Analyzing Water Suitability for Aquaculture

Student Name: Visaj Nirav Shah **Student ID:** 201801016
Student Name: Aayush Desai **Student ID:** 201801060

Mentored by: Prof. Sanjay Srivastava, Ms. Nikita Joshi

I. Problem Statement

There are vast applications of Internet of Things and Machine Learning in the field of Aquaculture. One of the most critical factors in ensuring a healthy and productive livestock is the water suitability to sustain life. With this project, we aim to apply machine learning algorithms to data of parameters of water, mainly, Temperature, pH Level and Turbidity. We study these characteristics and develop a model to predict the suitability based on the data available.

II. Architecture

First the user enters the input on the website. User is asked for five data inputs: pH of the water (ph), temperature at 30cm depth (temp30), temperature at 60cm depth (temp60), turbidity at 30cm depth (turb30) and turbidity at 60cm depth (turb60). We have two pathways (two buttons) for processing a user's input. Let's examine the steps of each. Implementation of each is also given in the next section.

A. *The Dummy Button*

If user presses the Dummy Button:

1) First the input is sent to the backend through HTTP request . Then with the help of dummy data we predict the quality of water by using an already created KNN model. The data used here is static data. Which gives the fair result to the user.
2) After the prediction is done, the result is sent back to the website which displays it on the screen.

B. *The AWS Button*

If user presses the AWS Button:

1) First the input is sent to the backend through HTTP request. We first fetch the link of the dataset where the data is stored that we have created. Then we load the data with the help of the URL from the AWS dataset.
2) Then with the help of data retrieved we predict the quality of water by using KNN model. The data used here is dynamic data because the dataset is updated every time. A new KNN model with different Training and Testing set is created each time.
3) We then store the input data by the user along with the prediction in to the datastore. This helps in updating the real time data which improves the ML model in real time.
4) After we store the data, the result is sent back to the website which displays it on the screen.

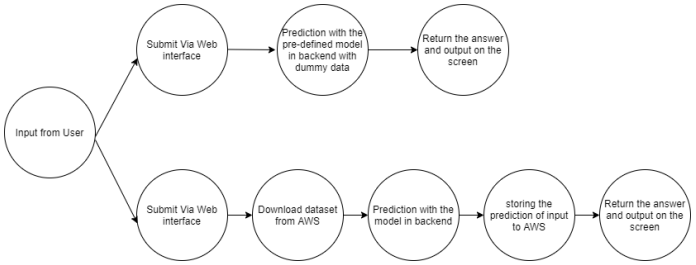

Fig. 1: Process Flow for Each Pathway/ Button

III. Implementation Details

A. *Selecting the Best Model*: *Notebook 1*

Before proceeding to the actual implementation, we need to determine which model is best suited to our dataset. For this, we first analyze the dataset and then process towards trying out the different models.

Dataset

Number of observations in dataset = 9623
Parameters columns = {ph, temp30, turb30, temp60, turb60}
Classification column = {class}

ph	Indicates the pH of sample
temp30	Indicates the temperature of sample at a depth of 30 cm
turb30	Indicates the turbidity of sample at a depth of 30 cm
temp60	Indicates the temperature of sample at a depth of 60 cm
turb60	Indicates the turbidity of sample at a depth of 60 cm
class	A binary variable with value 0 indicating Not Safe (NS) water and 1 indicating Safe (S) water

The entire coding was done using Python. Various libraries like *scikit-learn, Pandas, Numpy, Matplotlib* were used. For connecting with AWS and downloading the data from the same, Python modules like *AWSIoTPythonSDK and boto3* were used.

We split the dataset into Training and Testing sets using the 70-30 split rule, which means that 70% of the rows are used for training and 30% for testing.

This split was done using the scikit-learn module *from sklearn.model_selection import train_test_split* with *test_size = 0.3* (70-30 split rule). Plot of values of different parameters in Training set is given in the notebook.

Machine Learning Models

Logistic Regression: Implemented using *from sklearn.linear_model import LogisticRegression*. An object of *LogisticRegression()* was created to make the model.

Random Forest Classifier: Implemented using *from sklearn.ensemble import RandomForestClassifier*. An object of *RandomForestClassifier()* was created to make the model. 100 trees (n_estimators) were used in this model.

KNN - K Nearest Neighbors: Implemented using *from sklearn.neighbors import KNeighborsClassifier*. An object of *KNeighborsClassifier()* was created to make the model. k = 3 was used in this model.

How was the k for KNN selected? A range of k = [1, 25] was tried by fitting the data, and the accuracy for each k was determined. The value of k for which the accuracy was highest, in our case k = 3, was finally used to make the model.

The *.fit()* and *.predict()* methods were used for fitting the data and generating the predictions respectively. The metrics module of scikit-learn *from sklearn.metrics import mean_absolute_error, accuracy_score* was imported for calculating the Mean Absolute Error (MAE) and Accuracy for each of the models.

Once we had achieved the perfect accuracy and least possible MAE with KNN, we decided to use that model for the final implementation. Please note that the performance of each of the algorithms every time we run the model will be different since new Training and Testing sets are generated each time. This generation is a random process, and we have considered the output we got on our final run.

B. KNN Implementation for Dummy Button: Notebook 2

Considering the Training and Testing sets we got in Notebook 1 and based on the best performing model there, we create the KNN model for predicting the category for dummy data ('Dummy' button) i.e. if the user inputs a data here, it is not sent to AWS and the fixed pre-determined model is used for generating predictions.

The Training and Testing sets generated in the Notebook 1 are uploaded. The KNN model with k = 3 is made using the steps mentioned earlier. Once this model is trained, the user input is provided for generating a prediction for that single set of values of ph, temp30, turb30, temp60, turb60. The predicted value 0 or 1 is then displayed on the website as 'Water Not Safe' or 'Water is Safe' respectively.

How is the user input provided? The user inputs the values in the front-end website, which are then transferred to the Python notebook in the form of a JSON structure. This JSON is converted into a Python Dictionary (a data structure in Python) using *json.loads()*. The values from this Dictionary are then provided to the KNN model for predicting the values using the *.predict()* function as before.

C. AWS + KNN Implementation for AWS Button: Notebook 3

This is the actual implementation for real data that can be deployed. Each time the data is submitted using the 'AWS' button, the system gets the latest available dataset from AWS using the *boto3* client and *get_dataset_content()* function. This function returns the dataset URL which can be used for further processing.

The dataset from AWS is spilt similarly as done in Notebook 1. A KNN model with k = 3 is created and the Training sets are used to train the model. User's input is also taken in a similar manner as in Notebook 2.

A new aspect in this deployment is sending the input data to AWS. This helps us store the past data and use that to further enhance our model in future. The input data is sent to AWS datastore using channels and pipelines (as implemented in lab sessions). Next time, the updated dataset is used, this input also becomes a part in the Training/ Testing set, hence increasing available data and generating a better model. The input data along with the generated output is sent to AWS using the MQTT Protocol - *myMQTTClient.publish()*.

Finally, the result corresponding to the generated prediction is shown on the website.

Please note that we are updating our dataset every minute (the least time period available on AWS). Hence, once data is sent using the 'AWS Button', changes in the dataset are reflected only after one minute.

IV. RESULTS

Here, we discuss the performance of all the Machine Learning algorithms that were used, before the final one was selected. The performance is measured in terms of two metrics - Mean Absolute Error (MAE) and Accuracy.

Machine Learning Models' Performance

Machine Learning Model	Mean Absolute Error (MAE)	Accuracy
Logistic Regression	0.174922	0.825078
Random Forest Classifier	0.000346	0.999654
KNN - K Nearest Neighbors	0.0	1.0

From the above table, it is clear that KNN gives the best performance.

For further delving, please visit our Project GitHub repository: https://github.com/Aayush-Desai/iot

REFERENCES

1) Arif Istiaq; Akter Tasmima; Ahammed Md. Ferdous; Ali Md Younus Nahid, Abdullah-Al; Arafat. Ku-mwq: A dataset for monitoring water quality using digital sensors. 2020.
2) M. Ahmed, M. O. Rahaman, M. Rahman, and M. A.Kashem, "Analyzing the quality of water and predicting the suitability for fish farming based on iot in the context of bangladesh," 2019. [Online]. Available: https://ieeexplore.ieee.org/document/9068050.