Machine learning to predict relevant support content based on historical user interactions

Final Report

Client: Workiva
Advisor: Neil Zhenqiang Gong
Team Members and Roles:
Erin Elsbernd: Communication Coordinator and Machine Learning Lead
Ram Luitel: Project Manager and Software Architect
Faizul Jasmi: Testing and AWS Tech Lead
Taizhong Huang: QA Lead
Christian Chiang: Webmaster and AWS Tech Lead
Khoa Bui: Webmaster and DB lead

http://sddec18-16.sd.ece.iastate.edu/
sddec18-16@iastate.edu

Table Of Content

Acknowledgement

We feel very thankful and fortunate to be assigned to this project which will help us develop our machine learning skills. The success and outcome of our project required a lot of guidance and assistance from many people and we are extremely privileged to have this input.

We respect and thank Mr. Alex Kharbush for developing this project and giving ISU students the chance to tackle the problem. We are also grateful for the resources and direction he provided throughout the two semesters. We would also like to recognize Dr. Neil Gong for being our faulty advisor and meeting biweekly to provide necessary support and guidance.

We heartily thank Dr. Thomas Daniels for constant encouragement, support and guidance which was helpful for us to successfully complete our project work. Also, we would like to extend our sincere esteems to all teaching assistants for their input.

1. Project Objective

a. Problem Statement

Workiva has an application called Wdesk. At the moment Wdesk users utilize a search engine to search for help articles when they are having a problem with the app. However, the search bar is not the most effective at listing specific help articles based on the needs of the user. Searching for a topic requires browsing through many different articles, and then browsing through the contents of each of those articles to hopefully arrive at the solution to the user's problem.

Workiva would like to automate the help article search process by tracking the user's actions while using Wdesk and predicting a help article based on these actions. Currently there are no automated tools that suggest relevant help articles to the Wdesk user. Therefore, if a user cannot troubleshoot an issue, they will often call a customer support number. With this current setup, if Workiva wants to expand their business, they will need to hire additional customer support staff to handle larger accounts. This is not a sustainable business model. Workiva would like to provide a better customer support experience with their Wdesk app, by using predictive models to help the user troubleshoot instead of humans. This will save lot of time and money.

b. Clients

Workiva is an enterprise software company based in Ames, Iowa. Founded in 2008 as Webfilings, its main product is Wdesk, a cloud-based enterprise management and auditing software-as-a-service platform that enables companies to create and file financial and compliance reports and documents to the SEC and other federal and state regulatory agencies.

c. Target users

Our end product is a model that will be used by developers at Workiva to integrate with the Wdesk app. However, the model will directly impact how Workiva customers use and interact with the Wdesk app.

2. Requirements specification

a. Functional requirements

- The recommendation model should be able to make recommendations for help articles and display the recommended article ID
- The model must be able to run on AWS.

b. Non-functional requirements

- The recommendation model should be optimized to run as quickly as possible.
- The recommendation model should be scalable with large and real-time data.
- The model should be written in Python.

3. Development process

a. Rationale

We used the Agile software development process. First, we worked with our client to gather preliminary requirements. Next we did research on recommendation algorithms and began the initial design of the project. After we started to develop, we had to change some of our approaches based on the data we received from the client or poor performance of a model. Working in an Agile environment allowed us to be more responsive to changing requirements and roadblocks. We used storyboarding on trello, and also used github for version control. Our test design diagram referenced in

section 7(a) reflects the iterative nature of our design and implementation process throughout the project.

4. Design plan

Our final system was the result of a long, iterative process. At its core, our system reads in data on user actions, processes this data into features, feeds these features to a random forest classifier, and finally, the classifier outputs a help article prediction. This entire system also runs on AWS.

a. Use-cases (tied to requirements)

Our project has a single use case: given a user action-event sequence, predict a relevant help article based on this action sequence. In order to support this use case, we had to create a system that could read in user data, process this data, create features from the data, use those feature to train a prediction model, and have the model predict help articles based on user actions with over 70% accuracy. We then had to deploy this system on AWS and ensure the entire system was written in Python to meet the requirements of our client.

The most important aspect of our project was the model to make help article recommendations. To find a model we implemented several machine learning and deep learning models and selected the best performing one for our final model to use in production. It quickly become clear that the random forest classifier was the best performing. It didn't take as much time to train as an SVM and also had greater baseline accuracy than any of the neural network approaches we tried. However, a model is only as good as the data it is provided with.To process our data and generate features we used the scikit-learn and nltk libraries. This was because we found much success in treating the data on user action events as text data, and the aforementioned python libraries made creating features from the user data less burdensome.

We used tf-idf matrices, bag of words and n-grams approaches to try to capture important action events and sequences of user interactions. We mapped each user event-action to an int, and then took all the sequences of events for each user and put them into a term frequency inverse document frequency matrix, or tf-idf matrix. This matrix stored data on important events like which actions were most frequent and which types of actions commonly occurred together.

Since we had over 170 help articles to predict and only 690 data points, we had to group the help articles try to increase the prediction accuracy of the models. This way instead of training a classifier to predict one of 170 help articles for a given data point, we would only need to train a classifier to predict one of approximately 20 subgroups. This helped our models perform better given the limited amount of data we received.

b. Modules

i. Dependency/Concurrency

We have several lambdas to do different tasks for the project. This essentially means the lambdas are interdependent on other lambdas output which means that some of the lambda are preconditions for others. Refer to Figure 1 in Architectural overview section for an illustrative view.

ii. Interfaces

Since we do not have any to build any front-end facing application for this project, we do not have any user interfaces(UI) to present. Also the interface that records the user activity data was provided by the client so we do not have any interface for the project.

iii. Architectural overview

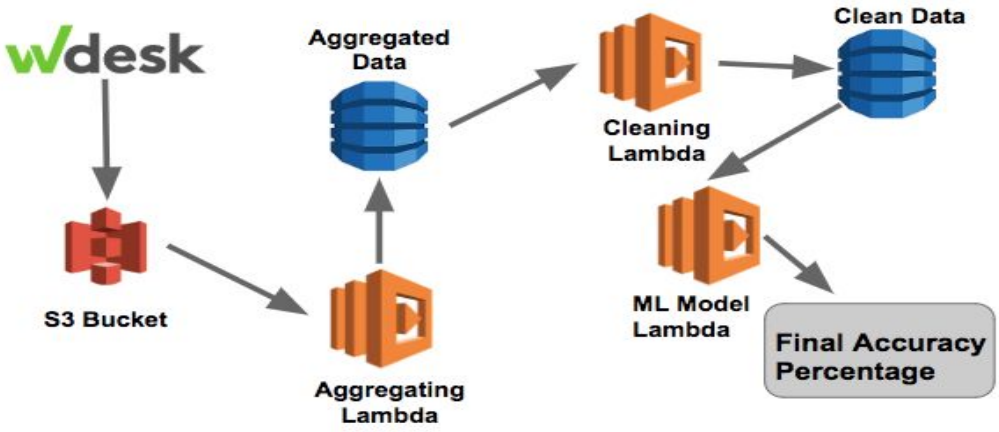

Figure 1: Architecture of project

The data will come straight from the WDesk application; it will come as raw data from the users, with many useless columns, and we will then store all those files in an AWS s3 bucket. Then, an Aggregating lambda will make all those files into one Dynamo Table for easier clean-up. Once the data is cleaned another lambda in charge of cleaning will parse through the aggregated table and select only the columns used by our models to be put into another table. Having a Dynamo Table will then allow the ML Model Lambda to consume it and produce the predictions based off the dataset.

c. Module constraints (tied to requirements)

Initially our team had little experience with machine learning and recommendation systems so we had to do a great amount of research and experimentation during the first semester. Having to learn and develop at the same time led to some unavoidable time constraints and delays in getting better prediction results. We also had to work closely with our client to ensure that our system was developed in a way where it could be eventually incorporated into the Wdesk app.

The strength of using models like random forests, SVMs, or neural networks is that they are auto trainable and environment independent. This is important since we wanted our end product to be part of a larger automated process that runs on an Amazon Web Service(AWS) environment. Since we are using supervised learning, we trained our models with labeled training examples, and then tested our models with labeled test data. This mean that we can keep and add to a large stored dataset in the cloud that we can use to update our models on AWS. As more labeled examples are added to the training data, we expect that the models will perform better over time as they are able to better recognize relationships between user actions and relevant help articles. Thus, one potential drawback of our project is that the success of our model is dependent on how much data we have access to to train our models. Overall, our model did not perform as well as we hoped, and this was partially due to have a limited amount of training data.

One of the drawbacks of using classification models is that performance decrease of the function has too many non-linear relationships. Similarly, the disadvantage of using the random forest model is that decision trees are disposed to overfitting since their bounds are not closed and the tree keeps on branching until we reach the endpoint. In other words, it is really difficult for us to come up with really good predicting model unless we try many different features in our models. To overcome this, we had to spend a lot of time tuning our random forest models and testing them many times on different subsets of the data to ensure the model was not overfitting.

Another potential drawback was using neural networks, and we eventually decided not to use them. Since we created our models ourselves and didn't rely on a machine learning as a service tools, we had to make sure we understood what was happening mathematically in the models. This way we could efficiently tune the models and make sure we were giving them informative features. Neural Networks are notoriously finicky when it comes to tuning and understanding them. Ultimately, we did not have great success on using neural networks for help article predictions.

5. Test Plan

The core of our project was a machine learning model and not a software application. That being said, we wrote tests for reading in data and assessing data quality. We also created test methods for gauging efficacy of features to give our classification

algorithm and for gauging the accuracy of predictions. We tested the models continuously throughout the process of development.

a. Unit testing (functional module)

We utilized unit tests throughout our project, most of these tests used built-in functions from machine learning and statistics libraries. We had basic unit tests for verifying the data we read from csv files. We also used visual and statistical testing methods to gauge the performance of our data features, model hyperparameters, and model performance.

- Test to make sure the data is not incomplete or mislabeled.
- Use F1 scores, ROC curves, and confusion matrices etc. to gauge the accuracy and prediction competency of our models. This involves testing to ensure a model has over 70% accuracy.
- Use cross-validation to test and tune features and model hyperparameters. For example, running different combinations of model parameters through many simulations to find the optimal parameter values to pass our random forest classifier: number of estimators, max depth of the tree, or the max number of leaf nodes.

b. Integrity testing (functional interface)

Data quality testing was critical for automation of data processing and feature generation. When starting out on this project we had to go through much trial-and-error to determine the best way to transform the data into a format that could interpreted by our classification algorithms. Having data quality checks helped to streamline the process of feature generation.

c. User-study (non-functional and functional)

The following list includes testing for non-functional requirements

- Performance: Test that the prediction model should be able to predict the helpful article within several seconds of the initial query.
- Scalability: Test that the model should eventually be able to process any size of data
- Extensibility: Test that the model should be able to compile and run on AWS.
- Usability: Acceptance testing by the client would check that the model we develop should be easy to understand and use for Workiva developers so that they can integrate it without any problems with their current application.

6. Implementation

a. Choice of languages, libraries

Our client required the machine learning models to be written in Python. Thus, we used several Python libraries to support our machine learning model and testing: scikit-learn, keras with tensorflow backend, matplotlib, and imblearn. We selected these libraries because they are the most widely used machine learning tools for python data analysis.

To process our data and generate features we used scikit-learn and nltk libraries as treated the user data as text data and time series data, depending on the type of model used. This way we were able to use bag of words and n-grams approaches to try to capture important action events and sequences of user interactions. For oversampling, we utilized the imblearn library. Finally, for visually testing and assessing model performance, we used matplotlib for graphs.

There are several other open-source tools for our project. Tensorflow is not the only machine learning framework available, but as its provided by google the documentation and ease of use attracted us to that tool. Keras makes interfacing with Tensorflow easier and we initially used Keras with Tensorflow when experimenting with using neural networks. However, in our final models we used scikit-learn to create our random forests. Scikit-learn also has great support online, as well as regular updates. This ensured that models could eventually be safely used in Workiva's application without fear of being downgraded or unsupported in the future.

Another big platform for machine learning are Google Cloud and AWS, they both offer services that will facilitate the cloud hosting of our models. They provide machine learning as a service tools for both data processing and actual machine learning models themselves. We have not looked into using any of their data tools yet, but we believe their automated data processing tools may be useful for us next semester when we want to get our model running on AWS and have the entire data ingestion, processing, and model prediction process streamlined. As for this term, our main focus is on creating models that will yield at least 70% of accuracy.

Regardless, we still used AWS to automate and facilitate some of our application microservices. DynamoDB was used to store the data provided from the WDesk application; this allowed us to access the most recent and most updated dataset available every time. The data comes as a bunch of "dirty" .csv files that are aggregated and cleaned using a Serverless lambda from AWS. This provided an effective way to sort and clean the data, allowing the models to consume the latest version of processed data.

b. Choice of development frameworks

For the machine learning algorithms, we developed primarily using Jupyter Notebooks and python ide like Visual Studio Code and PyCharm. The Notebooks were used because they make working with data extremely easy, and allowed us to add visual plots and charts to our code. This way we could trace how the data changed for feature processing, and also see how our models performed. We could also save these Notebooks and present them to our client to give an idea of our model performance and the types of features we were creating from the data.

In the cloud section we decided to use Amazon Web Services (AWS) instead of other services like Google Cloud or Microsoft Azure. All providers have similar services, but the main reason in choosing AWS over any of them is the familiarity the team has with working with it. AWS has proven to be the a reliable source and there is great documentation online. For the cleaning and whole architecture it was decided to use Chalice for our API Request and AWS lambda deployment framework. Chalice facilitates the creation of lambda and creates endpoint routes for all your different needs. This is the go-to framework the industry uses when creating an AWS serverless application.

c. Algorithmic choices

In terms of model selection, we settled on using random forests and neural networks mainly because their baseline prediction accuracy with our data and feature processing was higher than other models. Our client had initially suggested using LSTM neural networks, i.e. recurrent neural networks with long short-term memory units. However, the baseline prediction accuracy achieved from NNs was very low compared to that of random forest models. Thus, we decided to use random forests for our final model.

For feature generation we decided to treat the user action data as text data. To start preparing our data, we took a folder of over 1000 csv files on user behavior and condensed them into one dataframe that contained information on the user, the date, the actions the user took while using WDesk, and the help article they found useful based on this action sequence. The help article is the dependent variable we are trying to predict. The user actions each correspond to some string label as do the help article titles. For ease of working with data, we mapped all of these strings to a dictionary of ints. This process is contained in a python file on our github, but we also edited it so that we could eventually automate this data preparation task and put it on AWS. After the data preparation, we began creating features. By creating text features we went from 15% to over 50% accuracy with our random forest model. Thus we have been creating text features, then feeding them to a neural network or random forest model, testing the model and then looking at the breakdown of accuracies for each class label to tweak our features and model. For example, if we found that our model was very good at predicting a certain article that appeared a lot, we would go back

and create features that weight certain action sequences more for help articles that are predicted less often. Using confusion matrices and looking at the precision and recall values for each class/help article was an excellent guide for this process.

Grouping the help articles was another important strategy for increasing the accuracy of our predictions. We initially had over 170 help articles to predict, but with fewer than 700 data points to train with, our models were performing poorly. For this reason we decided to group the help articles together. This way our classifier would only need to predict approximately 20 help articles, instead of 170. Our final system had two classification rounds. First, given data on a user action-event sequence, a classifier would predict a larger help article group. Then, we would give this same user data to a second classifier to predict the specific help article. Overall, this strategy didn't always achieve over 70% accuracy, but did help us get closer to that benchmark.

d. Maintainability of software for future evolution

Since the early stage of the project, we have been prioritizing the maintainability and expandability of the software. This played a big role in our decision in choosing the technology that we wanted to use. Scikit learn is very popular and will be maintained for a long time. This allows long term support and expandability when needed. The software also used the most recent versions of Python which is Python 3.7.

e. Safety and security concern, if applicable

As like all other cloud computing AWS also comes with some security concerns. However, this part of work really depends on client once we deliver our code to them. As far as our project is concerned, we do not have security vulnerabilities. Just like AWS mantra about security " trust but verify". If our client applies this mantra when they integrate our prediction model with their existing system security protocols, then there should be no issue. The only concern we can say as of now is if incorrect data is fed to the models. Then the models might not predict helpful articles accurately because the data is useless or in an incorrect format.

f. Engineering tradeoffs

Due to limited budget, time and human resources, we filtered most of data that we think useless, such as, the type of computer, the type of browser and when users made those behaviours. Because we aren't fully utilizing all of the data features, it is likely our model only picks up on certain patterns of behaviours, which may be keeping the accuracy low. Due to the size of our training and testing data, the program is single threaded. This saved us a lot of time focusing on the prediction model.

7. Traceability

a. Requirements → Design → Implementation → Test plan

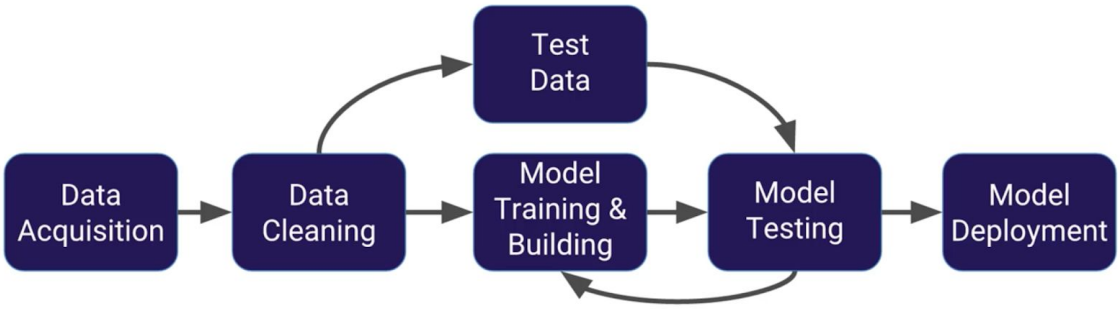

Figure 2: Testing Process Diagram

Figure 1 above illustrates our overall testing process, in addition to the overall development, i.e. traceability, of our project. The very first step is data collection which for us was provided by the client. Each time new data is received it must again be processed and cleaned. We created lambdas to clean the data. After the data is cleaned and processed into usable features, it will be used to train a model. The model training and building process is very iterative. We had to train the model and then test it to see how it performed. Using visual and statistical testing methods, we had to determine how to tweak the model. This could mean either changing the hyperparameters of the model, changing the features we gave the model (Test Data), or some combination of the two. Thus, this cycle would continuously repeat itself. When we were given new data from our client, we would start the process all over again at Data Acquisition. When we had a sufficiently performing model, we could then deploy it for use in a pseudo-production environment.

8. Testing, Validation and Verification

a. Test case evaluation (automated vs. manual, and coverage)

Because our core project is a machine learning model, we had to test it manually by actually feeding the model data and testing its performance using metrics like accuracy, precision, recall, confusion matrices, and ROC curves. There is no pass or

fail with such metrics, and so we generally were more concerned that the results got better over time. These metrics were also all contained within functions from the scikit-learn library. Therefore, once we knew we were feeding the functions the right information, we could guarantee that the tests were returning trustworthy information about our model performance. Generally, we gave these functions the prediction results from our model, and the actual results that should have been output. For example, we could give our model a sequence of user actions that should result in recommending Help Article #2, but our model predicts Help Article #3. Doing this many times can help us develop a sense of the model's performance. We were able to automate this process somewhat by giving models different parameters and outputting the results within a loop. But overall, we still had to go in and examine the results ourselves, and manually tune the models through trial and error as needed.

In terms of coverage, these metrics can't necessarily reveal specifically what is wrong with the model, but they can give you hints. For example, when we originally grouped the 170+ help articles into 20 groups, we found that some of the groups had values of 0.0 for precision and recall. This meant that our classifier was never even predicting these group IDs and hinted that our class distributions were unequal. One way we overcame this was through oversampling, reducing the number of classes for the classifier to predict, and ensuring the help articles within a class shared some common theme.

b. Validation and verification as appropriate for the project.

The process of developing our prediction models involved the following steps.

- In the future, user activity will be monitored and stored. This data will be processed in real time as the user of Wdesk uses the application. However, we were given and used a static dataset and didn't need to worry about real time data processing and storage.
- Our process takes in raw data and processes it as required by the prediction algorithm we use, i.e generate features.
- The processed data will be tested that it still holds value, i.e. when we map strings to ints, we test that this is actually the case.
- The generated features were tested for their value using cross-validation.
- The model was built and trained using training data.
- The model was tested using our test data. Based on the outcomes of the testing step, we often had to go back to the data processing phase and repeat until we got a higher accuracy with a given model. See functional and non-functional testing for more specifics on model testing.
- After we got the desired performance of our model, we deployed it for use in production environment.

9. Project management

a. Roles

Erin Elsbernd: Communication coordinator and Machine Learning Lead
Ram Luitel: Project Manager and Software Architect
Faizul Jasmi: Testing and AWS Tech Lead
Taizhong Huang: QA Lead
Christian Chiang: Webmaster and AWS Tech Lead
Khoa Bui: Webmaster and DB lead

b. Responsibilities

Erin Elsbernd: Communication coordinator with client and responsible for leading the ML models.
Ram Luitel: Manage the team and responsible for design any architect changes to the project.
Faizul Jasmi: Responsible for testing machine learning models and AWS architecture.
Taizhong Huang: Responsible for manual QA
Christian Chiang: Responsible for AWS and lead the team for necessary changes to technology stack and project architecture.
Khoa Bui: Responsible to project website and database related works

c. Timeline (projected vs. real)

We managed to follow our timeline overall. Our deliverables from February to May 2018 were done on time. August 2018 marked the start of the 2nd semester of the Senior Design Project. During that time, we discussed refining and training our model by getting more data from Workiva. However, we did not receive as much new data as we had hoped, and we also did not verify that new data wasn't duplicated data. This did not directly affect our timeline, but it affected the key result of the planned deliverable. Our final model did not have at least 70% accuracy. By October 2018, we started to port our models and data cleaning models to AWS to make the whole process an automated process.

d. Lessons learned

Overall, machine learning was a very new concept to most of the members on the team. It would have been better to have more members research and understand machine learning concepts so that we had more resources devoted to creating a successful model. It also would have been helpful in terms of having more people investigate the data we were given. We didn't fully utilize all the information provided in the datasets from our client. Having more individuals investigate the data for new insights may have helped us create better features to give our model

Another factor was having regular meetings. In the first semester, we met every week as a time in person, but during the second semester we did not meet as a group every week. Even though we had good communication over online chat, meeting in person helped keep everyone on task and on schedule. We may have been more productive if we had kept to weekly check-ins over video chat.