***Predictive Maintenance***

***Using Machine Learning***

***Prepared By: Joshua J.***

***Date: April 3, 2018***

***By: Team Ostriches***

***Product Owner: Craig W.***

***Scrum Master: John S.***

***Development Team: Tapan S., Joshua J., Nic L., Mike M.***

**Table of contents:**

Purpose………...……………………………………………………………………………………….2

Summary of Deliverables Generated………………………………………………………………..2-4

Variations from Plan……………………..…………………………………………………………….4-5

Why What Was Done Is Valid…..………..……………………...…………………………………...4-6

Roles and Responsibilities……………………………………..……………………………………..7-8

Traceability Matrix…..………………………………………………………………………………….8-9

1. ***Purpose :***

The document you are reading currently should serve as validation of why our program satisfies the requirements up to this date and time, we will go into detail of the deliverables, the variations that we made from the plan, why what was done is valid, roles and responsibilities, and traceability matrix. The following document is tailor made towards our sponsors from ASRC Federal Missions Solutions or any naval maintenance workers interested in learning more about the path we took and why it was optimal with our requirements.

***2.) Summary of Deliverables Generated :***

==========================================================

0.1-0.4:

Nothing of importance to the project was conducted on sprint 0

Summary:

N/A

=====================================================================

1.0-1.12:

1.1 - 1.4 Nothing of importance was completed it was setting up for project.

1.5 - Created preliminary GUI for the program using TKinter

1.8 - Create an Experiment Log that logs Python Scikit classifier results

1.9 Input CSV source file as an array in the program

Summary:

This increment was for the dev team to go over the rest of the spikes that were needed to be covered in order to further the project. With that being said we setup the shells for the GUI, Experiment Log, and setup the CSV source file as an array for the classifier.

=====================================================================

2.0-2.18:

2.3 Connect the classifier with GUI

2.4 Put a timer on the GUI (tracks the time to run the application)

2.6 Program Driver Creation

2.7 Experiment with different classifiers

2.8 Update the Experiment log

2.9 Conduct test runs to add to the experiment log

2.11 Re-scale Data set using normalization

2.12 Put a timer on the training algorithm

2.13 Create a Decision Tree Classifier

2.14 Augment the .CSV data

2.15 Train the classifier

2.16 Generate random data

2.17 Randomize how training set and validation set get selected

2.18 Generate Histograms

2.19 Develop a Binary Classifier

Summary:

Once the spikes were taken care of and understood, The coding had commenced. This was a substantial increase of production from sprint zero, this increment was getting the GUI up and running along with the Classifier running, with augmented .CSV data, generated random, and randomize training set and validation set.

=====================================================================

3.0-3.12:

3.1 - Make classifier serializable

3.2 - Normalize/Generate one standard random data set

3.3 - Make Template for Code Commenting (Readability)

3.4 - Add Variability (Seeding Data)

3.8 - Redo Split Function

3.9 - Test Multiple Data Rows

3.10 - Extend the bounds of generated data

3.11 - Fix The Bug Related to Trying Multiple Runs of the program

Summary:

This increment was another good one, as normalizing, seeding, and commenting code were all great additions. Seeding created stability for the program. The redo split had to be fixed to randomized as we first had it set only to pick from the top 30 percent of the data for validation which always yielded that the machine did not need to be repaired. Generalized the data by increasing the bounds by timesing the max by two and dividing the min by two. Created the pickle to serialize the classifier for transport. Test multiple data rows by picking from the data set every 100th data point then finding the average for a validation of yes or no. the bug happened when doing the serializing of the Classifier, was fixed once we finished the spike.

=====================================================================

***3.) Variations from Plan :***

2.13 Create a Decision Tree classifier

During this increment it was decided that other classifiers would be experimented with to see what would yield a better result for predictive maintenance. This is where the first variation of the plan occured. There was a need wanted to implement different ways of making a prediction for the data through multiple classifiers. Allowing the user to choose which classifier they wanted to use when running the program. It was then recommended that there should only be a focus on one classifier for the project at hand.

So, what was decided was to put the other classifiers in a text file. So, that if a user wanted to use another classifier for the predictive maintenance they may have the option to do so. Justification for this rationale is with decision tree it is easier to draw out what is occurring in the program making it easier to show and tell figure heads of company who may have little knowledge of the application. Why K-Nearest Neighbor was chose ultimately was because the comfort with the classifier, as well with the best produce results in the experiment log.

3.4 Add Variability (Seeding)

This was a variation from the original plan because instead of are classifier being completely random when selecting data. We added seeding which gives the same random starting from a specified point. This allowed us to gain some sort of constance for the application, which we were actively seeking out for. This affects the application in only making it better as if an error occurs you have a more finite number answer of where the problem could occur. As where before, it was harder to tell because there was no constantance.

4.0 DoD Non-functional requirements

The team was required at the beginning of the project to make a security password to protect the program. Later on, it was discussed with our sponsor that this wasn’t needed to be considered done. The Non-functional requirements then for changed to having the program run under a certain time limit and use Python 3.5 or higher. This was also more difficult to do with TKinter as we found out it was a single thread GUI framework. This limits the options for the GUI thus resulting in problems for the Non-functional requirements. This would be a good start for the next team if ASRC decided to continue doing this for next semester.

***4.) Why What Was Done Is Valid :***

What was required was met in this project. Will go over the requirements and justify how it was satisfied in a way that is acceptable. Team Ostrich, had a DoD to be covered, with that being said there were a plethora of options to get the job done. Meaning, that what their could have been many ways the project could have been done differently with certain approaches. With that let’s go over and explain those spots with why certain options were the right approach for the team!

Option one was the GUI, as this was the first wall hit by team ostrich. The GUI could have been made with PyQT, TKinter, and WXPython. The team decided it was wiser to go with TKinter. Reasoning for this is that it comes with python making it easily accessible to anyone who would like to modify the GUI if need be later on, since it ships with Python. The others were not as accessible so it made the team shy away from them. Plus, the simplicity that it had was attractive, since the team was recommended not to focus on the GUI but the Classifier and its accuracy. Also, when making the GUI and learning that the the aesthetic was high priority, we decided to make the browse and run button a single button. Satisfying, two DoD’s while also making the program more efficient by it not having to many buttons making its usability higher. After creating the buttons, the team made a Title with the rowan icon to show that the team and the product was produced by rowan students. Following that with one button that browses and runs the program ostriches made the browse only take .CSV files to work with the program at hand. After you can see the location of the file in a text box on the GUI. Following that after the file is selected it will automatically run, it was decided for that approach because it would not waste time for the user. This then gave a binary result in a Yes or NO.

Option two was that the team must come to a conclusion on the algorithm to analyze the data (Classifier). The team suffered with this for awhile as the first algorithm or Classifier chosen was a Supervised algorithm known as K-Nearest Neighbor. With this the understanding of how machine learning started to get easier and easier to grasp. Following learning this the team then went on to a couple other Classifiers or algorithms. They were as followed, Decision Tree, Neural Net, Gaussian Naive Bayes, Gradient Descent. With experimenting with all of these there was a decision to be made. Between all of the following algorithms which one was going to stay as the main algorithm for the project. Based off the results, comfort, and understanding.

The team thought it was best to continue with K-Nearest Neighbor. Within the experimental log anyone can see it was producing the best accuracy results higher than what was in the DoD. Actually the numbers it has produced are considered outstanding coming in at 90% accuracy. Also, since the team learned how to utilize K-Nearest Neighbor first, it was easily the one algorithm that gave the team the most comfort which should with the results. This also proves that we found a viable way with the algorithm to train itself. Along with selecting the algorithm and training it we had to make it produce a binary result which was tasted in the GUI as stated above giving a Yes or No.(option 1) This also proves with the accuracy so high that the trained algorithm, knows what a “pass” or “fail” is conquering another DoD. Lastly with the Classifier or algorithm we needed to make it portable.

This was done by pickling it or also known as serizalzing the Classifier. Basically this turns the Object Classifier to a text file so you can use a thumb drive/flash drive and transport it then unpickle it and it goes from a text file back to a Object. This was all achieve by making the right choice of picking the best classifier to us while also using the best accuracy. With, that being said as stated in 3 above we had a variation occur because we had coded the other classifiers that worked and we have a text file allowing anyone with access to the code to use another classifier of their choice. Currently we are working on an Unsupervised K-Cluster to run as an option as well.

Third and final option was how to split up the testing and validation data for the Classifier. The team decided that the best fit for this was a 70 to 30 split. 70 going to the testing data, while the 30 went to validation. This was chosen because it was a good way to give accurate results with over fitting too much, but not to a degree that makes it not accurate. This could have been done 60 to 40 or 80 to 20. The team was recommended by a rowan professor Dr Ho to try and work up to 70 to 30. Starting from 80 to 20 and working down as it would be easier to see where the data starts getting in the different shades of gray. Pulling the 20 percent for variation form the top ten from the top (good) vs the ten from the bottom (bad) and increase it till we find a good ratio. This ended up making the splitting of the validation and test data come to a conclusion of 70 to 30 split yielding the best results for accuracy the team.

***5.) Roles and Responsibilities :***

**Dev team members:** *Joshua Jackson, Nic LasSala, Mike Matthews, Tapan Soni*

**Responsibilities** :

*Joshua Jackson* - The responsibility of this dev team member for developing was to reorganize the code (readability) , work on classifier, GUI, and also work on the experiment log. For verifying this member conducted GUI testing including aesthetic and see if the product yields binary results which in turn returns a Yes or No. The solutions this team member would implement would be to the classifier splitting data and seeing if given all data that was needed to continue without repair it would do so. Also, making sure GUI was working properly.

*Nic LaSala* - The responsibility of this dev team member for developing was to reorganize the code (readability), work on classifier, GUI, and also work on the experiment log. For verifying this member conducted GUI testing including aesthetic as well as if the product yields binary results which in turn returns a yes or no. The solutions this team member would implement would be to the classifier splitting data and seeing if given all data that was needed to be repaired it would do so. Also, making sure GUI was working properly.

*Mike Matthews* - Responsibility of this dev team member for developing was to reorganize the code (readability), work on classifier, GUI, and work on experiment log. Also, worked on other classifier (Decision Tree classifier) in getting a drawn out tree. For verifying this member conducted GUI testing including the aesthetic and to see if the product would yield binary results which were yes or no. The solutions this team member would implement would be to the classifier splitting data and seeing if given all data that was needed to continue without repair it would do so. Also, making sure GUI was working properly.

*Tapan Soni -* The responsibility of this dev team member for developing was to reorganize the code (readability) , work on classifier, GUI, and also work on the experiment log. For verifying this member conducted GUI testing including aesthetic and see if the product yields binary results which in turn returns a Yes or No. The solutions this team member would implement would be to the classifier splitting and data seeing if given all data that was needed to be repaired it would do so. Also, making sure GUI was working properly.

***6.) Traceability Matrix:***

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirements Document ID** | **Requirements Description** | **Design Specification ID** | **Test ID** |
| 3.1 | The application must have a GUI | 6.3-6.4 (GUI) | 1 |
| 3.1.1 | The GUI must have a title bar | 6.3-6.4 (GUI) | 1 |
| 3.1.2 | The GUI must have a browse button and the function to allow the user to browse for .CSV files | 6.3-6.4 (GUI) | 1 |
| 3.1.3 | The GUI must show the user the file location they have chosen | 6.3-6.4 (GUI) | 2 |
| 3.1.4 | The GUI must have a run button to analyze the .CSV data | 6.3-6.4 (GUI) | 1 |
| 3.1.5 | The GUI must have an output area, where it shows the user the result of analyzing the data read in by the .CSV data | 6.3-6.4 (GUI) | 1 |
| 3.1.6 | The GUI must have a Help button which explains each part of the GUI to the user and how to use them | 6.3-6.4 (GUI) | 6 |
| 3.2 | The application must have a predictive algorithm to properly analyze the data | 6.1, 6.2 | 3 |
| 3.2.1 | The algorithm must intake data from a .CSV file in the form of an array | 6.3 | 2 |
| 3.2.3 | The algorithm should have an initial accuracy of at least 50% to be improved as it learns | 6.1, 6.2, 6.3 | 2 |
| 3.2.4 | The algorithm must return a binary result of “Needs Repair” or “Does Not Need Repair”, with room to improve to a ternary or even part specific result | 6.1 | 2 |
| 3.2.5 | The algorithm must use statistical classification in order to learn to identify what is a “pass” and what is a “fail” | 6.1 | 3 |
| 3.2.6 | All classifier objects used must be serializable | 6.1 | 4 |
| 3.3 | The application must only intake .CSV files, no other data is to be read | 6.2, 6.4 (GUI) | 2 |