Use machine learning to predict relevant support content based on historical user interactions

DESIGN DOCUMENT

Sddec18 -16
Client: Workiva
Advisers: Neil Zhenqiang Gong
Team Members/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

Revised: April 19/Version 2

Table of Contents

1 Introduction 3

1.1 Acknowledgement 3

1.2 Problem and Project Statement 3

1.3 Operational Environment 3

1.4 Intended Users and Uses 3

1.5 Assumptions and Limitations 4

1.6 Expected End Product and Deliverables 4

2. Specifications and Analysis 4

2.1 Proposed Design 4

2.1.1 Functional Requirements 5

2.1.2 Non-Functional Requirements 5

2.2 Design Analysis 5

Testing and Implementation 6

3.1 Interface Specifications 6

3.2 Hardware and software 7

3.3 Functional Testing 7

3.4 Non-Functional Testing 8

3.5 Process 8

3.6 Results 9

3.6.1 Implementation Issues and Challenges 9

4 Closing Material 9

4.1 Conclusion 9

4.2 References 10

4.3 Appendices 10

List of figures/tables/symbols/definitions

Figure 1: design diagram

1 Introduction

1.1 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 requires 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 provides throughout the week. 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. Joseph Zambreno for constant encouragement, support and guidance which will be helpful to us to successfully complete our project work. Also, we would like to extend our sincere esteems to all teaching assistants for their input.

1.2 Problem and Project 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.

1.3 Operational Environment

This will not be a stand alone application and will be dependent on the virtual environment provided by Workiva. Our application will eventually be part of the larger Wdesk application that Workiva client's use. The end product of this project will need to run in the cloud to simplify infrastructure management, deploy more quickly, to ensure a lower cost, and give a real time solution. Our end product will be deployed in Amazon Web Service(AWS). As our application will be platform independent users can use it on any operating system where Wdesk runs.

1.4 Intended Users and Uses

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.

1.5 Assumptions and Limitations

The only cost factor for our project is Amazon Web Service licence which our client will provide to us. The assumption we have is that our end product should compile and run on AWS as this is one of the requirements that our client has. If our models have over 70% prediction accuracy than Workiva would like to use our best performing model. The limitations we have are technical skills among the team members. Only a few team members have machine learning experience through either classes or internships. We intend to mitigate this issue by setting aside research time for getting familiar with machine learning models and libraries. Some other limitations may include integration of our system with the Wdesk application. As Wdesk is a commercial application we might not have access to it when we are required to integrate our product with the full application. Testing limitations is another challenge as we can only test our model with the smaller datasets provided by our client.

1.6 Expected End Product and Deliverables

At the end of the next semester (December 2018), we expect to have a fully functional model that can predict helped articles and support users in problem-solving. We will try our best to increase our model's accuracy to the maximum. With higher accuracy, our model can assist users by providing most useful article for them to solve their problem, and save them time in finding a solution.

Another our goal is to implement our model on Wdesk at Workiva. To achieve this requirement, we must achieve 70% or higher on model's accuracy. We will keep collecting more data from our client to improve our models. We also need to have our model run on AWS and have an automated process of data cleaning, model updating, and article prediction. Ideally, we would like to build a system where the model could be updated weekly given new data.

For the final deliverable, we also need to deliver all project related documents that include design, code, project architecture documents and potential documents that include instructions on how to use our models.

2. Specifications and Analysis

2.1 Proposed Design

We are planning to implement several machine learning and deep learning models and select the best performing one for our final model to use in production. To implement our models we will be using scikit-learn and keras with a tensorflow backend. We selected these libraries because our client requested that we use python, and they are the best machine learning tools for python data analysis. Thus far, we have learned to implement basic machine learning and deep learning models on simple datasets. Given preliminary results on using user data, we think that using random forests and neural networks may be promising models to use with the Workiva data.

To process our data and generate features we plan to use scikit learn and nltk libraries as we plan to treat the data as text data or time series data, depending on the type of model we use. This way we can use bag of words and n-grams approaches to try to capture important action events and sequences of user interactions.

There are several open-source tools for our project. Tensorflow is not the only backend machine learning framework available, but as its provided by google the documentation and ease of use attracted us to that tool. Keras could perform the role of backend we assigned Tensorflow. They are

both great frameworks for machine learning. Scikit-learn is another great tool for basic machine learning models and we have been using it for both feature creation and model creation of random forests and SVMs.

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.

2.1.1 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.

2.1.2 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.
- The model should be easily readable by Workiva developers for future integration with WDesk app.

2.2 Design Analysis

To begin our design analysis we researched various supervised machine learning models. After input from our client and faculty advisor, we also researched neural networks and time series models because we are ultimately working with time series data. At the moment our team is subdivided into two sub-teams. One team will be creating machine learning models using random forests and Autoregressive Integrated Moving Average (ARIMA) models. The other team is experimenting with neural networks and markovian networks. The selection of these models came after research into supervised learning classification and working with time series data. Even though random forests are used for classification, we also wanted to try working with classic machine learning models and see how well they could perform against neural networks given good features. Given our preliminary results, we think random forests and LSTM neural networks may be a strong performer. LSTM networks can work well with time series data and text data. This means they will work well if we process our data like a time series model, or if we process our data in terms of text features.

For generating features, we plan to use text feature generation methods including bag of words and n-grams. With these features we won't necessarily need to use time series models but can still get good predictions. We decided to use text feature generation methods with our data so it could better capture important user action sequences. Using text features with random forests has taken our accuracy from around 15% to over 50%, so we believe generating text features are a promising avenue to follow with our final model.

The strength of using all of the models that we described above is they are auto trainable and environment independent. This is important since ultimately we want 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 can train our models with labeled training examples, and then test our models with labeled test data. This means 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 we add more labeled examples and feedback from our previous models to our training data, we expect that our 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. However, if we are able to build a good automated data processing and model updating system on AWS, then we can easily incorporate more data into our machine learning model, and models generally perform better given more 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 need 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 is not overfitting.

Another potential drawback is using neural networks. Since we are creating our models ourselves and not relying on a machine learning as a service tools, we need to make sure we understand what is happening in the models well so we can efficiently tune the models and make sure we are giving the model good features. Neural Networks are notoriously finicky when it comes to tuning and understanding them. We have already had some issues with understanding how LSTM works. To overcome this, we will need to put more research into neural network architectures and understand the basic math behind them. Overall, we still believe they will be a promising model for predicting help articles.

3 Testing and Implementation

3.1 Interface Specifications

We will use python for data processing, for creating, and for testing our models. Therefore, we will use PyCharm, Ipython Notebook or other Python IDE to develop our project.

We will use the following python libraries for model creation and testing: Scikit-learn, Keras with Tensorflow, Numpy, and Pandas.

We will mainly test if our machine learning models can predict the testing data correctly after being trained. We will select the best performing model as our final model by using a script to compare them.

So far, we have received our data from our client by emails. We will create a Python script to clean the data by using Ipython Notebook. Then, we will manually input our data into our different models to see what accuracy they get. If our final model gets over 70% accuracy, we will set it up on the AWS cloud and test. After that we will add a script to receive and clean data automatically.

Finally, our final model will only receive features of user's behaviours and return a prediction. If the prediction does not match, the model will use this case for training data.

3.2 Hardware and software

CSV: In computing, a comma-separated values (CSV) file stores tabular data (numbers and text) in plain text. We will use python to parse the data from CSV files given to us for testing.

We will test if our data is processed to specifications by using a Python script before analyzing it.

Scikit-learn: Scikit-learn (formerly scikits.learn) is a free software machine learning library for Python. It is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. We will use the open source code and models from Scikit-learn to create our models, e.g. 'RandomForestClassifier()'.

TensorFlow: TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and also used for machine learning applications such as neural networks. We will use it to build our deep learning models.

Keras: Keras is an open source neural network library written in Python. It is capable of running on top of TensorFlow. It is designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. We will use it to support our deep learning model for TensorFlow.

Pandas: Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. We will use Pandas to parse our data from CSV files.

Since they are all open source libraries, each version update may cause methods being changed. We will test if every method works as what we expect and develop in docker containers to avoid version conflicts.

Amazon AWS: Amazon Web Services (AWS) is a subsidiary of Amazon.com that provides on-demand cloud computing platforms to individuals, companies and governments, on a paid subscription basis. We will upload our final project to AWS cloud for scaling.

In conclusion, we will test if we input our data and use open source code correctly. We will test our models using scikit-learn features like ROC curves and confusion matrices. We will also test if our code can run and scale on AWS as well.

3.3 Functional Testing

Due to the nature of our project we will not be overly reliant on unit tests. We will use visual and statistical testing methods to gauge the performance of our data features, model hyperparameters, and model performance.

I. Test to make sure the data is not incomplete or mislabeled.
II. 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.

III. Use cross-validation to test and tune features and model hyperparameters.

3.4 Non-Functional Testing

The following list includes testing for non-functional requirements

I. Performance: Test that the prediction model should be able to predict the helpful article within several seconds of the initial query.
II. Scalability: Test that the model should eventually be able to process any size of data
III. Extensibility: Test that the model should be able to compile and run on AWS.
IV. 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.

3.5 Process

We will be testing the models throughout the process of development. We will use python for data processing, and for creating and testing our models. Therefore, we will use PyCharm, Ipython Notebook and other Python IDE to develop our project. Our test cases will then be able compile and execute in any of these environments.

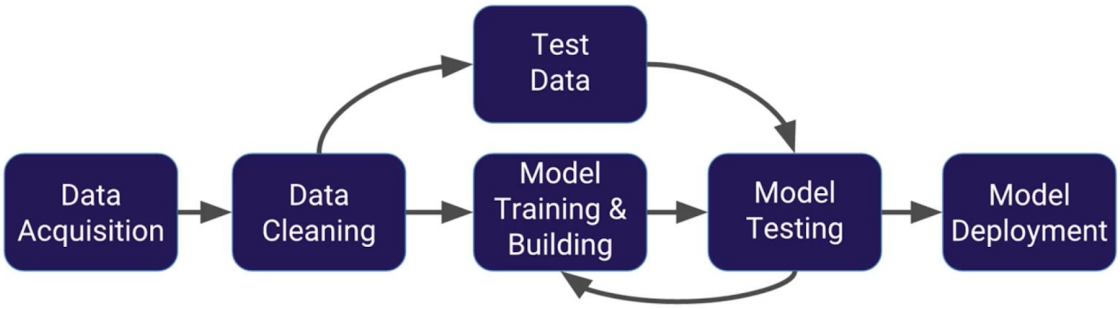


Figure 1: Design and Testing Process Diagram

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

- User activity will be monitored and stored. This data will be processed in real time as the user of Wdesk uses the application. However, at the moment we are using a static dataset and do not need to worry about real time data processing and storage.
- Our model will be using scikit-learn by taking in raw data and processing it as required by the prediction algorithm we want to use.
- Scikit learn and nltk libraries will be used to treat the data as text data or time series data, depending on the type of model we use to process our data and generate feature.
- Bag of words and n-grams approaches is used to try to capture important action events and sequences of user interactions via text features.

- The processed data will be tested that it still holds value, i.e. if we map strings to ints, we test that this is actually the case.
- The generated features will be tested for their value using cross-validation.
- The model will be built and trained using training data.
- The model will be tested using our testing data. Based on the outcomes of the testing step, we may have to go back to the data processing phase and repeat until we get over 70% accuracy with a given model. See function and non-functional testing for more specifics on model testing.
- After we get the desired performance of our model we can deploy it for use in production.

Once we have a working model, the next step will be to get it running on AWS. Ideally our client would like the prediction system to be able to update weekly, if not daily, given new data. To this end, we will need perform integration tests to ensure our model can run well on AWS. We will also need to develop an automated system for data preparation and cleaning to handle new data and feed it to our model so that our system can be retrained and updated. For this we will use Python Unit Tests to test our updating system. This will involve testing our data preparation methods, testing our data merging methods, and also testing the updated model.

3.6 Results

3.6.1 Implementation Issues and Challenges

During the initial development phase of our project, we have had some issues with merging code with different versions of python, scikit-learn, tensorflow, etc. We plan to overcome this problem through creating a Docker container and having everyone work within the same environment.

Another issue we potentially foresee is training models with a small amount data. At the moment, the total number of data points we have is 1200, and this may not be sufficient for a neural network model. We want to overcome this challenge by focusing heavily on creating good features from our data. We will also need to do more research into using neural networks with smaller datasets.

Our client would like our final model to scale well and run on AWS. Since we don't have experience with building models that can scale and update daily, this will pose a significant challenge to us as we begin this process in the second semester.

4 Closing Material

4.1 Conclusion

Thus far we have done significant research into model selection and feature generation for our data. We have also considered how we would test our models and features, and gauge their efficacy. To best approach the project, we are subdividing into two teams who will focus on specific models and test their performance. Given our current results, we do think that using text features and neural networks may be a promising area of focus given our discussions with our faculty advisor, client, and our own research into these topics.

We believe our design approach, as noted in our Design Diagram, is well-suited for the project. This will involve a circular process of data cleaning, feature generation, model selection, model

tuning, and model testing. We will continuously repeat this process until we can gain at least a 70% accuracy rate or higher on a model. This Design Diagram and approach was created after researching how Data Scientists and Machine Learning Engineers typically structure their data analysis projects.

We believe our testing approach, as noted in the Testing Process Diagram, is well-suited for the project. This will involve a circular process of data cleaning, feature generation, model selection, model tuning, and model testing. We will continuously repeat this process until we can gain at least a 70% accuracy rate or higher on a model. This testing approach was created after researching how Data Scientists and Machine Learning Engineers typically structure their data analysis projects.

To achieve such a goal, the plan is divided into 2 phases.

First Semester (Phase 1): In phase 1, we would first gather information and requirements from the client. The client will also give us data for us to work on and progress towards designing a model for machine learning. From thereafter, we would constantly do testing, analyzing, improving, parameter tuning and learning of the models that we have designed.

During this phase, the goal is to come up with the best possible model that generates the most accurate results.

Second Semester (Phase 2): Continue working on model refinement and tuning as needed. Get model to deploy on AWS and re-engineer it to be able to update weekly given new data. Ultimately, we will need to create an automated process on AWS for data ingestion, data cleaning, data processing, model training and tuning, and model prediction.

4.2 References

Amazon Web Services (aws) - Cloud Computing Services. https://aws.amazon.com/

Comma-separated Values. https://en.wikipedia.org/wiki/Comma-separated_values

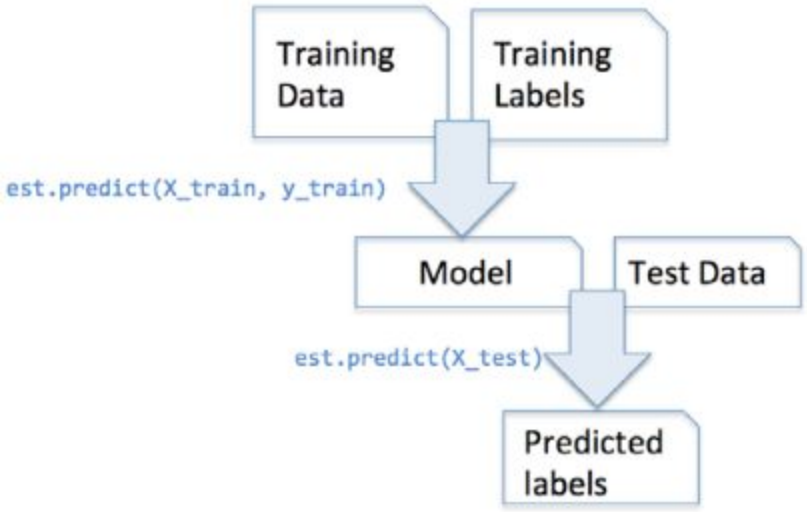
James, Gareth et al. *An Introduction To Statistical Learning: with Applications in R*. Springer Science+Business Media. 2013.

Python Data Analysis Library. https://pandas.pydata.org/

Scikit-learn: Machine Learning in Python - Scikit-learn 0.16.1 Documentation. http://scikit-learn.org/

Tensorflow. https://www.tensorflow.org/

4.3 Appendices



Raschka, Sebastian. “Python machine learning”