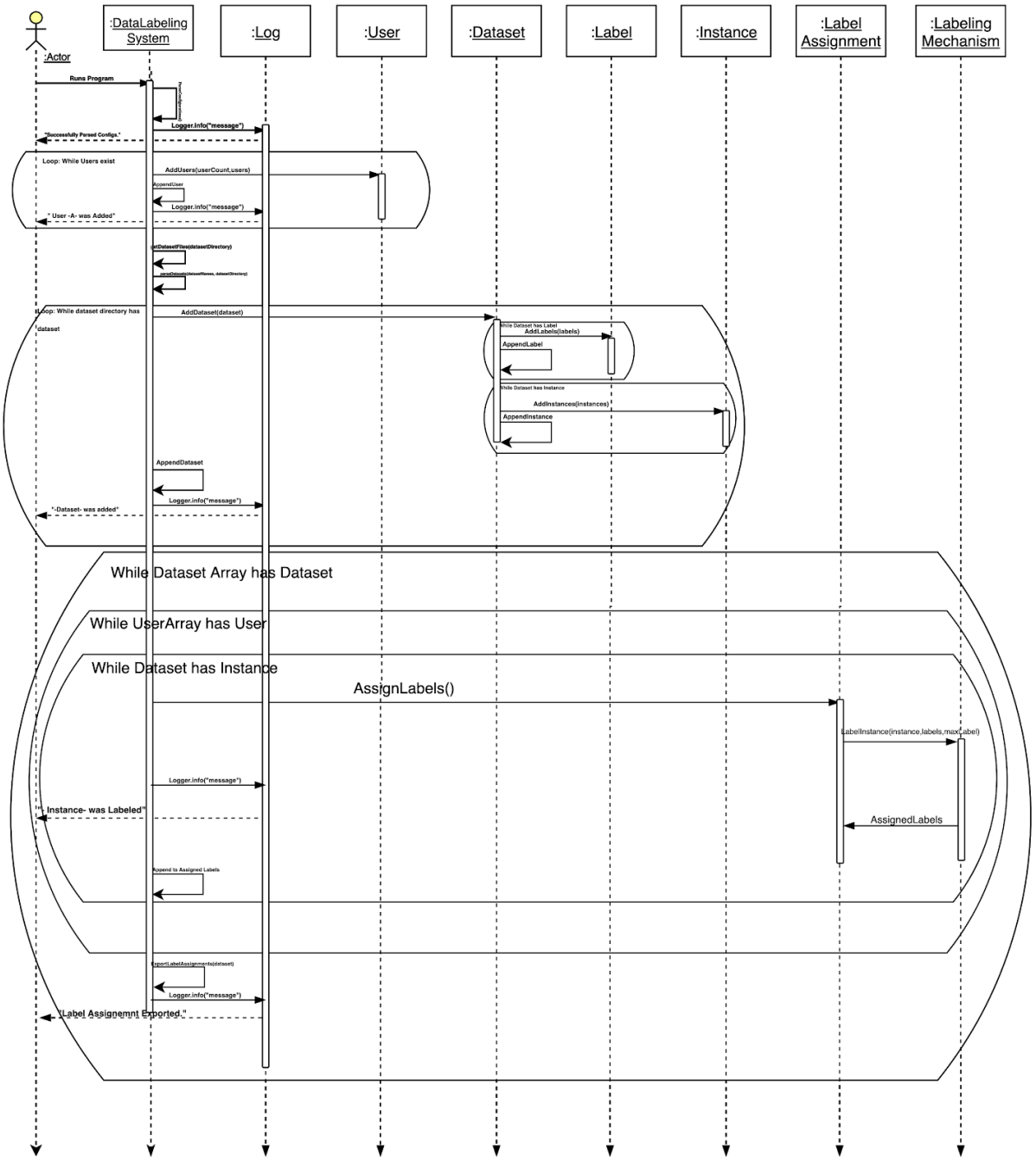
3.4.1.5 Maintainability

Our product is very flexible and can work with many kinds of labeling mechanisms and different datasets and labels.

3.5 Domain Model



3.6 System Sequence Diagram



4 Glossary

Functional Requirements: In software engineering and systems engineering, a functional requirement defines a function of a system or its component, where a function is described as a specification of behavior between outputs and inputs.

Non-functional Requirements: In systems engineering and requirements engineering, a non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions. The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture, because they are usually architecturally significant requirements.

Class: In object-oriented programming, a class is an extensible program-code-template for creating objects, providing initial values for state and implementations of behavior. In many languages, the class name is used as the name for the class, the name for the default constructor of the class, and as the type of objects generated by instantiating the class; these distinct concepts are easily conflated. When an object is created by a constructor of the class, the resulting object is called an instance of the class, and the member variables specific to the object are called instance variables, to contrast with the class variables shared across the class. In some languages, classes are only a compile-time feature, while in other languages classes are first-class citizens, and are generally themselves objects. In these languages, a class that creates classes is called a meta-class.

Function: A function is a unit of code that is often defined by its role within a greater code structure. Specifically, a function contains a unit of code that works on various inputs, many of which are variables, and produces concrete results involving changes to variable values or actual operations based on the inputs.

Method: A method in software engineering is somewhat similar to a function, except it is associated with objects/classes. Methods are very similar to functions except for two major differences.

- The method is implicitly used for an object for which it is called.
- The method is accessible to data that is contained within the class.

Instance/Object: In a computer system, any time a new context is created based on some model(its class), we say that the model has been instantiated. In practice, this instance usually has a data structure in common with other instances, but the values

stored in the instances are separate. Changing the values in one instance will then not interfere with the values of some other instance.

Attributes: Object specific properties are called, instance attributes. Think of it as the properties of instances.

Interface: In computing, an interface is a shared boundary across which two or more separate components of a computer system exchange information. The exchange can be between software, computer hardware, peripheral devices, humans, and combinations of these.

Dataset: A data set is a collection of data. In the case of tabular data, a data set corresponds to one or more database tables, where every column of a table represents a particular variable, and each row corresponds to a given record of the data set in question.

Java: An object-oriented programming language that we will use to compose the tools we use in this project.

Classifier: An algorithm used in Machine Learning areas to divide the data in two or more classes/labels.

Log: In computing, a log file is a file that records either events that occur in an operating system or other software runs, or messages between different users of a communication software. Logging is the act of keeping a log. In the simplest case, messages are written to a single log file.