IT Case Data Analyzer

Design and Planning Doc

Ray Smith, Josh Duchene, Francesca Tyler, Matthew Duff,
Nathan Lenar, Daniel Owens, Nash Wellington

***<2019-03-01>*, version 1.1**

Document Revision History

Rev. 1.0 <2019-02-28>: initial version
Rev. 1.1 <2019-03-01>: updated design details, implementation plan, and testing plan

Table of Contents

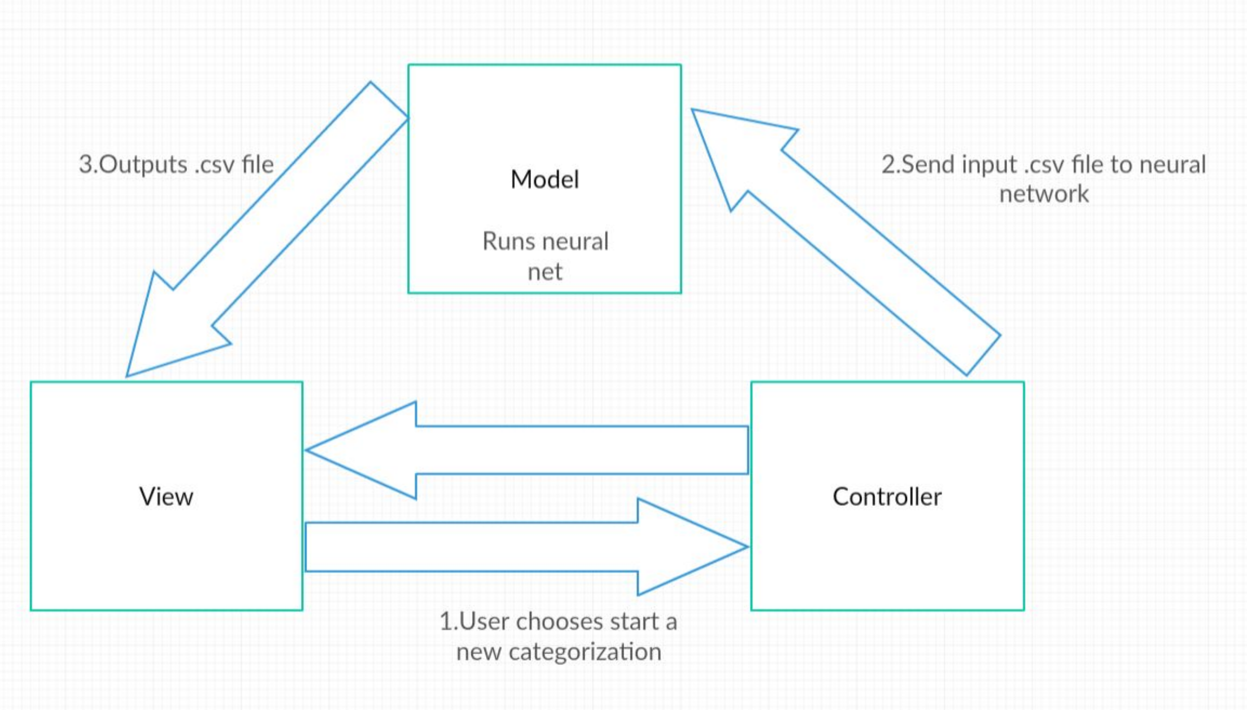
System Architecture

For this application we will be using model-view-controller for our system architecture. This is a method for separating the user interface of an application from its domain logic. This is the best architecture for us since our user interface doesn’t interact much with our domain logic other than selecting an input and displaying the output. Thus, the inherent separation of the two in this architecture is the simplest way for us to implement it.

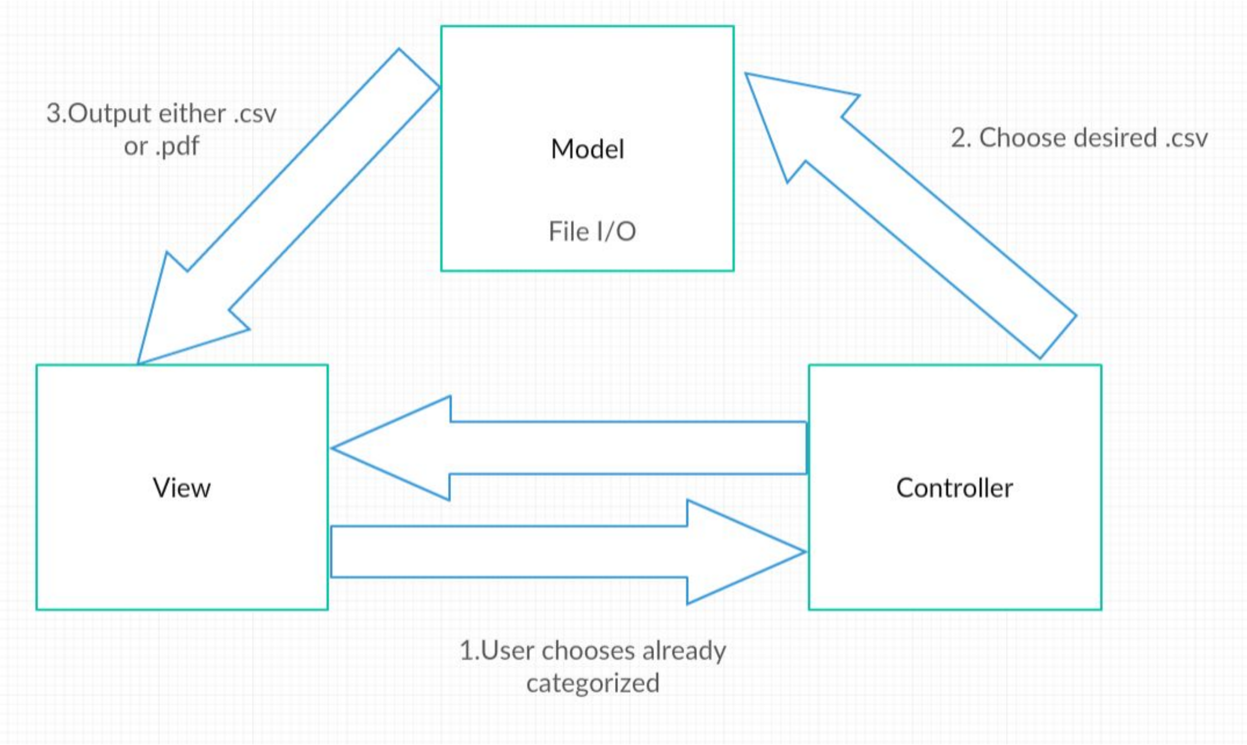
Model:
The model for us will do the primary computation needed to categorize the cases inputted. The model section will have a pipe and filter architecture of its own described in the pipe and filter section below.

View:
The view section is pretty self-explanatory but will consist of the UI where a user can select the .csv file they want to categorize as well as being able to select already categorized files. It will also display files that are already categorized and allow a user to view them in .csv and .pdf format.

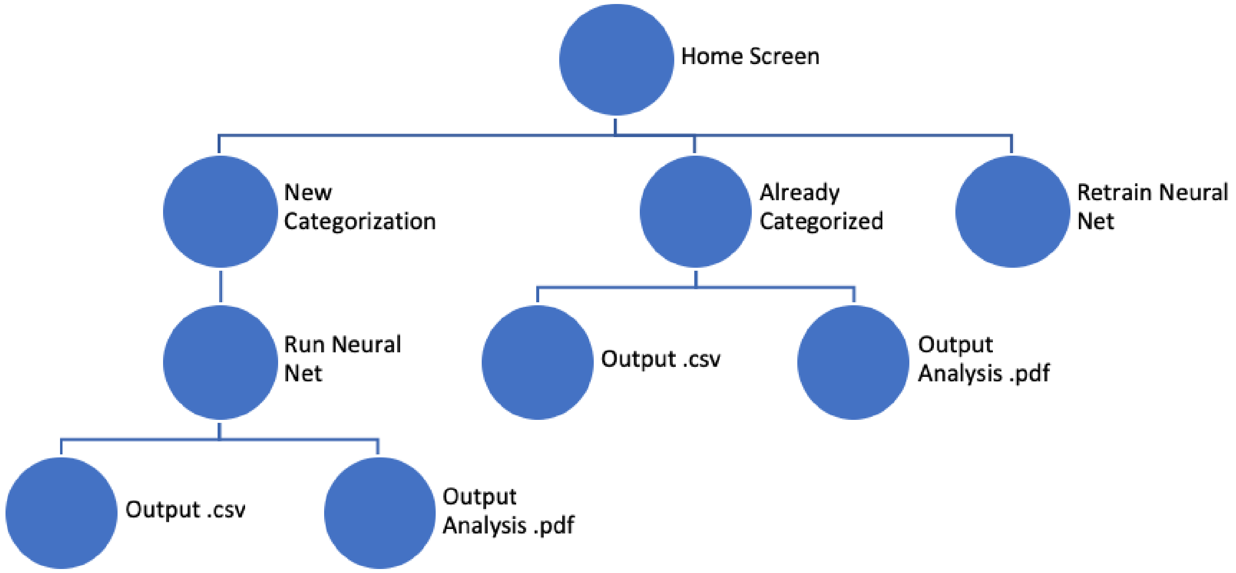
Controller:
The controller aspect of our architecture will be pretty minimal but directs inputs chosen from the view to where they need to be processed.


Model
Runs neural net
3.Outputs .csv file
2.Send input .csv file to neural network
View
Controller
1.User chooses start a new categorization

This model actually runs the neural network.


Model
File I/O
3.Output either .csv or .pdf
2. Choose desired .csv
View
Controller
1.User chooses already categorized

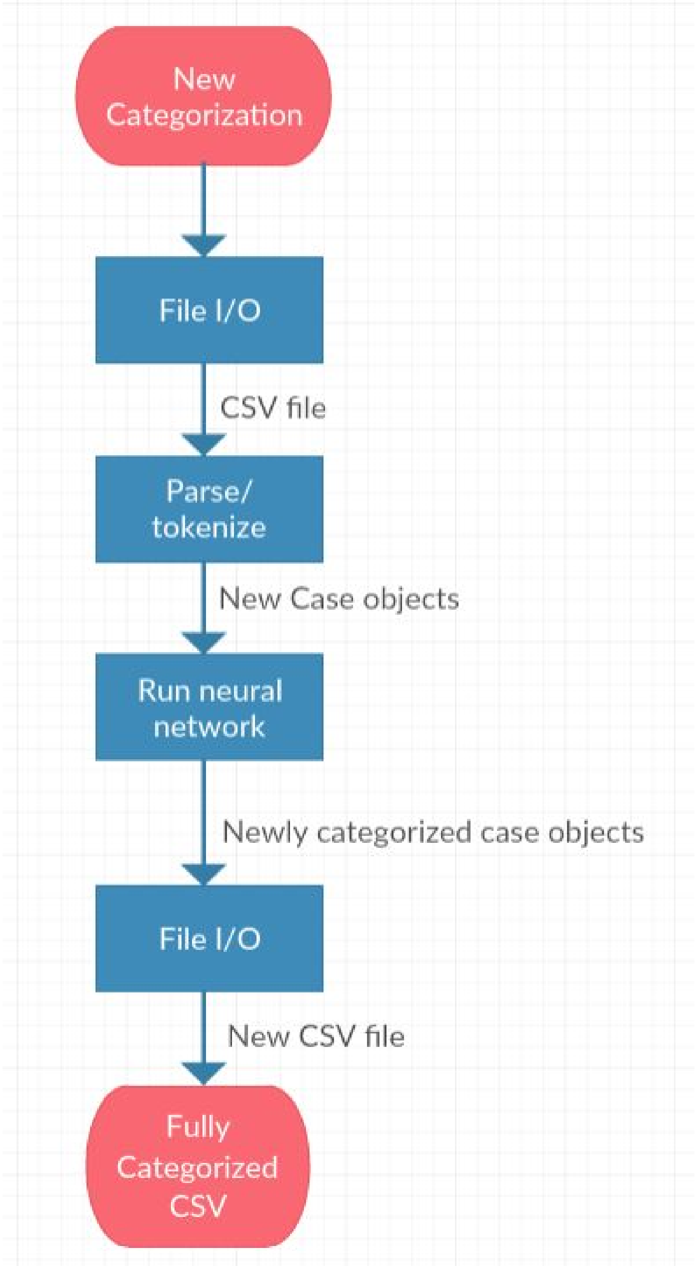
This model pulls from already categorized .csv files.



The tree diagram shows the full application layout.

Pipe and Filter:

Pipe-and-filter architecture is another way to describe our system architecture. As stated above, the user interface does not interact much with the domain logic aside from taking input and displaying output. The pipe-and-filter model, which transforms input to the output, is similar to the primary function of our program. The IT Case Data-Analyser will take a .csv file, parse/tokenize the data into case objects, put those through a neural network to categorize the data, and output a nice, organized .pdf. This progression is depicted in a diagram on the following page. Pipe-and-filter is not very good at handling interactive applications which is okay for our application because it has very minimal user-interface. The user simply needs the final output and not incremental updates of the output, so this architecture's biggest disadvantage will not greatly affect our program.



This diagram outlines the basic pipe and filter architecture we will be using.

Design Details

File In/Out:

csv_in.java, csv_out.java, pdf_out.java, categorized.java, categorized_in.java, categorized_out.java

- csv_in.java handles everything related to reading in from a .csv file generated by WiscIT.
- csv_out.java handles everything related to writing to a .csv file with the newly categorized data.
- pdf_out.java handles everything related to writing a .pdf file of a categorized data set
- categorized.java is an class mainly used as a wrapper object for holding a full data set that has been categorized. This will also handle combining categorized objects.
    - Name of Data Set
    - Date Created
    - List of Cases
- categorized_in.java reads in serialized data of already categorized data sets from our local database.
- categorized_out.java writes out serialized data of already categorized data sets to our local database.

One aspect of the program is reading in .csv files. All reading from .csv and writing to .csv files can be handled by using the Apache Commons CSV library. (http://commons.apache.org/proper/commons-csv/)

Creating .pdf files is a little more complicated but can also be done in java using the iText and PdfBox libraries. (https://www.baeldung.com/java-pdf-creation)

We will also need to save objects created to store classifications to a file, which will be done using object serialization and object input/output streams. (https://www.geeksforgeeks.org/serialization-in-java/)

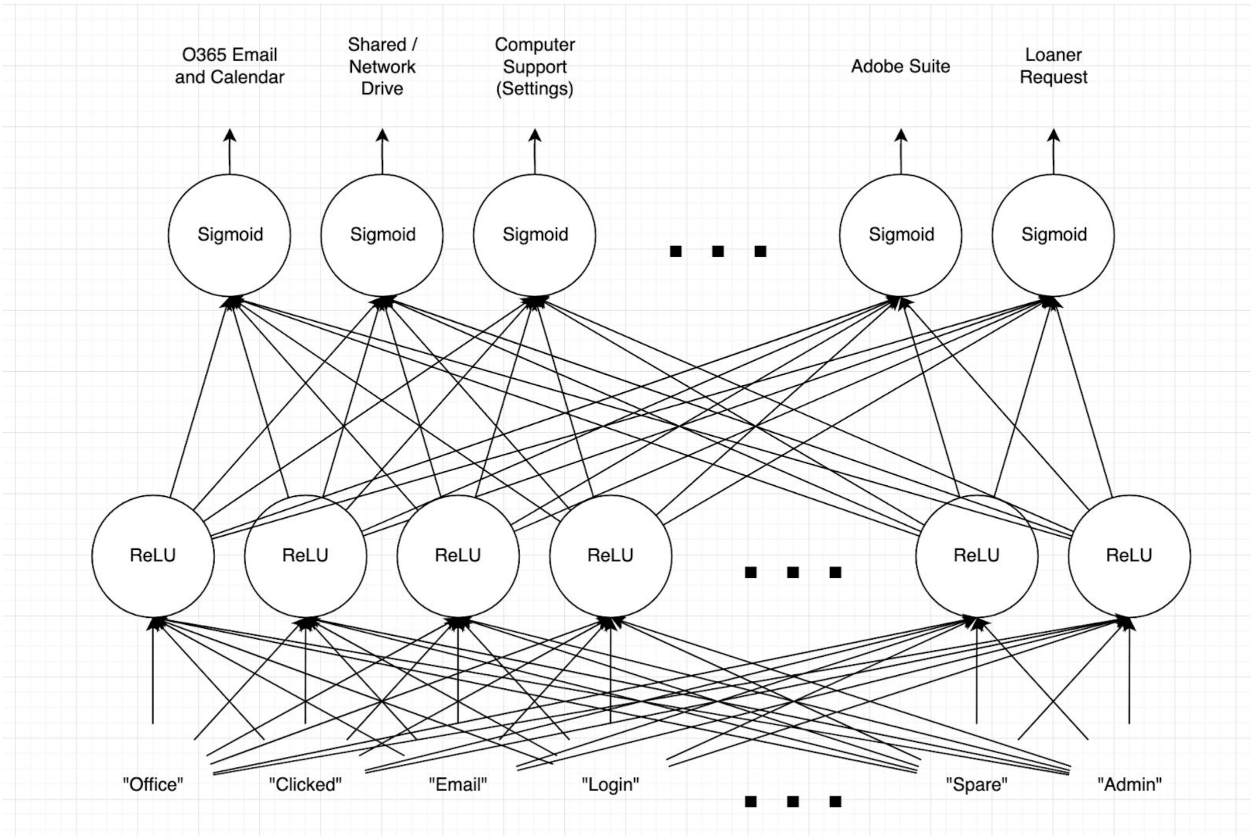
Database:

We will use a simple local database of .txt type files to store categorized objects which categorized_in.java and categorized_out.java will read from and write to. The only thing that needs to be stored really are the categorized cases for ease of customer use so that the raw data doesn't have to be run multiple times should they need to get the output file(s) again.

Neural Net:

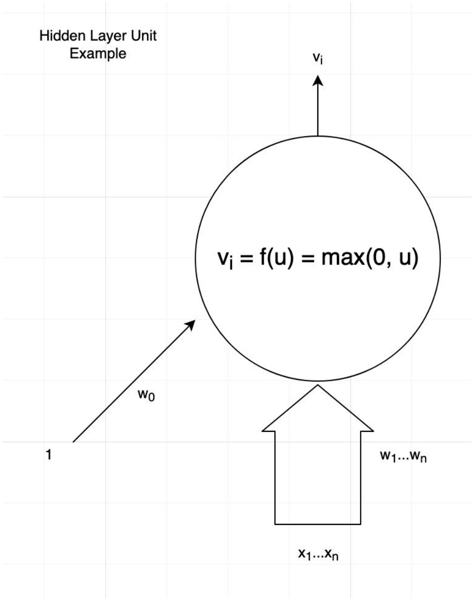
neural.java, train_neural.java, run_neural.java

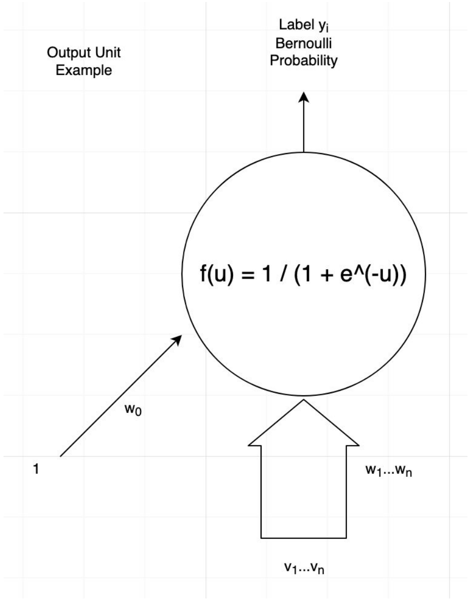
- nueral.java defines an object that is the neural network. It has all of the weights, the nodes, functions, etc. defined.
    - List of weights
    - List of nodes and their connections
    - Functions in nodes
    - Inputs
    - Outputs
- train_neural.java contains the methods needed to train the neural network and returns a weighted neural net. This will need to take in a training set and run through multiple epochs of training to set the weights.
- run_neural.java contains the methods needed to run a trained neural network and returns a categorized object. This will take in an uncategorized data set in the form of case objects tokenized by tokenization.java.



The neural network model is planned to have 20 possible labels, which will be assigned based on the trained network. For now we only plan on having 2 layers to the network, one hidden layer with many units, and an output layer with 19 units. The hidden layer units will use the Rectifier function ReLU: f(u) = max(u, 0) and the output layer units will use Sigmoid functions to approximate a step function, taking in input from all of the hidden layers. In both cases, u is the weighted input, so the sum of all x * w or v * w.

The units in the output layer (top) and hidden layer (bottom) are described in more detail below:





We will have an output for each label, and the Sigmoid function for the unit will correspond to a Bernoulli probability that the input has that label. This allows for multiple labels to cases that fall under multiple categories, but in most cases the label will be assigned to whichever has the highest probability. There is not an output layer unit for “General Question” since that will be used whenever there is not a label with a high probability.

The Inputs to the neural net are the number of occurrences of the keywords given below. (These will be narrowed down)

Outputs/Labels and their Keywords from the Data

- Office 365 Email and Calendar
    - Office, 365, O365, Email, Calendar, Outlook

- Shared/Network Drive
    - Drive, Shared, Network, SharedDrive, Mount(Maybe)
- Microsoft Office
    - Word, PowerPoint, Excel, OneNote, Outlook, Power
- Computer Support Settings
    - Monitor, “Computer Name:”, Firewall, Setup, New, Computer, Setup
- Purchase Request
    - Requested, Requesting, Request, Purchase, ALF,
- Onsite Assistance
    - Person, Onsite, Site, Physical, Location, Come
- Wisclists
    - Wisclist, List, Lists, Mass, Email
- Audio/Visual Support
    - Realtime, Real, Time, Audio, HDMI, Webchat, Web Chat, Voicemail, Google Hangouts, Camera, Projector
- Device Repair
    - Repair, Broken, Imaged, Reimaged, Issues, Computer, Wiped, Formatted, Format, Re-Imaged, Fix, Fixed
- Service Account Creation (Email)
    - Email, Role, Based, New, Employee, Staff, Member, Setup, Service, Account, Creation
- Admin Account/Password
    - Admin, Account, Password, Login, Log, In, Access, Administrative, Administrator, Credentials, Permissions, Permission
- Login Issues
    - Log, In, Login, Password, Username, Incorrect, Wrong, Unable
- Virus/Malware
    - Click, Link, Clicked, Phishing, Email, Infected, Virus, Malware, Scam
- Printer Support
    - Printer, Printers, Printing, Duplex, Scan, Fax, Print
- Multi Factor Authentication
    - Duo, Multi, Factor, Authentication, MFA, Multi-Factor, Token, Fob, OTP, Time, Password
- Room Access
    - Room, Floorplan, Floor Plan, Request, Requesting, Access, Office, Classroom, (Room Number)
- Network Connectivity
    - VPN, Wifi, Internet, Ethernet, LAN, WLAN, Outage, Outages, Server

- Adobe Suite
    - Adobe, Photoshop, Indesign, Illustrator, Premiere Pro, Premiere, Acrobat, PDF, Creative, Cloud, Lightroom, Bridge, Audition, Dreamweaver, Licence, Suite
- Loaner Request (Computer/Mifi)
    - Verizon, Mifi, Rental, Loaner, Rent, Spare, Borrow, Borrowed, Request, Day, Week
- General Question
    - No Keywords, anything not close to another category goes here

We will also need functions to train the neural net taking in a training set, and then to categorize a new data set.

At one point, we were considering a different model for the Neural Network, where instead of many output units, we had a single with a more complicated step function with 20 steps, rather than numerous 1 step functions. This would have simplified the network design as a whole, but would have required careful ordering of the labels, and would have likely led to less accurate categorizations, which is why we chose this model in the end.

Parsing/Tokenization:
tokenization.java, case.java

- tokenization.java contains all the methods needed to tokenize a description string for a case, described below.
- case.java is a class that will extend Serializable and is used to define an individual case object, which will be held in the list of a categorized object.
    - Case Number
    - Case Owner
    - Case Requestor
    - Date Requested
    - Date Completed
    - Description
    - Tokenized Description
    - Neural Net Input Variables/Features based on the Tokenized Description
    - Category/Label(s)

The tokenized data as well as the other parsed sections will be saved as a “case” object.

- Tokenization of Description Section
    - Segmentation - separate words, remove punctuation -> Array/List of tokens
    - Casing - make everything lowercase
- Relevant Data Sections to Save
    - Case Number
    - Requestor
    - Description
    - Owner
    - Date
    - Call Source

User Interface:

window_main.java, window_new.java, window_categorized.java

- window_main.java will extend JFrame and define the main menu window of the application, containing all necessary buttons, text fields, etc. It will call methods to access the other 2 user interfaces.
    - window_new.java
    - window_categorized.java
- window_new.java will extend JFrame and define the menu for categorizing a new data set. It will call functions from and use:
    - csv_in.java
    - neural.java
    - run_neural.java
    - csv_out.java
    - pdf_out.java
    - categorized.java
    - categorized_out.java
    - case.java
- Window_categorized.java will extend JFrame and define the menu for viewing already categorized data sets. It will call functions from:
    - categorized.java
    - csv_out.java
    - Pdf_out.java
    - categorized_in.java
    - categorized_out.java
    - case.java

Using JFrames, we can easily create buttons and text fields to allow the user to interact with the backend functions described previously. Netbeans makes creating interfaces easy with drag and drop options, which will save time and allow more effort to be put into the network, which will take the most time.

Implementation Plan

Iteration 1:

In iteration 1, we will implement all of the code necessary to categorize a data set. Broadly, this means we will need to be able to train a neural network, input the .csv data sets that need to be categorized, clean this data, and run the neural network on the cleaned data to categorize it.

User Stories to be Implemented:

- Train Neural Network
- Categorize New Data Set of Cases

Most of the classes being implemented in this iteration rely on the case and categorized classes, so these will be our first priorities. Then, we will proceed along two paths: 1) processing the .csv input file and 2) training the neural network. For processing the .csv file, we first need implement the tokenization class, as csv_in will rely on this. For training the neural network, we first need to implement the neural class, as this will be used by run_neural and train_neural classes. Then, we'll implement the run_neural class, as this will be used by the train_neural class.

Unit testing will be built by the individual developers responsible for their module, and tests run as they are built until all are passed. The developer will be expected to run these tests every time they modify their code.

Classes to be implemented:

- case.java
- categorized.java
- tokenization.java
- csv_in.java
- neural.java
- run_neural.java
- train_neural.java

<table>
<tr><td>Name</td><td>Train Neural Network</td></tr>
<tr><td>Programming Tasks</td><td>
<table>
<tr><th>Task</th><th>Difficulty</th><th>Assigned To</th></tr>
<tr><td>case.java</td><td>1</td><td>Nate</td></tr>
<tr><td>categorization.java</td><td>1</td><td>Nate</td></tr>
<tr><td>neural.java</td><td>3.5</td><td>Dan, Nash</td></tr>
<tr><td>train_neural.java</td><td>4.5</td><td>Ray, Josh</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>csv_in.java, case.java, categorized.java, tokenization.java</td></tr>
<tr><td>Iteration</td><td>1</td></tr>
</table>

<table>
<tr><td>Name</td><td>Categorize New Data Set of Cases</td></tr>
<tr><td>Programming tasks</td><td>
<table>
<tr><th>Task</th><th>Difficulty</th><th>Assigned To</th></tr>
<tr><td>run_neural.java</td><td>2.5</td><td>Ray, Josh, Nash, Dan</td></tr>
<tr><td>csv_in.java</td><td>2.5</td><td>Matt</td></tr>
<tr><td>tokenization.java</td><td>2.5</td><td>Fran</td></tr>
<tr><td>Learn TestNG</td><td>3</td><td>Matt</td></tr>
</table>
</td></tr>
<tr><td>Dependencies</td><td>Train Neural Network</td></tr>
<tr><td>Iteration</td><td>1</td></tr>
</table>

Iteration 2:

In iteration 2, we will implement a GUI that allows the user to interact with the program. This will include features for outputting the categorized data sets and interacting with previously categorized data sets.

User Stories to be Implemented:

- Display Categorized Data Sets
- Merge Datasets
- Output .csv File of Categorized Data Set
- Output .pdf File of Simple Metrics

The back end classes will likely be implemented before the GUI classes are, though we could at least get the layout of the GUI classes figured out while the back end is being implemented. For the back end, we will probably implement the categorized_in and categorized_out classes first, as this will help with implementing the GUI. Then we'll implement the csv_out and pdf_out classes. For the GUI, we'll first implement the window_main class, as this won't rely on the back end classes being implemented. The window_new class will be implemented next, as most of the I/O used by this window will be implemented in iteration 1. Finally, we'll implement the window_categorized class.

Unit testing will be built by the individual developers responsible for their module, and tests run as they are built until all are passed. The developer will be expected to run these tests every time they modify their code.

Classes to be implemented:

- csv_out.java
- pdf_out.java
- categorized_in.java
- categorized_out.java
- window_main.java
- window_new.java
- Window_categorized.java

<table>
<tr><td>Name</td><td>Display Categorized Data Sets</td></tr>
<tr><td>Programming Tasks</td><td><table>
<tr><th>Task</th><th>Difficulty</th><th>Assigned To</th></tr>
<tr><td>window_main.java</td><td>1.5</td><td>TBD</td></tr>
<tr><td>window_categorized.java</td><td>2</td><td>TBD</td></tr>
<tr><td>window_new.java</td><td>2</td><td>TBD</td></tr>
<tr><td>categorization_in.java</td><td>1.5</td><td>TBD</td></tr>
<tr><td>categorization_out.java</td><td>1.5</td><td>TBD</td></tr>
</table></td></tr>
<tr><td>Dependencies</td><td>categorized.java</td></tr>
<tr><td>Iteration</td><td>2</td></tr>
</table>

<table>
<tr><td>Name</td><td>Output .csv File of Categorized Data Set</td></tr>
<tr><td>Programming Tasks</td><td><table>
<tr><th>Task</th><th>Difficulty</th><th>Assigned To</th></tr>
<tr><td>csv_out.java</td><td>3.5</td><td>TBD</td></tr>
</table></td></tr>
<tr><td>Dependencies</td><td>None, as long as we know what format categorized data sets will be stored in. We could use mock data sets to implement and test this</td></tr>
<tr><td>Iteration</td><td>2</td></tr>
</table>

<table>
<tr><td>Name</td><td>Output .pdf File of Simple Metrics</td></tr>
<tr><td>Programming Tasks</td><td><table><tr><th>Task</th><th>Difficulty</th><th>Assigned To</th></tr><tr><td>pdf_out.java</td><td>5</td><td>TBD</td></tr></table></td></tr>
<tr><td>Dependencies</td><td>None, as long as we know what format categorized data sets will be stored in. We could use mock data sets to implement and test this</td></tr>
<tr><td>Iteration</td><td>2</td></tr>
</table>

<table>
<tr><td>Name</td><td>Merge Datasets</td></tr>
<tr><td>Programming Tasks</td><td><table><tr><th>Task</th><th>Difficulty</th><th>Assigned To</th></tr><tr><td>Additional method in categorization.java</td><td>2</td><td>TBD</td></tr></table></td></tr>
<tr><td>Dependencies</td><td>None, as long as we know what format categorized data sets will be stored in. We could use mock data sets to implement and test this</td></tr>
<tr><td>Iteration</td><td>2</td></tr>
</table>

Testing Plan

Primer:
Our team’s goal is to design the project using a Test Driven Development (TDD) approach. Using the TDD approach will give us the primary benefit of having code built in such a way that it can be tested easily and automatically.

Unit Testing:
Unit testing will be built by the individual developers responsible for their module. The tests will be implemented using TestNG, an open source software built on top of the JUnit testing interface. We will be implementing it using the Triple A approach to unit testing (Arrange, Act, and Assert). This will be implemented using TDD principles where we establish the tests and then build code that passes those tests, so the Unit Testing phase will be using white box testing principles.

Integration Testing:
Integration testing will occur as interrelated units complete. As such, we’ll expect a bottom-up integration scheme where smaller units perform integration tests between each other to form larger, tester subsections. Eventually it will yield a fully fledged program. The biggest difficulty will come from designing driver classes which act to combine the modules until the larger module can actually be implemented.

We will be implementing integration testing using a program called Spock for Java, which is described as an enterprise-ready testing and specification framework. This framework relies heavily on gradle, maven, or ant based builds and as such we’ll have to format our project around these aspects. These tests will be built as we try to integrate the pieces together and should be run everytime we change the code of any submodules.

System Testing:
Once every unit has been integrated we will start testing the system. We will implement this using FitNesse, a testing interface that allows for a table of inputs and expected outputs. This yields a list of successes and failures. This will 1) Determine how well trained our neural network is and 2) give a margin of error for our client, a request he did have for us during our client meeting.

For our system testing phase we will be categorizing a selection of entries, placing them into an input table with our expected categories, and from there determining the percentage of current placement our software gets.

Acceptance Testing:

After we have completed system testing we can finish with acceptance testing. Acceptance testing is similar to system testing, except the input data used will be pulled from the actual Cherwell Client. From here we can manually check random clusters to guarantee our software works with new data.

Regression Testing:

Regression testing will be an ongoing process in which each group member performs a combination of unit testing and integration testing as they add/change features within a unit. After functionality has been added, the developer will use unit tests to make sure that the code passes the tests designed beforehand. Then, they will test that the cross component functionality hasn't been hindered using the aforementioned integration testing techniques.

Performance Testing:

We do not foresee performance testing being a necessary part of our testing cycle. The training of the network will be the most time intensive portion of our program, as we expect it to take O(M*N) time, where M is the number of weights, and N is the number of entries. This is not concerning, though, as it's us, the developers, that have to run this functionality. The user will probably never see or use this part of the program.

For the user we're expecting O(N) time, where N is the number of entries they're hoping to have sorted. With a trained network it should be constant time to categorize each individual entry.

Beta Testing:

As soon as we complete system testing and basic acceptance tests, we plan to move forward with Beta test for our client. This will allow us to get valuable feedback not only on its ability to classify cases but also on our User interface look and feel and our outputted PDF. This is likely to be one of our most valuable forms of testing because it gives us insight on how our users interact with our software and what can be done to cater to that. This testing will most likely come near the end of our production cycle, as most things will need to be done. Planned beta test come at the end of iteration 1 and the end of iteration 2.

Bug Tracking:
Tracking bugs will be done using github’s built in bug tracking (Issues Tab) where we can all share our individual issues and everyone can be informed about them/look into fixing them. As we post “Bugs” and issues, github’s interface allows us to track them and close them when they are fixed. This feature allows us to point exactly to where in ourcode we think the issue is occuring.

This tool can also be used to track “To Do” items that aren’t done yet, so other team members know a certain aspect of the code isn’t done yet, and to point out areas of concern in our other team members’ code using “Invalid” or “Question” tags.