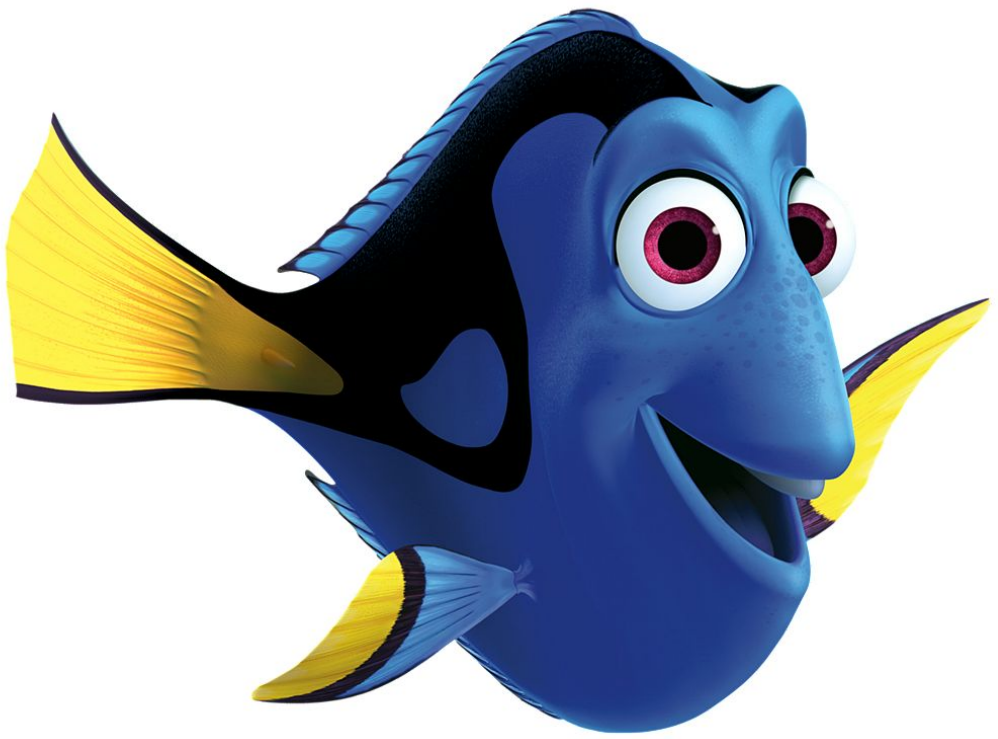
Never-Ending Medical Operative (NEMO)



Unit Test Cases and Plan
Version: 1.0

Authors:
Cody Moffitt
Jackson Chandler
Andrew Pachuilo
Oliver Payne

Table of Contents

Chapter 1: Introduction

1.1: System Overview

The Never-Ending Medical Learner (NEMO) is being developed for the Kansas University Medical Center as a tool for machine learning research. The primary purpose of the system is to aid domain experts in developing predictions relating to patients. The system uses machine learning techniques in order to make conclusions and predictions from prior data. The system shall operate by taking questions from the domain experts and creating learning algorithms to generate predictions. The system shall make use of a feedback mechanism, where users must give responses to the learners based on the prediction models they develop.

The system's structure is generalized in order to allow the addition of other machine learning techniques. However, the basic framework for the system has been constructed. The NEMO system shall be comprised of a data mart and three major components: the AI controller, the data loader, and the web application. The AI controller will be used to create and manage learning algorithms in response to questions and feedback given by domain experts. The data loader will be responsible for loading test data and learner data into the data mart from KUMC's external database. The data mart will hold all of the test and learner data, as well as all of the users' questions and feedback. Finally, the web application will be the domain experts' interface with the NEMO system, where they will be able to create questions and submit feedback.

The purpose of this document is to provide unit tests to verify the functionality of the system. The testing environment and individual unit tests for requirements are detailed in the sections below.

1.2: Test Environment

Certain components of the system are assumed to have access to a data mart that is up and running along with a web server that is up and running. The data mart will have a TestUser account that can be loaded with dummy questions as needed along with other dummy users that can be loaded with dummy questions. Classifier and classifier optimizer test data is included in a zip file with this document for tests 2.19 through 2.24. There are files that will be used as learner data or test data. There are also results to verify against tests against. The data was omitted from this document for readability.

The web client framework and data mart creation script can be worked on concurrently. Once a script for correctly creating the data mart is made, functionality of the web client can be worked on with use of dummy data being loaded into the data mart. The AI controller framework can be worked on by using dummy questions to ensure the controller is spawning AI processes. While functionality of the AI controller and web client are being worked the data loader can also be worked on. Once the data loader is able to load the data mart with valid i2b2 data, the AI controller can be worked on ensure it produces valid results. During this period the web client can be further tested with actual data to ensure that it runs correctly. The system can be tested in its entirety when the AI controller is fully tested and functional.

1.3: Terminology

AI: Artificial intelligence. It is the process of simulating one or more human characteristics with a computer. In this particular case, the artificial intelligence is machine learning.
AI Model: This is a trained classifier.
BLOB: Binary Large Object. A datatype in SQL. It is used in the NEMO data mart to contain serialized AI models.
C4.5 Decision Tree: A classifier algorithm that uses a decision tree.
Classifier: A classifier is an algorithm used by AI models to train on data sets and make predictions.
Database: A collection of a large amount of data. This data is organized into entries of records. Each record is made up of several components.
Data Mart: A repository for the components of the NEMO system. This database will contain a smaller subset of the medical database, which will be partitioned into test data and learner data for the AI learners to use. The database will also contain the questions posed to the learners, along with any other information associated with them. Finally, the data mart will contain saved instances of the learning algorithms themselves.
Dummy AI Model: An arbitrary AI model. A trained classifier model stored as a BLOB in the data mart and used by the AI manager to respawn trained AI agents. A dummy AI model is used for testing.
Dummy Question: A question created in NEMO with arbitrary narrowing attributes and parameters. It is basically a test question. The dummy question to be loaded for us in testing will be “How many patients with diabetes are readmitted within 30 days?”.
Ensemble Learning: A machine learning technique in which multiple learners are trained to solve the same problem.
Global Dashboard: This is the area of the web client where users can view other users’ questions.
Hard Edit: An edit of a question that changes the original question in the database without creating a new one.
i2b2: A National Center for Biomedical Computing which developed a biomedical informatics framework funded by the U.S. National Institutes of Health.
KUMC: Kansas University Medical Center.
Learner: An algorithm that is purposed with evaluating data and forming conclusions based off of it. A machine learning algorithm.
Naive Bayes: A classifier algorithm utilizing Bayes Theorem.
NEMO: Never-Ending Medical Operative. The project is based off of the Never-Ending Language Learner (NELL) system. The goal is to create a system that continually learns on real-world data in order to make predictions and answer medical related questions.
Neural Network: A classifier algorithm that utilizes a neural network inspired by biological neural networks.
Random Forest: A classifier algorithm that utilizes random feature selection.

Soft Edit: An edit of a question that creates a new question in the database.
SQL: Structured Query Language. The data mart will be a SQL server, which is a type of database server.
SQL Query: A request for a specific data set to be sent to a SQL server
Support Vector Machine: A specific classifier algorithm that utilizes vectors and some complex geometry.
User Dashboard: This is the area of the web client where users can view only their own questions.

Chapter 2: Unit Test Cases

2.1: User logs into NEMO web app

Author: Cody Moffitt
User Story Covered: 2.01
Test Description:
This test will verify that a user is able to login to the NEMO web app.
Preconditions:
- The tester has database credentials
- The test user has been created and registered in the data mart

Input:
- Username: TestUser
- Password: TestPass
- URL for NEMO webapp
- Various button clicks as specified in the Test Steps

Post Conditions:
- A user session is in the data mart for TestUser

Test Steps:
- Enter the URL for the NEMO webapp
- Click Login, and a popup will be displayed
- Enter “TestUser” into the User Name field
- Enter “TestPass” into the Password field
- Login the the SQL database
- Run the query: Select * from User u Inner Join UserSession us on u.ID = us.UserID Where u.Name = ‘TestUser’
- The query should return one result

Expected Results:
The query should return one row, describing the test user and the test’s user’s session. Session login time should match the time you logged in (corrected for UTC time).

2.2: User poses question via web application

Author: Jackson Chandler
User Story Covered: 2.02
Test Description:

This test will ensure that a question exists for a certain user. For this test the data mart must be running and a user needs to have submitted a question. The data mart will be checked before and after each to make sure the question exists.

Preconditions:

- A user is logged in.
- A user has not exceeded the max number of questions

Input:

- A question to ask.
- Clicking submit button.

Postconditions:

- A question is added to the queue to be answered in the data mart

Test Steps:

- A User inputs a question and clicks submit.
- The SQL query for the user is check to see if they are at the max number of questions.
- If they are the user is notified that they are at their max
- If not then the question is added to the table.
- Check the SQL query for new questions by a user.
- When new the new questions is found the user will be notified.

Expected Results:

If a user has not submitted any previous questions, then a blank table is expected. Otherwise, a list of questions is noted. The user will then be notified that the submission was a success.

2.3: User poses question that already exists and chooses to duplicate it using pre existing AI models

Author: Andrew Pachuilo
User Story Covered: 2.3
Test Description:

This test will ensure that the system can recognize that a duplicate question is being asked and allow them to use pre-existing AI models for processing their question.

Preconditions:

- Tester has not exceeded their maximum number of questions.
- A dummy question exists within the data mart.
- The dummy question has dummy AI models.

Inputs:

- No specific input. User button clicks.

Post Conditions:

- Newly posed question is in the data mart with chosen dummy AI model.

Test Steps:

- Dummy question is formed via dropdown menus
- Submit question button is clicked.
- Choose yes when asked to use pre-existing AI models.
- Choose and make note of the chosen dummy AI model.
- Issue SQL query to receive both the dummy question and its newly formed duplicate.
- Review the AI model for the newly formed duplicate question.

Expected Results:

The SQL query should show that a question that is a duplicate of the dummy question with a recent timestamp. The AI model for the newly formed duplicate question should be the model that was chosen during creation of the question.

2.4: User deletes question

Author: Oliver Payne
User Story Covered: 2.4
Test Description:

This test will verify that the deletion of a question is working properly. In order to test this, there must be a question in the database for the system to delete. To test this, a dummy question will be submitted and then deleted. At each point during the operation, the question table will reflect the entries of the data mart. Because of this, the data mart can be checked to see if the operations (submitting, deleting) were successful.

Preconditions:

- N/A

Input:

- No specific input. User button clicks.

Postconditions:

- The question has been deleted from the data mart

Test Steps:

- Log into the web client and navigate to the question creation page.
- Using the drop-down menus, construct the question “How many patients with diabetes were readmitted within 30 days?”. Submit this question to the database.
- Check the question table to see that the question has been added (the question table will query the data mart for entries)
- Delete the entry from the database
- Check the question table again to see if the deletion was successful (again, the question table will query the data mart to reflect its entries)

Expected Results:

An entry to the question table should appear when the dummy question is submitted. When the user deletes a question from the web client, the data mart is instructed to remove the corresponding entry from its data. Once the deletion has been performed, the question will not exist in the database. The question table on the web client side will reflect this, and the previous entry will be removed.

2.5: User does a hard edit of question

Author: Andrew Pachuilo
User Story Covered: 2.05
Test Description:
This test will verify that the system edits a question correctly.
Preconditions:
- Tester has a dummy question on their account.

Inputs:
- No specific input. User button clicks.

Post Conditions:
- Edited question is in the data mart

Test Steps:
- Run a SQL query of test user account's questions and log results.
- Dummy question is chosen by the tester. Taking note of the question's current parameters.
- The question is edited, via dropdown menus, to be "How many patients with lupus are readmitted within 30 days?". Taking note of changes made to the question.
- The submit question button is clicked.
- 'No' is clicked when asked to pose as a new question.
- Run a SQL query of test user account's questions and log results.

Expected Results:
The two SQL queries should show that the dummy question has been correctly edited with the chosen parameters.

2.6: User does a soft edit of a question

Author: Cody Moffitt
User Story Covered: 2.06
Test Description:
This test will verify that the system duplicates an edited question correctly.
Preconditions:
- The tester is logged in with user name “TestUser” and password “TestPass”
- The test user has only one question on their account

Input:
- No specific input. User clicks GUI elements as specified in Test Steps

Post Conditions:
- The test user now has two different questions on their account

Test Steps:
- Login with the test user credentials
- Navigate to the Question page
- Verify that the test user only has one question for their account
- Select edit on the question
- In the edit dialogue, delete the narrowing attributes
- Add a different attribute with arbitrary parameters
- Click “Submit”
- Choose to save it as a new question when given the option
- Navigate back to the Question page
- Verify that the user now has two questions on their account
- Verify the original question has the same attributes and parameters it had before
- Verify the new question has the attribute and parameters specified

Expected Results:
The test user will now have two questions on their account. The original question, the same as it was before the edit, and the edited question.

2.7 User gives feedback to AI learners

Author: Oliver Payne
User Story Covered: 2.7
Test Description:

This test will verify that the user is able to submit feedback for a question in the database. First, the system must detect that a question is ready to receive feedback. Next, the system must offer the user the ability to submit feedback for the question via the web client. The web client must then receive the feedback from the user and deliver this to the database. Once this is done, the question must be handled according to the user's feedback.

Preconditions:

- N/A

Input:

- No input. The user clicks GUI elements as specified in the test steps

Postconditions:

- The feedback from the user is saved in the data mart and will acted on accordingly by the AI controller

Test Steps:

- Log into the web client and navigate to the question creation page.
- Using the drop-down menus, construct the question "How many patients with diabetes were readmitted within 30 days?". Submit this question to the database.
- Wait for the question to be run on the AI learners.
- Check on the question page to see if the status of the question is "Awaiting User Feedback"
- Select "Give Feedback". When prompted "Are you satisfied with the performance of the AI?", select "No" and when prompted "Do you agree with this prediction?", select "No", and finally, when prompted "What would you like to do with this learner?", select "Reset Learner". Submit the feedback.
- Check on the question page to see if the status of the question is "Processing User Feedback".
- Check the question page later to see if the status of the question is "Running on Algorithm"

Expected Results:

When a learning algorithm detects that it is finished running on a specific question, it waits for the user to deliver feedback. The status of the question is updated to display "Awaiting User Feedback". The user will then deliver feedback to the learner. Once the feedback has been submitted, the status of the question will display "Processing Feedback". Once the AI controller is able to get back to the question, it will handle the question according to the feedback given. With the feedback described above, the question will be rerun on a new learning algorithm. During this time, the status of the question will read "Running On Algorithm". As always, the question page on the web client will reflect the status of each question held in the data mart.

2.8: User searches global dashboard for question

Author: Jackson Chandler
User Story Covered: 2.08
Test Description:

This test will show that the user can search the global dashboard for questions. The data mart must be running and there must also be previous questions in the data mart to display. Dummy questions will be implanted into the data mart to ensure that there are questions.

Preconditions:

- User is logged in.
- Dummy questions must exist in the data mart

Input:

- A question to be selected on the global dashboard to search

Postconditions:

- A list of similar questions to the searched question will be displayed

Test Steps:

- User select questions and additional arguments with drop down boxes on the global dashboard
- The Search button is then clicked.
- The data mart is checked to see the number of questions for the search.
- The display is checked to ensure that the number of displayed is the same as the data mart's.

Expected Results:

A set of dummy questions should appear in the table before the search. When the user has searched for a question, all similar questions are displayed.

2.9: User sorts the questions on the dashboard

Author: Cody Moffitt
User Story Covered: 2.09
Test Description:
This test will verify that the questions on the dashboard are sorted correctly
Preconditions:
- The tester is logged in to the web app with user name “TestUser” and password “TestPass”
- Three or more questions exist in the data mart

Input:
- No specific input. User button clicks

Post Conditions:
- The questions are sorted alphabetically or reverse alphabetically in the dashboard

Test Steps:
- Login with the test user credentials
- Navigate to the Global Dashboard page
- Click the up arrow on the first column header
- Verify that the list of questions is now sorted in alphabetical order based on that column
- Click the down arrow on the first column header
- Verify that the list of questions is now sorted in reverse alphabetical order based on that column
- Repeat this sorting process for each column, up and down, verifying the results

Expected Results:
Each column will be sorted alphabetically upon having its header’s up arrow clicked, and reverse alphabetically upon having its header’s down arrow clicked.

2.10: User duplicates a question from global dashboard without using existing AI models

Authors: Andrew Pachuilo
User Story Covered: 2.10
Test Description:

This test will verify that the user can duplicate questions asked by other users with their associated AI models.

Preconditions:

- A dummy user has a dummy question in data mart.
- Dummy question has dummy AI model.
- Test user has not exceeded their maximum number of questions.

Inputs:

- Clicking another user's question.
- Clicking duplicate question.

Post Condition(s):

- Newly formed duplicate question exists within the data mart under test user's account with chosen dummy AI model.

Test Steps:

- Run a SQL query of test user account's questions and log results.
- Click another user's dummy question.
- Click the 'duplicate' button.
- Click 'Yes' to using a pre-existing AI model.
- Click the dummy AI model to choose it.
- Run a SQL query of test user account's questions and log results.

Expected Results:

The two SQL queries should show that the dummy question has been correctly duplicated with the chosen AI model under the test user's account.

2.11: User asks similar question from global dashboard

Author: Oliver Payne
User Story Covered: 2.10
Test Description:

This test will verify that the user is able to submit a question that is similar to a question that has already been asked. In order to do this, one user must ask a particular question first. After this, the second user must select this question from the global dashboard and choose “Ask Similar Question”. Next, the system must allow the user to make changes to the question. Once the question is submitted, it should then appear in the global dashboard, in addition to the original question posed by the first user.

Preconditions:

- N/A

Input:

- No input. The user clicks on GUI components as described in the test steps

Postconditions:

- Two similar questions posed by two different users are in the database

Test Steps:

- Log into the web client and navigate to the question creation page.
- Using the drop-down menus, construct the question “How many patients with diabetes were readmitted within 30 days?”. Submit this question to the database.
- Logout of the web client and log back in under a different user.
- Navigate to the global dashboard page. Select the first question that was asked by a different user and choose “Ask Similar Question”
- When redirected to the question creation page, simply select submit
- Navigate back to the global dashboard. Locate the two identical questions and check their “User” columns to verify that they are both different.

Expected Results:

When a question is submitted, the question will appear on the global dashboard page. A different user that navigates to the global dashboard selects the original question and chooses “Ask Similar Question”. This takes the user to the question creation page with the entries of the drop-down menu set to match the original question. When the user selects submit, the question will be submitted to the data mart. When the user navigates back to the global dashboard, two entries of the same question will appear. One entry will list the first user under its “User” column. The other entry will display the second user, the one that asked the similar question.

2.12: Admin tells data-loader to force fetch data

Author: Jackson Chandler
User Story Covered: 2.13
Test Description:

This test is to make sure that the data mart is correctly populated using a source i2b2 database.

Preconditions:

- Data mart exists.
- Source i2b2 database is accessible and running

Input:

- The Ip address of where the data loader source is located.

Postconditions:

- Data mart is updated with correct data.

Test Steps:

- An admin opens a terminal and locates the data loader's source ip address.
- An admin issues the force-fetch command.
- Check that the database is populated with the new data through data checking and logs.

Expected Results:

The data mart is expected to be populated with the new data from the data loader force-fetch command. The user or admin will receive a message that the force-fetch was a success.

2.13: Data loader runs on schedule

Author: Jackson Chandler
User Story Covered: 2.14
Test Description:

This test is to ensure that the database is updated when it is scheduled to do so.

Preconditions:

- Data that is to be put into the data mart needs to be in i2b2 format

Input:

- No input other than data from scheduler

Postconditions:

- Data mart has correct data and updated when scheduled.

Test Steps:

- Set the scheduler to update at a certain time in the near future.
- Check the data and logs before the set time.
- Wait until set time has passed.
- Check the data and logs after the set time.
- Compare the old data and logs to the new to show success.

Expected Results:

The data mart is expected to be updated after the time set to update has passed. The new data will be accessible now.

2.14: Admin edits data loader scheduler

Author: Cody Moffitt
User Story Covered: 2.15
Test Description:

This test will verify that the command line tool correctly edits the data fetcher configuration file and doesn't allow invalid data.

Preconditions:

- The source I2B2 server is up and running

Input:

- Administrative username
- Administrative password
- Values to be covered in Test Steps

Post Conditions:

- The data loader configuration file will specify a fetch time of 10:00
- The data loader configuration file will specify an interval of 3

Test Steps:

- Start the data loader command line tool
- Enter “edit configuration” when presented with a list of options
- Enter “fetch time” when presented with the next list of options
- Enter “00:00”
- Enter “y” to confirm the change, the first menu should be shown again
- Enter “edit configuration” when presented with a list of options
- Enter “fetch interval” when presented with the next list of options
- Enter “2” (for 2 days)
- Enter “y” to confirm the change, the first menu should be shown again
- Enter “exit” to exist the data loader command line tool
- Open the data loader configuration file with a text editor
- Find the line “fetch time: 00:00” to verify that the configuration was changed
- Find the line “fetch interval: 2” to verify that the configuration was changed
- Repeat the process, but enter “24:00” for fetch time, the tool should reject this change, since “23:59” should be the highest acceptable fetch time
- Repeat the process, but enter “10:00” for fetch time, and “3” for “fetch interval”
- Open the data loader configuration file to verify these changes have been made

Expected Results:

The data loader configuration file should contain the two lines:
“fetch time: 10:00”
“fetch interval: 3”

2.15: Admin changes source database

Author: Oliver Payne
User Story Covered: 2.16
Test Description:

This test will verify that the user is able to change the source database of the data loader. In order to test this, the user must compare specific entries in the old data to the data that is retrieved from the new database. The new entries should all be different, since they are being pulled from a different database. A successful change in the source database is verified by saving the records of a specific entry and making sure that it does not appear in the newly fetched records.

Preconditions:

- There are patient records in the data mart

Input:

- A string of characters representing the location of the new database.

Postconditions:

- The data loader is set up with the new source database and is retrieving data from it

Test Steps:

- Directly query the data mart for patients with the first name “John”
- Select one of the patients and save the corresponding ID number
- In the config file for the data loader, change the source database entry
- Go to the data loader command line tool. Choose the menu option for “Change source Database”
- Force a data fetch from the new database
- Query the database to ensure that new entries were retrieved
- Search for the patient ID number retrieved above

Expected Results:

When the data loader’s source database has been changed and a fetch has been forced, there will be a new set of data in the data mart. This is tested by querying the database. The load was successful if there are entries in the data mart. Searching for the patient ID number verifies that the data read in from the new database is different than the previous data. The search should come up empty because the old data no longer exists.

2.16: Admin creates data mart from script

Author: Andrew Pachuilo
User Story Covered: 2.17
Test Description:
This will ensure that the script correctly creates the data mart.
Precondition:
- Data mart doesn't already exist.
- The source i2b2 database is accessible and running.

Inputs:
- `./create-data-mart`, data mart creation script

Post Conditions:
- Data mart now exists

Test Steps:
- Run the data mart creation script.
- Run a SQL query for printing all the tables existing in the newly formed database.
- The query's results are checked against the proposed schema (See HLD 4.1).

Expected Results:
The query returns results that represent the proposed schema (See HLD 4.1).

2.17: AI controller spawns instance of AI

Author: Andrew Pachuilo
User Story Covered: 2.18
Test Description:

This test is to ensure that the AI controller spawns AI instances and doesn't go over its maximum limit on spawning processes..

Preconditions:

- Dummy questions exist in the data mart.
- The number of dummy questions exceeds the maximum number of processes allowed.
- Each AI instance is processing dummy tests and returning dummy data.
- Each AI instance eventually terminates and logs when it terminates.

Inputs: N/A
Post Conditions:

- AI instances are running concurrently in the background.

Test Steps:

- Note the maximum limit on processes.
- The AI controller is started via command line from it's source directory.
- AI controller prints to a log a timestamp with the current number of processes it currently contains.
- It spawns an instance of an AI algorithm passing it a unique ID.
- Each AI instance prints to a log the ID it was given and the timestamp at once it started.
- Once an AI instance is finished it prints its ID and the timestamp at once it ended.

Expected Results:

The log should show that AI instances are being spawned and that the AI controller reached its maximum limit on the number of processes but never exceeded it.

2.18: AI processes a question

Author: Cody Moffitt
User Story Covered: 2.19
Test Description:

This test will verify that the AI has processed a question.

Preconditions:

- The tester is logged in to the web app with user name “TestUser” and password “TestPass”
- The test user has no questions on their account
- The tester has administrative access to the database

Input:

- No specific input. User button clicks

Post Conditions:

- TestUser will have one question on their account with a status of “Waiting for Feedback”
- An AI model will be stored as a BLOB in the data mart for that question

Test Steps:

- Login to the web app with test user credentials
- Create a new question
- Verify that the question is “queued” status
- Wait up to 10 minutes for the question’s status to change to “Need Feedback”
- Click the question and verify an answer is provided
- Login to the SQL database and run this query:
- Select AI from User u

```
        Inner Join UserSession u on u.ID = us.UserID
        Inner Join Question q on u.ID = q.User ID
        Inner Join AIModel ai on q.ID = ai.QuestionID
        Where u.Name = ‘TestUser’
```

- Verify that the AI column is returned, and within it is contained binary data (BLOB)

Expected Results:

The question posed by the user will now have an answer, and an AI model will be stored in the data mart.

2.19: C4.5 Decision Tree Classifier

Author: Cody Moffitt
User Story Covered: 2.19
Test Description:
This test will verify that the C4.5 Decision Tree classifier is working as expected.
Preconditions:
- The tester has a driver to test the algorithm

Input:
- iris.arff (included with this document)
- iris_test.arff (included with this document)

Post Conditions:
- The test driver has generated output matching the data in j48Classifier.arff

Test Steps:
- Run the automated test driver for C4.5 Decision Tree classifier with iris.arff as the training data and iris_test.arff as the test data.
- Verify that the output rows match the @data section of the j48Classifier.arff (included with this document)

Expected Results:
The output of the test driver should match the three rows of data in j48Classifier.arff

2.20: Random Forest Classifier

Author: Cody Moffitt
User Story Covered: 2.19
Test Description:
This test will verify that the Random Forest classifier is working as expected.
Preconditions:
- The tester has a driver to test the algorithm

Input:
- iris.arff (included with this document)
- iris_test.arff (included with this document)

Post Conditions:
- The test driver has generated output matching the data in randomForest.arff

Test Steps:
- Run the automated test driver for Random Forest classifier with iris.arff as the training data and iris_test.arff as the test data.
- Verify that the output rows match the @data section of randomForest.arff (included with this document)

Expected Results:
The output of the test driver should match the three rows of data in randomForest.arff

2.21: Naive Bayes Classifier

Author: Cody Moffitt
User Story Covered: 2.19
Test Description:

This test will verify that the Naive Bayes classifier is working as expected.

Preconditions:

- The tester has a driver to test the algorithm

Input:

- iris.arff (included with this document)
- iris_test.arff (included with this document)

Post Conditions:

- The test driver has generated output matching the data in naiveBayesClassifier.arff

Test Steps:

- Run the automated test driver for Naive Bayes classifier with iris.arff as the training data and iris_test.arff as the test data.
- Verify that the output rows match the @data section of naiveBayesClassifier.arff (included with this document)

Expected Results:

The output of the test driver should match the three rows of data in naiveBayesClassifier.arff

2.22: Neural Network Classifier

Author: Cody Moffitt
User Story Covered: 2.19
Test Description:

This test will verify that the Neural Network classifier is working as expected.

Preconditions:

- The tester has a driver to test the algorithm

Input:

- ionosphere.arff (included with this document)
- ionosphere_test.arff (included with this document)

Post Conditions:

- The test driver has generated output matching the data in neuralNetworkPerceptronClassifier.arff

Test Steps:

- Run the automated test driver for Neural Network classifier with ionosphere.arff as the training data and ionosphere_test.arff as the test data.
- Verify that the output rows match the @data section of neuralNetworkPerceptronClassifier.arff (included with this document)

Expected Results:

The output of the test driver should match the two rows of data in neuralNetworkPerceptronClassifier.arff

2.23: Support Vector Machine Classifier

Author: Cody Moffitt
User Story Covered: 2.19
Test Description:
This test will verify that the Support Vector Machine classifier is working as expected.
Preconditions:
- The tester has a driver to test the algorithm

Input:
- iris.arff (included with this document)
- iris_test.arff (included with this document)

Post Conditions:
- The test driver has generated output matching the data in smoSupportVectorClassifier.arff

Test Steps:
- Run the automated test driver for Support Vector Machine classifier with iris.arff as the training data and iris_test.arff as the test data.
- Verify that the output rows match the @data section of smoSupportVectorClassifier.arff (included with this document)

Expected Results:
The output of the test driver should match the three rows of data in smoSupportVectorClassifier.arff

2.24: Ensemble Learning

Author: Cody Moffitt
User Story Covered: 2.19
Test Description:
This test will verify that ensemble learning is working as expected.
Preconditions:
- The tester has a driver to test the algorithm

Input:
- ionosphere.arff (included with this document)
- ionosphere_test.arff (included with this document)

Post Conditions:
- The test driver has generated output matching the data in stackingJ48andBayes.arff

Test Steps:
- Run the automated test driver for Ensemble Learning with ionosphere.arff as the training data and ionosphere_test.arff as the test data.
- Verify that the output rows match the @data section of stackingJ48andBayes.arff (included with this document)

Expected Results:
The output of the test driver should match the two rows of data in stackingJ48andBayes.arff

2.25: Admin edits AI config file and applies update

Author: Jackson Chandler
User Story Covered: 2.20, 2.21
Test Description:

This test is to ensure that the config file has been updated and applied to the system.

Preconditions:

- Config file exists
- The edited version of the config file is valid

Input:

- Any changes to the config file

Postconditions:

- The config file is updated.

Test Steps:

- The user opens the config file in preferred text editor.
- The current parameters are noted for later comparison.
- Tester changes parameters.
- They then save the changes to the file.
- When the file is saved or scheduler runs the parameters will be updated.
- The user will be notified of the update.
- AI controller prints parameters to compare to previously set ones to ensure that the change was saved.

Expected Results:

The user will then receive a notification that the updates were applied. The new parameters will be printed, so the user can see that the parameters have been changed.

2.26: User is automatically logged out

Author: Oliver Payne
User Story Covered: 2.22
Test Description:

This test will verify that the user is automatically logged out of the web client after being inactive for a period of time. To test this, the user will remain inactive in a valid session to see if the web client automatically logs him or her out. The period of inactivity can be set lower than usual for the testing phase. In this case, it will be set to 2 minutes.

Preconditions:

- The user is logged into a valid session

Input:

- N/A

Postcondition:

- User is logged out of web client due to inactivity

Test Steps:

- Log into the web client
- Perform no actions for 2 minutes
- Check to see if the user has been logged out

Expected Results:

The web client is set up to automatically invalidate a user's session due to extended inactivity. Therefore, after 2 minutes, the scenario will finish with the user's session being invalidated. At this point, the user will be prompted to reenter his or her login credentials

Chapter 3: Notes, Issues, Assumptions

- 2.15: It may be difficult (or impossible) to compare anonymized patient data. The complexity of this test will increase when factoring in this constraint. However, if this is the case, any alternate test that verifies that entries of the data mart differ before and after fetching new data would suffice to complete this unit test.
- 2.19 - 2.24: Automated test drivers will have to be developed for classifier and optimization testing, the classifier parameters should match the default parameters provided in WEKA's explorer, to match the test output. These tests could also be rewritten to use actual patient data when we have it organized.