**Predictive Maintenance System**

**Design Document**

Team: Ostriches

Author: Nicholas La Sala

Product Owner: Craig Wert

Scrum Master: John Stranahan

Development Team: Nicholas La Sala, Joshua Jackson, Tapan Soni, Michael Matthews

Sponsor: ASRC Federal Mission Solutions

**Table Of Contents**

1 Cover……………………………………………………………………………...……….1

2 Table of Contents………….……………………………………………………………....2

3 Introduction……………..………………………………………………………………....3

3.1 Purpose….…………………………………………………………………………3

3.2 Scope………………………………………………………………………………3

3.3 Exclusion, Assumptions and Limitations…………………………………………3

3.4 Definition of Done………………………………………………………………...3

4 Architecture………………………………………………………………………………..4

4.1 Logical View...…………………………………………………………………….4

4.2 Development View...…………………..………………………………………….5

4.3 Process View...…………………………………………………………………….6

4.4 Activity View……………………………………………………………………...7

5 Use Cases ………………………………………………………………………………....4

6 Python……………………………………………………………………………………..5

6.1 Classifier…………………………………………………………………………..8

6.2 Training Algorithm...……………………………………………………………...9

6.3 Generate Data…………………………………………………………………….10

6.4 GUI………………………………………………………………………………11

7 Python Packages…………………………………………………………………………12

7.1. Scikit Machine Learning..………………………….…………………………….12

7.2. Numpy and Scipy………………………………………………………………...12

8 Data………………………………………………………………………………………13

9 Experiment Log………...………………………………………………………………..14

10 Data Sample……………………………………………………………………………...16

**3. Introduction:**

This is the design document for a predictive maintenance system for the engine of battleship housing the Aegis defense system. A predictive maintenance system use data collected from sensors to determine if the engine needs maintenance or can continue running without being checked. To accomplish this a portable supervised machine learning classifier will be produced and trained to parse the data and determine whether or not maintenance is required.

**3.1. Purpose:**

Normally these larger components would have regularly scheduled maintenance at fixed periods of time where a staff member will have to run predetermined manual diagnostic checks, even if there is no indication that there is a problem. By implementing a predictive maintenance system, a naval crew would be able to reduce the costs of maintenance by only needing to perform checks and part swaps when they absolutely necessary.

**3.2. Scope:**

As of sprint 2 the current goal for the final delivered increment is to have a portable Machine Learning classifier that can be integrated into a larger system, and receive big data to make predictions with at least at 60% degree of accuracy.

**3.3. Exclusions, Assumptions, and Limitations:**

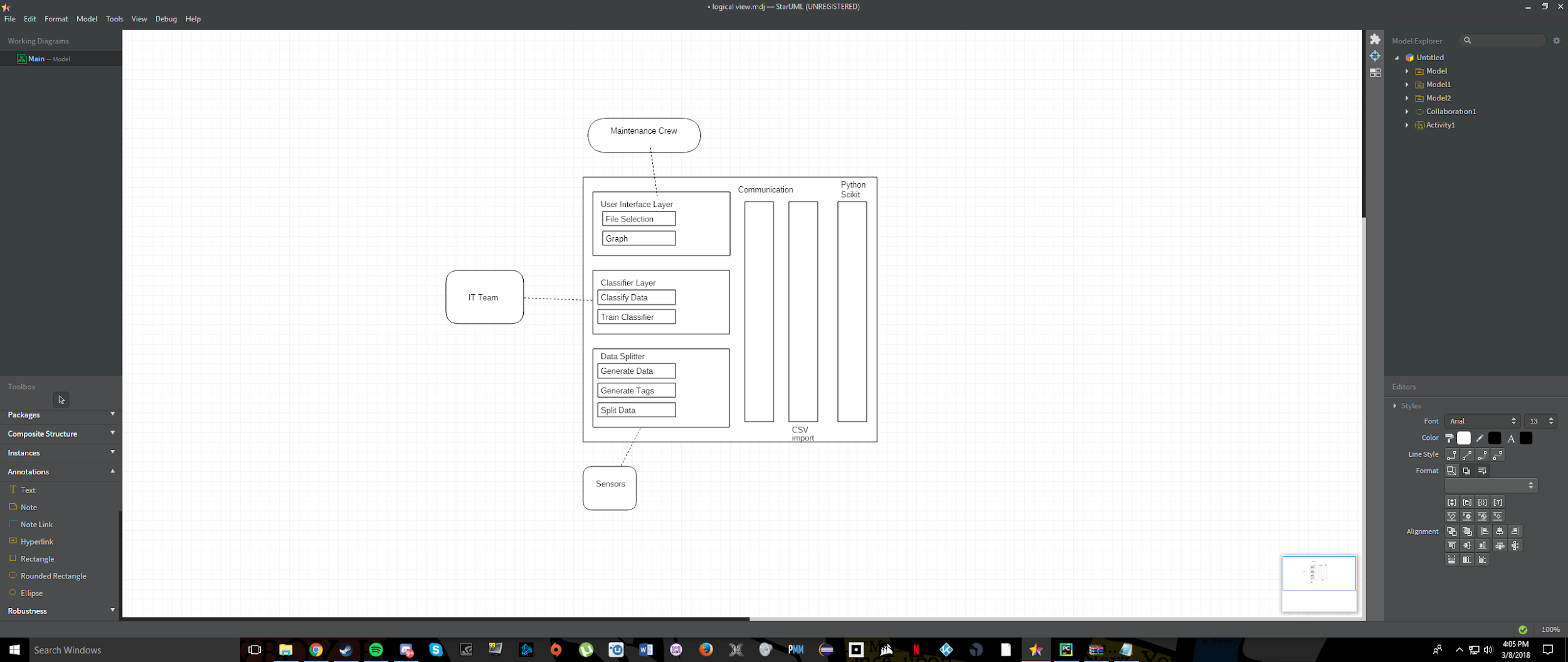
The classifier was built with context for the data being used, nor any information about the sensors being used to collect the data, and as result the classifier would made with the flexibility to work with any data set. Another limitation this has created is that any real time or internet of things functionality cannot be implemented without information about how to data will be collected and stored, or the system using the classifier.

**3.4. Definition of Done**:

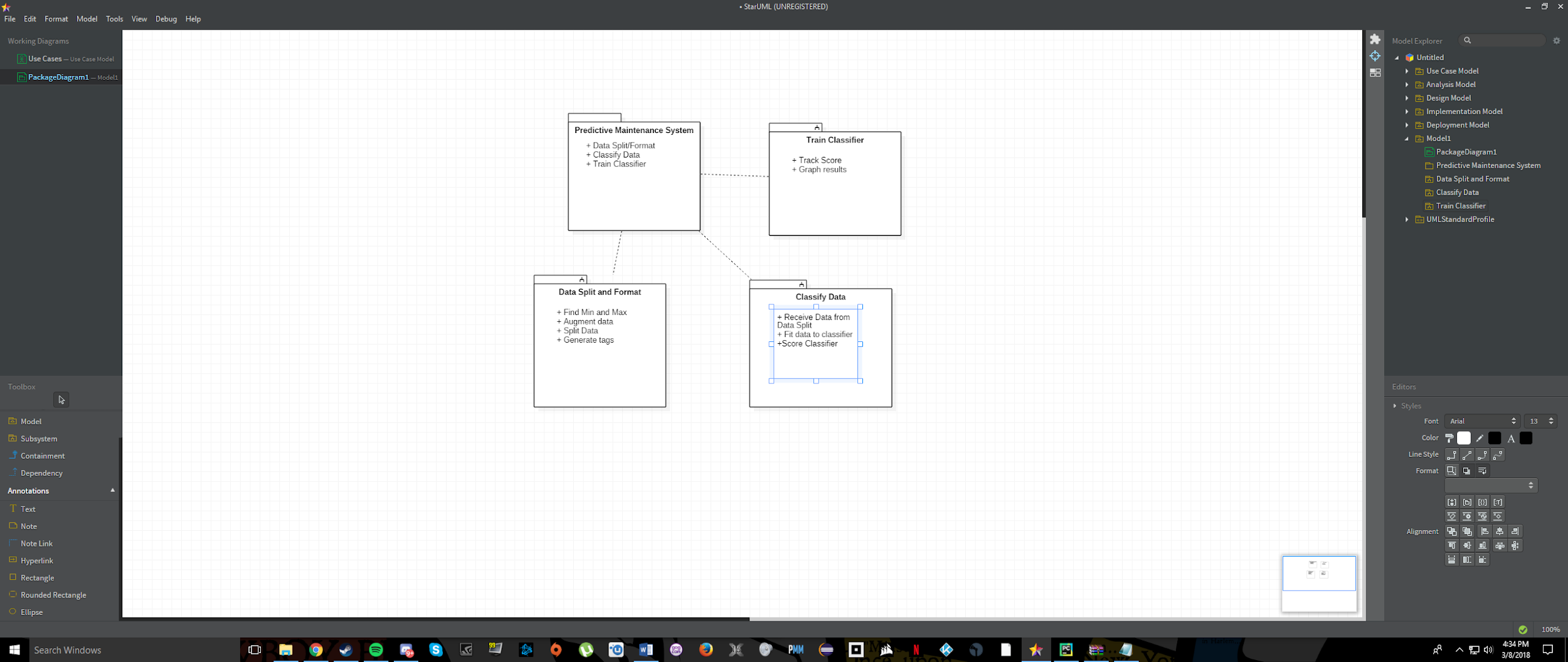
1. **Code:**
   1. Source Code must be documented with concise and clear comments
   2. Source Code must be uploaded to the GitHub repository for version control and code review.
   3. Source Code must be formatted properly as to be easily readable
2. **Experimental Log:**
   1. For each classifier, a detailed experimental log must be developed.
3. **Testing:**
   1. All modules of the program -- GUI, and training algorithm must be tested with the master data csv file that is provided, and for each test for the training algorithm, detailed notes must be logged in the experimental log.

**4. Architectural views:**

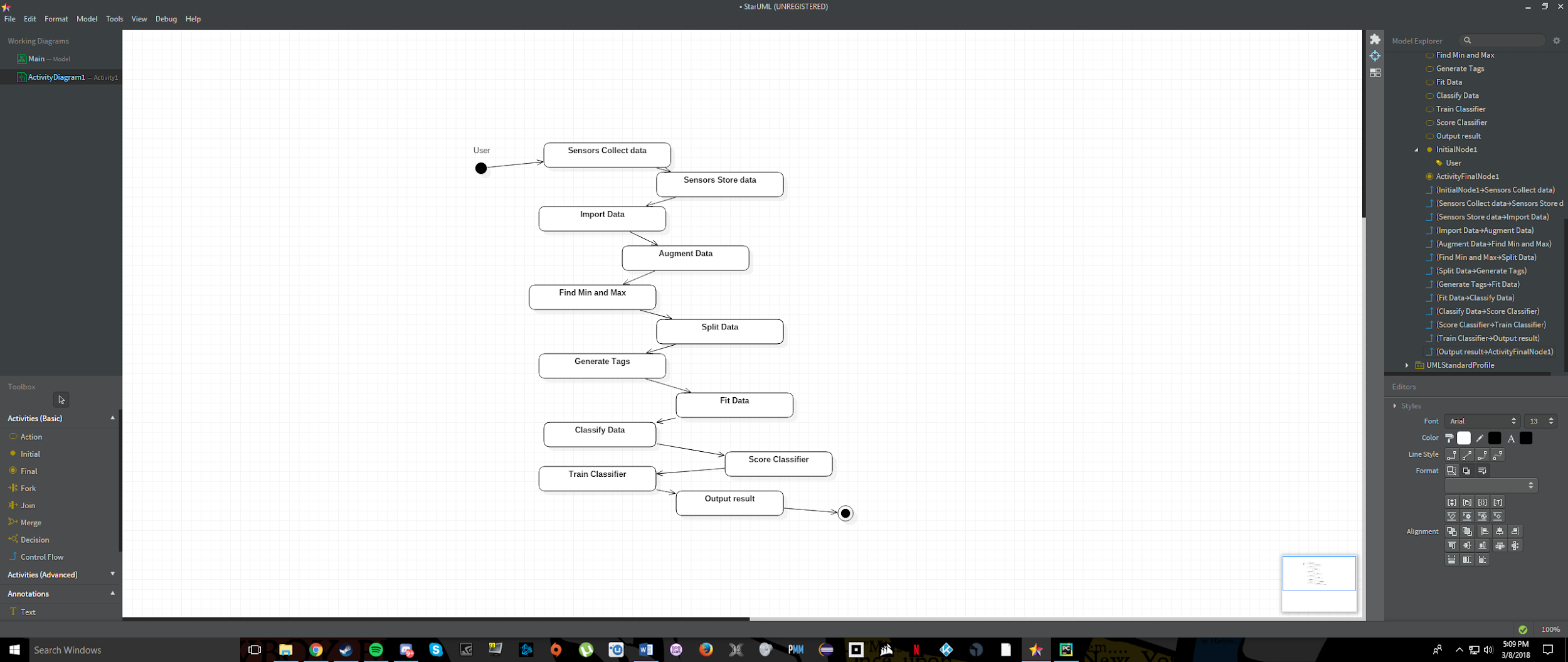
**4.1. Logical View:** The Key abstraction of the System.



**4.2. Development View:** Separates what is needed for each part.

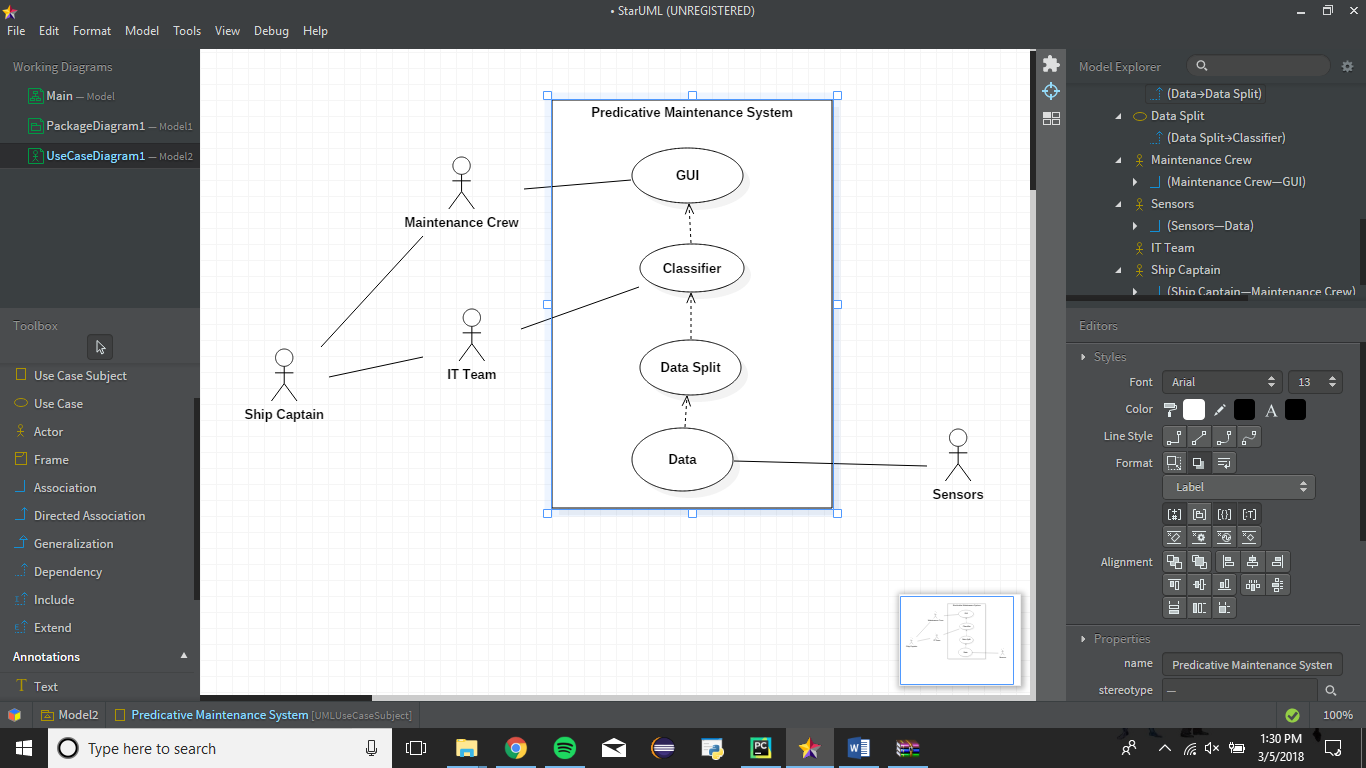


**4.3. Process View:** Shows how the system will work at run time.



**5. Use Case:**

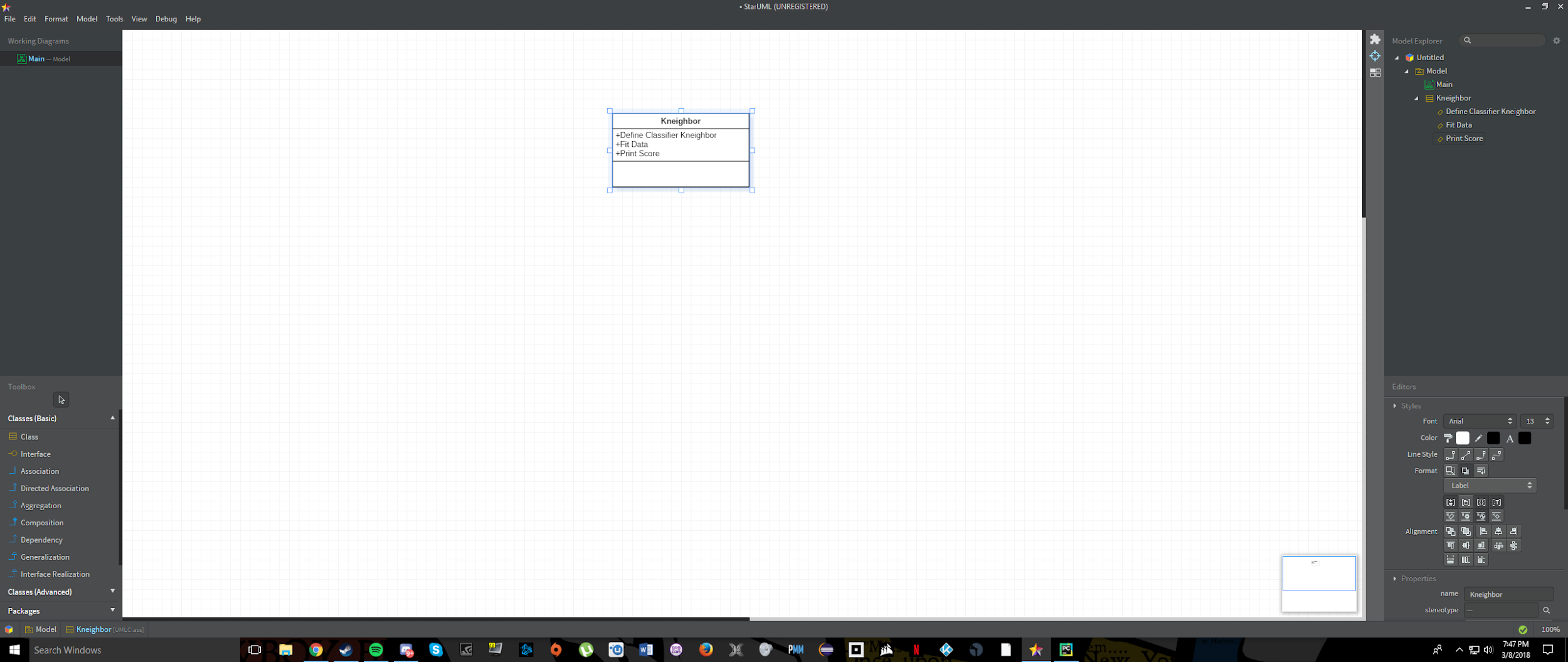
Actors: The actors that would be interacting the system when implemented with a battleship’s engine are the maintenance crew, the IT team, the ship captain, and the sensors that are collecting data. The maintenance crew are the people responsible for making sure the that the engine is running properly, and to make this job easier they would select the data set that need to check using the GUI, which runs the classifier to determine if the engine need maintenance. The IT team would monitor the classifier to ensure that it does not produce results that are not accurate. The ship captain would communicate with the staff to make sure that both functions are working as designed. The sensors that collect data from the engine would store that data as csv files that can be used by the maintenance team, with the GUI.



**6. Python Files:**

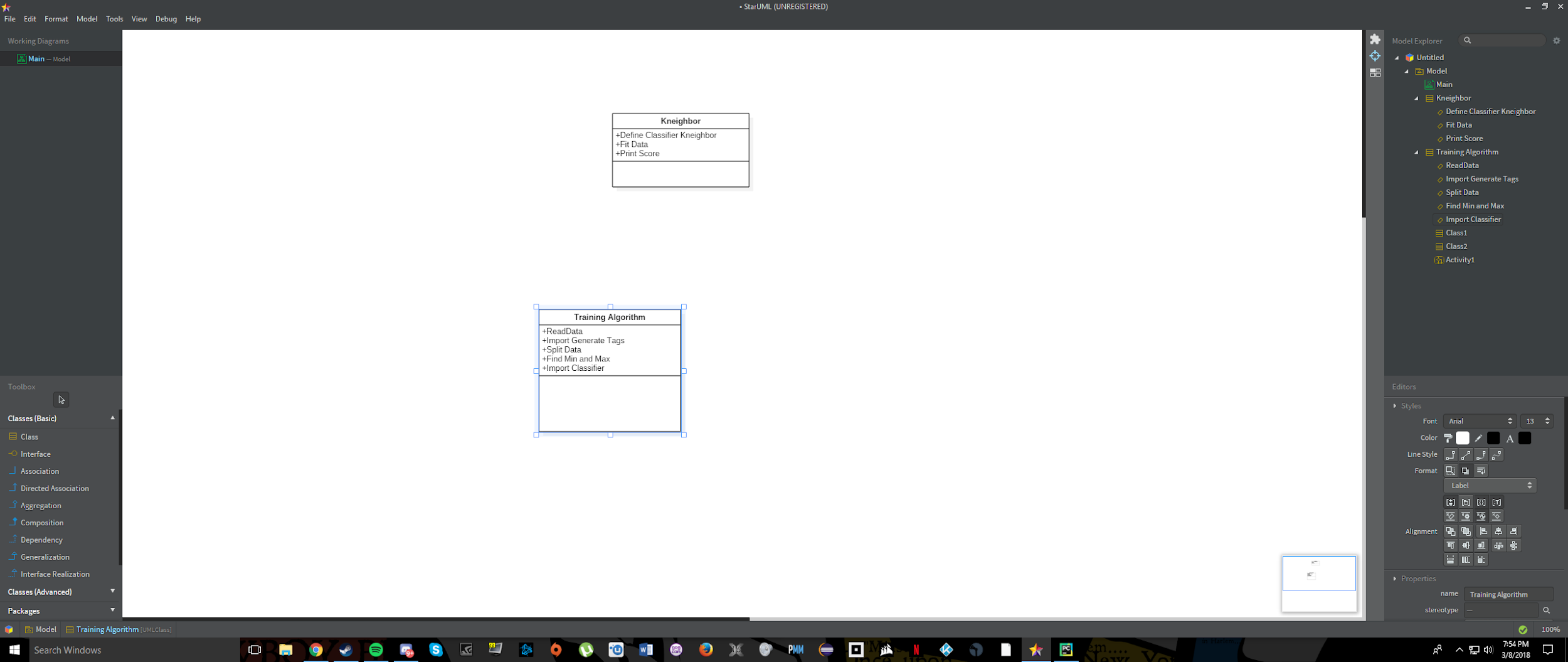
**6.1. Classifier:**

The classifier object is what uses machine learning to tag the data and decide whether or not maintenance is required. For testing purpose, a K-Neighbor, Decision Tree, and LinearSVC classifier where created and tested to compare the accuracy of the different types of classifier. As of the most recent version the K-neighbor Tree classifier has the most accurate scores of the classifier produced thus far.



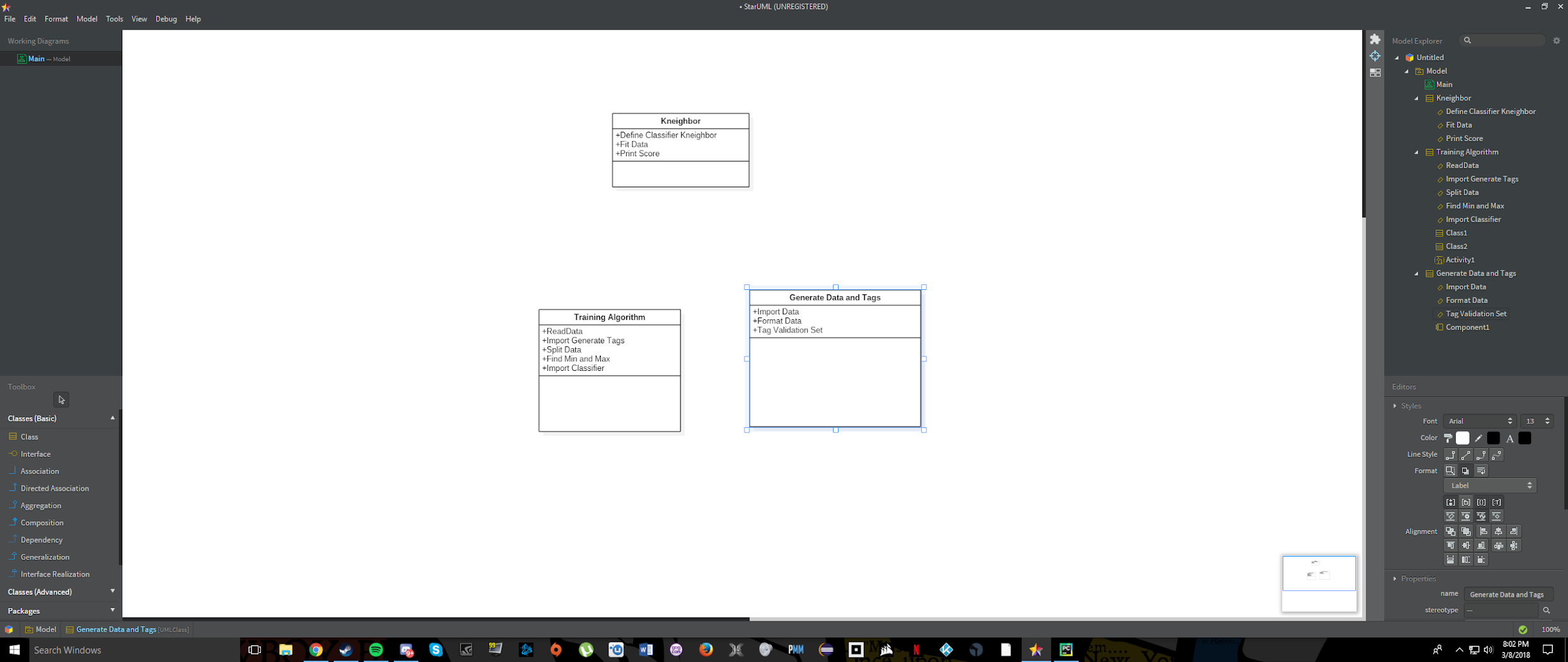
**6.2. Training Algorithm:**

The training Algorithm is the file that brings all of the file that uses the other modules to create a classifier, make predictions, and score the accuracy of the classifier. This is also the file that the GUI interacts with.

****

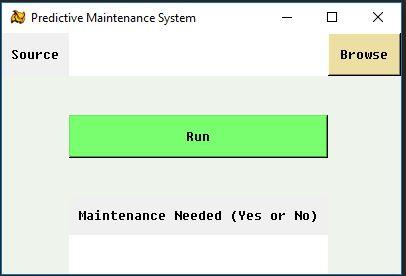
**6.3. Generate Data/Tags:**

These modules augment to sample data and sets and appropriate amount of tags for testing and training purposes. These classes were written to avoid overfitting the data, which would introduce bias to our testing that would score the classifier more accurate than it actually is.



**6.4. GUI:**

This is the visual interface that the end user would be interacting with when using our system after it is integrate with a ships engine. The GUI will allow to user to browse for a set of data from their hard drive and use to the classifier to determine whether the ship requires maintenance.



**7. Python Packages:**

**7.1. Scikit Learn: Machine Learning:**

Scikit Learn is an open source Python Library that provides prebuilt data types and tools relating to machine learning that would be overwhelming to create by hand. From SciKit we are using the testing the K-Neighbor, Decision Tree, and linearSVC Classifiers, and keeping an experiment log of the results to determine which classifier and with which values can tag the data with the best accuracy.

**7.2. Numpy and Scipy:**

Numpy is a package that provides numerous tools for scientific computing that is used to read and format the CSV files that contain the data to be used in the project. Scipy is also a scientific computing library that’s installation is required prior to the installation of Scikit for its libraries to work.

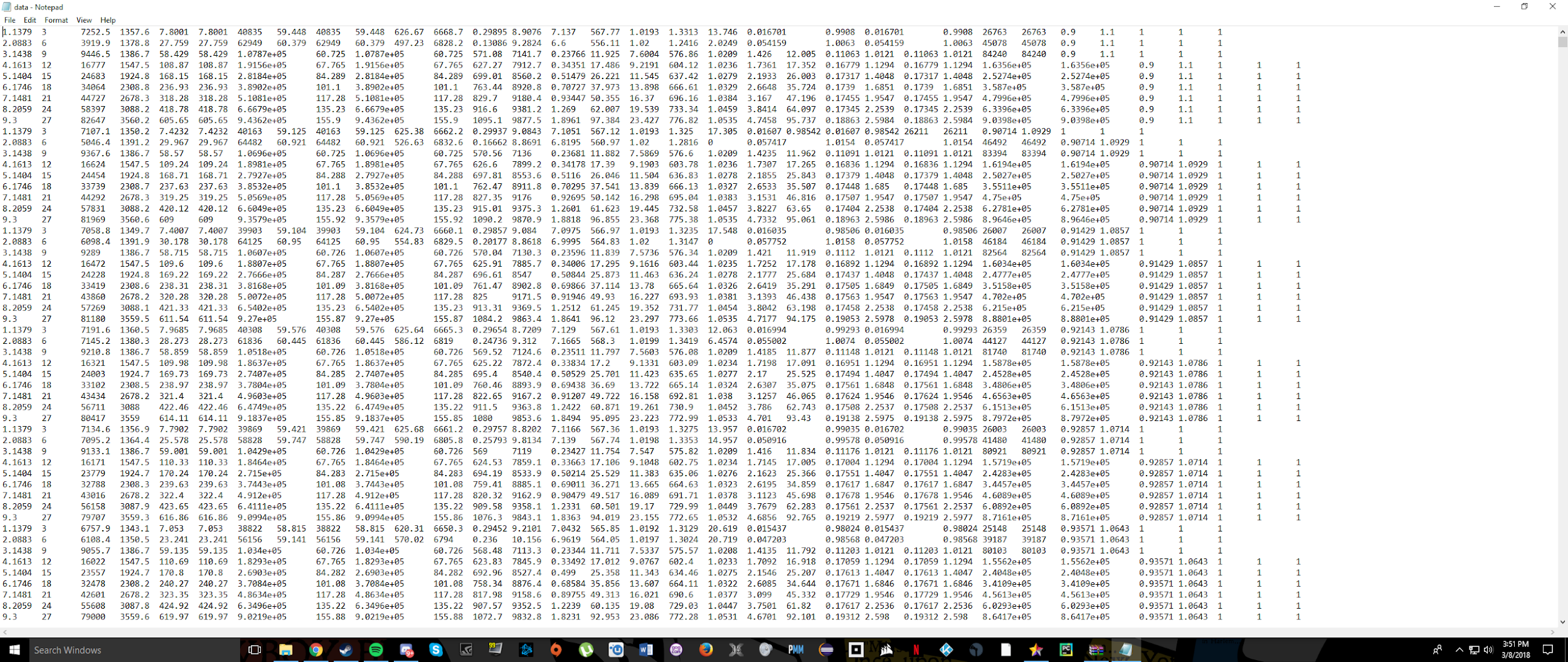
**8. Data**:

The data collected by sensors are divided into a training set and a validation set that will be used to train the classifier. 70% of the data is used for the training set, which is the set that the classifier tags, which will be scored for accuracy. The remaining 30% of the data is used for the validation set which is used by the training algorithm to score the classifier and train it so that its accuracy improves.

**9. Experiment Log**:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Date* | *Classifier Version* | *Training Set* | *Validation Set* | *Score* | *Percentage* | *Tagging system* | *Execution Speed (seconds)* | *Result* |
| 3/1 | K-nearest Neighbor(k=59) | First 70% | Remaining 30% | 0.2007818555 | 20.08% | First half 0 rest 1 | apprx. 30s | probably need to randomize which data goes to training set and validation set |
| 3/2 | K-nearest Neighbor(k=99 | First 70% | Remaining 30% | 0.2205828275 | 22.06% | First half 0 rest 1 | approx. 55s | For K Neighbor = 99, the test ran for 55 seconds. We might need to change the tagging from 50% good to 50% bad, to a more gradient tagging system such as 70 good and 30 back or 60 good and 40 bad |
| 3/2 | K-nearest Neighbor(k=99) | First 70% | Remaining 30% | 0.2205874 | 22.06% | First half 0 rest 1 | approx 113.7 | Everything was the same, but the time was much greater than the other runs |
| 3/2 | K-nearest Neighbor(k=999) | First 70% | Remaining 30% | 0.3664877036 | 36.65% | First half 0 rest 1 |  |  |
| 3/3 | K-nearest Neighbor(k=99) | First 70% | Remaining 30% | 0.29999 | 30.00% | First 79% to 0, and second 21% to 1 | approx 47.6 | Changed the tagging to 79% 0 and 21% to 1 |
| 3/3 | K-nearest Neighbor(k=79) | First 70% | Remaining 30% | 0.2999 | 30.00% | First 79% to 0, and second 21% to 1 | approx 43 | Changed the number of neighbors to 79 |
| 3/3 | K-nearest Neighbor(k=79) | First 70% | Remaining 30% |  |  | First 65% to 0 and second 35% to 1 |  |  |
| 3/3 | K-nearest Neighbor(k=999) | First 70% | Remaining 30% | 0.28386 | 28.38% | First 60% = 0 Rest 40% = 1 | approx 89 | Changed the tagging to 60% 0 and 40% 1 |
| 3/3 | K-nearest Neighbor(k=99) | First 70% | Remaining 30% | 0.2185 | 21.85% | First 60% = 0 Rest 40% = 1 | approx 44.2 | Changed the # of neighbors to 99 |
| 3/3 | K-nearest Neighbor(k=99) | First 70% | Remaining 30% | 0.33333 | 33.33% | First 80% = 0 Rest 20% = 1 | approx 47.6 | Changed the tagging to 80% to 0 and the second 20% to 1 |
| 3/3 | K-nearest Neighbor(k=999) | First 70% | Remaining 30% | 0.33333 | 33.33% | First 80% = 0 Rest 20% = 1 | approx 157.63 | Changed the number of neighbors to 999. The accuracy was the same as k=99 but the time was greatly increased |

**10. Data Sample**

****