Software Testing Plan

11/7/2025

Capstone - WillowWatt

Team Members:
Ayla Tudor
Elliott Kinsley
Luke Bowen

Sponsor - OP Ravi, Director of Energy Transformation, Willow Inc.
Mentor - Jeevana Swaroop Kalapala

Table of Contents

Introduction:

Software testing is a crucial part of any good software development. Knowing this, our team has defined several test cases that we can utilize to make sure our model/testing platform is working properly, even in unstable environments. Our client, OP Ravi, has expressed certain conditions that he would like to be met in relation to the model that we created. He stated that if we could get around 85% accuracy for predicting the energy consumption in all of North Campus, that this would be sufficient for the purposes of this project. When our team first started with this project, we did not have any access to NAU's energy data, so we began thinking about how we can test a prototype inside Willow. We came up with the idea to pull energy data from a well known data website called Kaggle.com. From this platform, we found energy data that went back a few years and tested the random forest regression algorithm inside of our first model. We then were able to export this data into the ONNX package to be delivered to our client. This went off with great success, as our model was easily created and ported into the Willow platform. However, our client noticed that our model will plateau around the outlier data, he stated that there was a clear line that the model would stop following the data trend and sort of just stop forecasting.

After some testing, our team concluded that this must be a problem with how we are training the model, specifically what the data had to look like to accurately predict the next point. Noticing that our data had a clear maximum when defined in the limits of the Willow platform, we deduced that giving our model the correct range of data is imperative to a functioning model. With this in mind, one key feature that our system will have is to check the bounds of the incoming data and compare them with the data in Willow platform. Having a way to make sure that our data is properly shaped before creating the model will allow our team to prevent any production problems with our model. Since our current system already uses Willow's data, it is slightly redundant to have this check because it is being trained on the same platform data that it is working on. However, our team has decided to keep this implementation in place for any future production where we want to use other data sources to make our model.

There are many ways that a model can be skewed or mistrained, so our team has identified some of the other major points that can cause this to happen, such as a mismatch in input and output granularity parameters. For example, if the data that we are testing has a granularity of five minutes, that is, one energy consumption data point every five minutes, but the model is trained to only handle hourly data, our model will likely be inaccurate by however much the mismatch is. To combat this problem, our team would like to create a test in our model making process to make sure that our model can handle any kind of granularity, then run specific user tests, such as a static input/expected output to ensure that we are not taking this model in a direction that we do not want. Our team also has a third way to test the overall accuracy inside of the Willow platform, which has a defined formula for finding an accuracy (%).

Unit Testing:

One of the most crucial parts of testing our software for our capstone project WillowWatt is going to be the use of unit testing. Unit testing involves testing individual components or functions of a program in isolation, to make sure that each part of the program is running correctly. This is important because when just running your program all together, it can be hard to test whether or not each little component is working as intended, especially as your program grows in size and complexity. This usually happens earlier in the development cycle to find and isolate bugs, before you move onto deployment or system-level testing. Running these tests can help you ensure that the product you are pushing to deployment is more robust at every level, and improves the code maintainability and reliability. For our program specifically, the goals of unit testing are very straightforward: ensure all individual components of the model are working correctly, and that edge cases (such as missing data or invalid inputs) are handled gracefully. One of the most important steps is going to be validating that individual functions (such as the model predicting and evaluation) produce the expected outputs with known inputs.

Luckily, unit tests are extremely common, and because of this in addition to their level of importance, there are an abundance of tools to choose from to use for unit testing. Within Python specifically, there are many choices, but we are going to proceed with the immensely popular "unittest" framework as our primary tool for both writing and running unit tests. The unittest framework provides a very structured and straightforward approach to writing unit tests, as it has built in functionality for setting up test cases (TestCase classes), assertions (assertEqual, assertAlmostEqual, etc..), and more, such as automated test discovery. To run these unit tests, it will likely be done in the form of a command line prompt, and then integrated directly into the CI/CD pipeline for automated validation. We will then be utilizing test coverage as our main testing metric. This will ensure that the testing is covering critical components of our program, such as our data preprocessing modules, model training/testing and evaluation functions, and our prediction outputs.

When writing our unit tests, the list of “units” we will be focused on testing is not very long, since there aren’t many moving parts to our project. This is largely because our product has no frontend aspect since Willow has allowed us to integrate within their platform, so we are purely focusing on the model. The main performance metric we will be focused on evaluating is our model accuracy. This will be what verifies the main correctness of our program. The next main unit we will be testing is our logging functionality. This is crucial to get correct, as the final step of our program (billing period identification) relies heavily on our programs logs to be correct. The final main unit being tested will be our billing period module. This is what is responsible for identifying the peak loads that we are predicting, and comparing them against the peak loads recorded so far within the current billing period. Some metrics we will be keeping track of in addition to these units we are testing will be; test coverage, execution success rate, and edge case handling. See below for some definitions on these metrics:

- Test coverage: the percentage of code in our program that is executed during our tests.
- Execution success rate: the ratio of passed vs. failed tests when running our testing scripts.
- Edge case handling: the validation of our program’s robustness against weird/abnormal inputs (such as missing or incorrect data).

The next section is going to outline the detailed plan for our testing of each unit outlined above.

Unit Test #1 - Model Accuracy:

As outlined above, arguably the most important aspect to test of our program is the accuracy of our model's predictions. An important aspect to mention is that our client is looking for our model to be within +/- 15% of 100% accuracy with our predictions, meaning we could have a model accuracy of anywhere between 85%-115% (percents above 100% represent overshooting our prediction) to consider a "passing" value. We have multiple equivalence partitions for this unit, including valid structured data (including a fair mix of weekdays and weekends), high-variance and low-variance inputs (attempting to expose scaling and normalization bugs), and small datasets (minimum rows for our model to pass). In addition to this, we have identified multiple boundary values for this unit. The main boundary value is the accuracy range I mentioned above that our client requested, which is having our model +/- 15% accurate with its predictions. The next boundary value we have identified is the time coverage of our model (data only for this billing period or more and how granular our data can get). Some selected inputs that we have decided on are testing using multiple months data vs. one month, comparing granularities between 12hr intervals and 5min intervals, as well as testing with some erroneous inputs like energy values ridiculously high (millions of MegaWatts) or NaNs.

Unit Test #2 - Logging Functionality:

The next unit we will be testing is the logging functionality of our program. Similarly, we have already identified multiple equivalence partitions for this unit. These partitions include running our operational logs (with normal data), and running some form of warning or error logs to handle fatal errors in the logging process. We have also identified a few boundary values for this unit. These include the timestamp precision, and ensuring that all required fields are present in the log file. Some selected input examples that we have decided on are just writing out a standard log with typical data, and ensuring the expected output is written to the relevant log file, making sure it is not duplicating logs if they already exist, with an erroneous input example being an unwritable path passed to the logging function to see how it handles a permission denied error.

Unit Test #3 - Billing Period Identification Module:

Our last unit to test is our final module that identifies the current billing period, our predicted peaks for that period, and compares those against the already recorded peaks in that period to identify any future potential peaks worth addressing. Our equivalence partitions identified for this unit are using standard monthly periods (e.g. 1st-30th) vs. actual billing periods (e.g. 14th-13th), and having multiple daily peaks (ensuring the correct peak is chosen for comparison). Our boundary values for this unit include the start and end boundaries (starting and ending exactly at the correct timestamps), and any tie cases (predicted peak exactly matches recorded peak). Some examples of selected inputs we are using are regular data for a real billing period date sequence (e.g. 15th-14th) where the predicted is less than the recorded (no flag should be thrown by the program), a sequence using standard monthly periods where the predicted is more than the recorded, and some erroneous testing inputs like having the model predict an unreasonably high peak and being able to report that a model error is possible.

Overall we will be using unit tests on our program to ensure each individual crucial component is doing its job correctly. We will be using the unittest framework within Python since our existing program is written entirely in Python. We have an extensive plan already laid out to ensure that each part of our program is tested, including abnormal edge cases, to ensure our model is ready for deployment and system testing.

Integration Testing:

Integration testing is another crucial step in a software development team's road to creating quality code structures. Our team has come together with our client to recognize some of the ways that our system can fail, and how we are able to ensure that our model is able to deploy inside of the Willow platform. Our team was able to come up with 3 major tests that we can deploy on our system to ensure that everything ports over smoothly.

Our first main concern when it comes to problems with using our model inside the Willow platform is a misalignment of shape in our model versus the expected shape inside of Willow. For example, if we make our model have three different input parameters, such as energy consumption, forecast horizon, and granularity, but the Willow platform has the environment setup to pass in just one parameter like energy consumption, our system will fail and we will get an error that the models shape does not align. To combat this problem, our team will make a python test inside of our github to test how many input parameters a model has, then we will want to hardcode (at least to start) how many input parameters the Willow platform is expecting, if there is a misalignment, then the user will be reported of this misalignment so they can change how the model is being made.

Another testing feature that our system includes is the ability to test whether the model has been properly packaged into the ONNX compiled package, since this is the expected format

Willow wants. Lastly, to ensure that our model is able to work inside of the Willow platform, we will want to test if the model is working at all, by testing a range of its outputs and looking for any weird spikes that don't make sense that would interfere with integrating into the environment.

Integration Test #1 - Shape Alignment:

To test whether our model has the right amount of input parameters compared to the expected environment, we will want to be able to directly access information of how many input parameters are to what model, we can do this by using the feature "model.n_features_in_" attribute inside of SkiKit Learn, which will report back how many input features a model has. We will then use this number with a hardcoded expected output parameters number, if these two numbers do not align then we will report an issue with integration into the Willow platform. Furthermore, I would like this shape test to also make sure that the data types match up to Willows standards.

Integration Test #2 - Proper Packaging:

This test should initially check if our model is in the ONNX package, if it is not, it should report back to the user if there was a problem. From here, this test should basically be a stress test/corruption manager to see if we can break the newly made model. We will try to have outlier data that we can throw at it, if the model does not give us information, or gives us the exact opposite of what we are asking (negative drops where positive is expected), the user should be reported that it failed the stress test.

Integration Test #3 - input/output errors:

Lastly, as a final report metric, we compare our newly made model to a previously made model (if there is one to compare) to see how far off the deviation is and if it is any more accurate. We will want to also notify the user if any anomalies are detected from a certain threshold. This will allow us to understand how our model is changing throughout our optimization process. Furthermore, our team also wants to make sure that our entire pipeline is working from end to end. To do this we could monitor what the data spits out before using it inside of Willow, then use our model inside of Willow, afterwards sending the data back to the python program to see if there has been any issues. This may or may not make it into production, since getting more access to the Willow platform takes time and consideration.

Usability Testing:

Usability testing focuses on how easily and effectively users can interact with a system to access and understand its functionality and results. The goal is to make sure that the outputs we produce are clear, accurate, and easily understood, even without a full user interface. For WillowWatt, usability testing will help confirm that the results generated by our forecasting model make sense to those who will be viewing them, including our client and project mentor.

Since our system currently runs locally rather than being fully integrated with Willow's platform, our usability testing will center on how well our outputs communicate key information from the forecasting model. We want to make sure that users can easily interpret the data and understand what it means for energy usage across NAU's campus.

The focus will be on readability and clarity rather than visual design since our model outputs are displayed directly from our Python code and terminal environment.

Our main usability goals are to make sure that the data our model produces is both accurate and understandable. This means testing whether our forecasts, peak load predictions, and accuracy metrics can be read and interpreted correctly by the users who are viewing them. We will also look at how the results are formatted to ensure that the information being displayed, such as highest predicted days and accuracy scores, are easy to follow and understand. The testing process will involve having our client, OP Ravi, and our team members review the forecast outputs and provide feedback on whether the data is presented clearly and if any additional information or formatting changes would make interpretation easier.

Usability Test #1: Expert Review

The first usability test will involve a structured review with our client, OP Ravi. The purpose of this test is to gather expert feedback on how our results are displayed and whether the information presented aligns with what Willow expects from the forecasting data. During this phase, our team will run the forecasting model locally and walk through the outputs with the reviewer while noting any confusion, unclear formatting, or missing context we observe. We'll collect feedback and discuss it as a team so that we can then make improvements to the presentation and outputs to ensure they communicate our results effectively.

Usability Test #2: Team Review

The second usability test will focus on a combined review by our team and our faculty mentor. Each team member, along with our mentor, will independently run the forecasting model and record observations about how easy it is to interpret and verify the displayed results. Team members and our mentor will then share their analysis, highlighting any parts of the output that may be confusing, repetitive, or in need of better organization. Based on this feedback, we'll make revisions to improve consistency and readability. Including our mentor in this process adds an additional perspective and helps ensure that the model's outputs are understandable not only to our team members, but also to an outside reviewer familiar with the project.

Usability testing will begin once the model's primary forecasting features are complete and functioning as expected. We plan to run test forecasts using historical building data to verify that the results display correctly and that key findings are clear. After gathering feedback from our client and reviewing it as a team, we'll make adjustments to improve the presentation and usability of our product, such as adding supplemental summary printouts or improving how results are grouped and labeled. These refinements will help make our outputs more transparent and consistent.

Overall, our usability testing focuses on ensuring that the information that our system provides can be easily understood and trusted. By gathering feedback from both our client, team, and mentor, we'll confirm that our results are communicated effectively and that anyone reviewing the forecasts can clearly understand what the model is predicting and why it matters.

Conclusion:

This testing plan outlines the approach our team will use to ensure that the WillowWatt forecasting system is accurate, reliable, and ready to be integrated and utilized by Willow. Each level of testing plays an important role in verifying that our system performs as intended. Unit testing allows us to confirm that individual components, such as data processing and model training functions, operate correctly and produce consistent results. Integration testing ensures that these components work together seamlessly to create a smooth workflow from data input to forecast output. Finally, usability testing helps us confirm that the system's results are clear, interpretable, and meaningful to the people who will be reviewing them.

These layers of testing give our team confidence that the WillowWatt system not only functions correctly from a technical standpoint, but also communicates its results effectively and as intended. Since our model currently runs locally, our testing focuses on validating the accuracy of its forecasts and the clarity of the displayed outputs. By running tests using historical data, refining how results are presented, and gathering feedback from our client and team, we can ensure that the information produced by our model is both trustworthy and easy to understand.

Our team's ultimate goal is to deliver a forecasting tool that performs reliably and provides actionable insights into energy usage patterns across NAU's campus. By following this testing plan, we are ensuring that every aspect of our system from data processing to user interpretation has been thoughtfully evaluated and improved. The end result will be a system that is technically robust, user-friendly, and valuable in supporting Willow's and NAU's efforts to make campus operations more energy-efficient.