Developing a Machine Model to Predict the Quality of Water Using Its Chemical Properties

Name: David Li Email: dwli@crimson.ua.edu

*Abstract*—This paper documents the process behind the creationg of a machine model developed in order to predict the pH of a body of water (pH being a proof-of-concept standin for any other notable measurement of water) using said body of water’s other chemical traits, such as conductivity or the amount of a certain compound within it. All steps involved in the training and deployment of a machine model are covered, including the formation and basis of its associated research question, the gathering, formatting, and processing of related data, the training of the machine model using AutoML tools, the validation of the model, the deployment of the model onto an easily accessible and usable web app, and finally a retrospective on the entire process.

Keywords—CS551, machine models, water chemistry, AutoML, Streamlit, Jupyter Notebook

# Introduction (*Heading 1*)

The importance of water as it pertains to the environment and everyday life needs no explanation. It is so important that entire government agencies and organizations are dedicated to the task of water conservation, collecting data about the various bodies of water within the US in order to assess the current state of the environment. I believe that training machine learning models with this collected data and deploying them into easily understandable and usable web applications will be able to help water conservation efforts. By using existing current and past data about bodies of water such as chemical composition, pollution levels, and the type and desnity of aquatic species living within them, a machine learning algorithm could theoretically predict future conditions of said bodies of water, possibly providing conservation agencies such as the United States Environmetal Protection Agency (EPA) useful knowledge on which are in the most critical condition and thus allow them to better allocate resources accordingly to protect them. Furthermore, it is also possible that machine models may reveal previously unknown correlations between the aforementioned properties of water bodies that theoretical research within the field of environmetal science can look into and expand upon.

# Research Questions

My research question, broadly speaking, asks what factors influence the quality of a body of water, and if one knows these factors, can they predict the future quality of it? For example, if an environmental scientist had access to a lake’s chemical composition, could they use the chemical composition of other lakes and the amount of fish in those lakes—as well as other factors such as lake size, depth, biome, etc.—to predict the amount of fish within the lake in question without even having to necessarily collect data on the subject directly. From this, could they also even go further and predict the future population of fish?

Originally, as a proof of the concept of using machine learning models to measure and predict water quality, I created a model that uses water chemistry data to predict the density of zooplankton, the idea being that since zooplankton are important elements of aquatic environments, it could represent one of many possible measurements of water quality—A higher density would correlate to a higher measure of water quality. Unfortunately, despite the fact that there did exist a decent amount of correlation between the water chemistry fields and the density of zooplankton (as shown in Figure 1), after cleaning, proccessing the data, and feeding it into the machine model, I was not able to achieve a very high accuracy for my final model (as shown in Figure 2), the problem being that there was simply not enough data to draw upon (the limiting factor here being data about zooplankton).

For the scope of this paper, instead of using zooplankton density I instead focused purely on water chemistry data—specifically, using pH as the target feature, as it both is an important factor when it comes to the quality of water and what type of life (if any) may flourish within it, as well as being a recognizable and understandable measurement to even people without much background in chemistry or biology. Compared to the density of zooplankton, which—due to lack of knowledge on the subject—I could only arbitrarily and without much scientific basis categorize (on top of logmathrimically normalizing the data, making the values even more abstract and not necessarily understandable), pH was a much more intuitive measurement to deal with (as will be explaining further later). Thus, I sought to find out—as part of the aforementioned broader question—the smaller scale question of whether or not one can use the chemical composition of a body of water to indirectly determine its pH level.

In a realistic scenario, being able to predict pH values based on other chemical characteristics is quite useless, as one can simply test for pH alongside testing for the other chemical elements, or possible even calculate it based the pH of specific elements within the water (although such an approach may not take into account non-element measurements within the dataset such as turbulence and conductivity). However, as I said earlier, the machine model which I deployed only serves as a proof of concept. Were there more usable information for other water components, I would have trained my model around them. However, as it stands, water chemistry has the most data pertaining to it available at least on the US Environmental Protection Agency’s (EPA) website [1], which is where I got my data from.

# Related Work

Admittedly, I am not the or even among the first to think of using machine learning models to help predict environmental conditions. For example, *A review of the application of machine learning in water quality evaluation* [2] discusses potential applications of using machine learning models on water measurements beyond even what I previously discussed as my intentions for initially pursuing the research question, such as using a machine model to show which features best contribute to determining water quality in order to know—using real-time monitoring—which aspects of water to focus on measuring.

Throughout my research, I have not found any instances of any environmental agencies using machine learning to its fullest extent, if at all. Most talk of artificial intelligence in the field of environmental science—such as the aforementioned research paper—are focused more on its potential and the theoretical side of their implementation rather than covering any specific scenarios which they have been used (however, this could be merely just the state of using machine models as a whole). AI within the world of conservation itself can be particularly controversial as although machine models have the ability to aid in conservation efforts (such as—for a very popular topic—in reducing carbon emissions), training them can also take up a large carbon footprint depending on the model and implementation. Thus, while creating my own machine learning model I was not able to draw upon much previous work on the subject in order to help my own efforts.

In terms of my own experience with the research question, I had none. As I stated in the previous section, the datasets and field I worked with in Homework 1 of the class were not reused for this final project, and furthermore I have little to no background in environmental science. All of my knowledge about how machine training works and how models are deployed was gained over the course of this class, with no prior experience.

# Methods

## Data Acquisition

All of the data used to train my machine model was gathered from the EPA’s official website [1]—more specifically, data from its National Aquatic Resource Surveys (NARS), which are surveys occurring every few years gathering data about various types of bodies of water in the United States. Many different types of surveys are available on the website, such as ones made for sediment composition and trophic status (albeit in a much more limited capacity than those for water chemistry, which is what I used). Even within surveys specifically for water chemistry, some of the surveys—older ones in particular—categorized their data differently or simply did not have enough features to be easily cross-referenced with the other, more recent surveys. Of the thirteen or so surveys (ranging from the years 2001-2019) I downloaded that had information on water composition, only six of them ended up being used. This number was narrowed down based on relevant data and similarity to the other surveys (the resulting batch only covering data from 2012-2019).

It is important to note that unlike my first attempt dealing with predicting zooplankton density, these surveys cover a variety of different types of bodies of water, such as lakes, wetlands, and so on. Although the type of body of water would have had to been disregarded during the machine-learning process despite possibly having a measurable effect on pH, I chose to use them all for the same machine model out of fear that once again I would not have enough data—Each survey had on average around 1300 datapoints, and since there were at most only two surveys per type of body of water that I deemed usable, there would have only been about 2500 datapoints for the model to draw upon if I only stuck to one type of body of water, which I felt would have been inadequate especially considering the amount of fields being dealt with. Were more usable data available for a specific body of water, I would have only focused one on type in order to get a more realistic result.

## Data Cleaning and Labeling

After downloading the data (which came in the form of .csv files), I renamed their default files to more readable names, as shown in Figure 3. In order to not only format and process the data but also train the machine model, I created a Jupyter Notebook file running Python 3.9, on which all code (aside from code relating to the web app development and coding process) was written. I read each of the tables into my Python code using the pandas library .read\_csv() function, creating six different dataframes. After this, for each one I used the filter() function to use only the columns which I felt were necessary for training the model or future data cleaning—that being the UID of each sample, and chemical result value.

While looking over the datasets during the acquisition phase above, I noticed that—seemingly regardless of year created or type of body of water being surveyed—the formats varied between either being what I labeled as “column” or “wide”-based, an example of each being shown in Figures 4 and 5 respectively. The column-based surveys had a dedicated column (labeled “ANALYTE”) listing the chemical/water characteristic measured per row, while the wide-based ones had a column per characteristic (which was more usable for a machine learning model to process). In order to convert the column-based dataframes into wide-based ones, I created a function called “processDfColumn” that filtered out the analyte column (alongside “UID” and the corresponding value) and used the pivot() function to turn the contents of the analyte column into their own separate rows.

For processing the wide-based dataframes, I created a another function called “processDfWide” that filtered out only the UID and the columns with the string “\_RESULT” in them to only fetch the chemical measurement results (as other extraneous columns containing information such as date acquired also existed), and after the fact removed the “\_RESULT” string to have the column names match those of the previously column-based datasets.

Rather than concatenate all of the datasets together immediately, I first counted how many null/missing values by creating a function called “checkForNulls”, which returned a table containing the sum and percentage of nulls per column (filtering out columns with no rows) by using the isnull() function. Through this, I was able to see that there were many chemicals/characteristics with little to no data about them recorded within each dataframe, such as TOC (total carbon) and NITRATE\_NITRITE\_N (amount of nitrate and nitrite). While it was a shame to exclude some of these fields as they might have been useful in predicting water quality (or more specifically in this case, pH), I ended up filtering any fields with more than 90% of their values missing within each table, the exception being NITRATE\_NITRITE\_N. Considering it is merely just the sum of the nitraite and nitrite columns, and considering some of the tables only had the NITRATE\_NITRITE\_N column and not ones for the specific compounds, for each table I created a column for or filled in the NITRATE\_NITRITE\_N using the nitrate or nitrite values and then got rid of the NITRATE\_N and NITRITE\_N columns instead.

After filtering out the mainly null columns, I then filled in the remaining null values for each table with the mean of the corresponding table. Alternatively, I could have also dropped any rows with any null values; however, this would have decreased the number of usable datapoints. Doing this method is not suitable for columns with mostly only null values as then the calculated might not be accurate, hence why I dropped them earlier. It was only after performing these processing steps which I combined the dataframes into a single one using the concat() function.

## Feature Engineering

At first, I had only one concatenated dataframe. I wanted the merged dataframe to automatically only keep the columns shared by all six of the original datasets—which is what the concat() function does by default. However, I saw that by doing this the of columns able to be used dropped from over twenty at most to only eight, with two of the datasets acting as limiting factors by only containing eight to ten unique columns (not counting the UID). Thus, I chose to move forward with two distinct dataframes—df1 containing data from all datasets but with only 8 features, and df2 containing data from only four of the six datasets but with 16 features.

Although having more features requires more datapoints to train a more accurate model, and having too many features—especially ones with little correlation to the predicted feature—can be problematic for a machine model to train, it is also true that having only a small amount of features is not optimal either, as there are less dimensions for the machine model to work with and tweak the values of in order to make the most accurate predictions. Not knowing if the tradeoff in this specific instance of several thousands of datapoints for twice the amount of features was worth it, every operation I performed henceforth until after the training stage was done twice, one for each concatenated dataset.

After this, I also used a combination of the pandas dataframes corr() function and the seaborn library mapping tools to create several graphs in order to plot the correlation of the given features, such as the pairplot shown in Figure 5. This was done to make sure that the features of the dataset did in fact have a suitable-enough amount of correlation to contribute to the prediction of the chosen feature. Initially when starting the project, I was partially unsure of which feature to actually choose to predict, and so having these visual elements helped me eventually decide on pH.

With pH chosen as the targe feature, I then began the process of processing it first by plotting it out on a stripplot to see if it needed any transformations or normalizations. Most of the values centered around the 7-9 range in what I considered to be a normal-enough distribution, meaning I did not have to transform or normalize the feature. I did group the pH values from continuous values into categorical bins in order to achieve a higher accuracy at the cost of the predictions being less detailed. From Homework 3 and my previous attempt with zooplankton densities, I saw that having five or more bins led to not very accurate results (depending on number of features and datapoints, of course). Thus, using the pandas cut function I grouped the values into three bins, with 0 indicating acidic water (from a pH of 0-6), 1 indicating neutral (pH of 6-8), and 2 indicating alkaline/basic (pH of 8-14).

## Model Development

Several different machine models are available to use via the sklearn Python library. Developing them is as simple as importing the desired one, setting it up with any desired parameters, splitting up the data into training and testing sets (for the model to be trained on and the for the result to be tested against for accuracy, as will be discussed more in detail in a later section), and fitting the data into the model using the fit() function. Notable types of models provided in sklearn include support vector machines (SVMs), decision trees, linear regression models, and neural networks.

SVMs classify data into different categories in an n-dimensional space using hyperplanes, such that a single hyperplane will separate the datapoints into two classes depending on which side of it they are on [3]. Linear regression treats the relationship between the input and target features as linear, with an assigned coefficient to each linear relationship determining its influence on the resulting prediction [4]. Decision trees consist of several different nodes that sort given data by starting it at a root node and sending them down the “branches” of the tree, splitting paths depending on different qualifying test cases at each split [5]. Finally, neural networks also consists of several different nodes, although unlike decision trees input is given to multiple nodes at the input layer, and depending on whether the given values (after having some mathetmical operations performed on them with each other) reach a certain threshold the corresponding node will fire and send the output to the next layer for processing [6]. Each type of model varies in application and performance, and have their own submodels, such as having a classification model for categorical data and regressor model for continuous data.

Although I ultimately did not choose a specific model to use manually, I did test the two dataframes using a few provided in the sklearn library just to get an idea of the results before moving into the AutoML stage covered in the next session. I did this by creating a function called “skManualModelTrain” which tries to fit the given data into four arbitrarily-selected sklearn models—one from each of the four categories discussed above. For a SVM, I chose LinearSVC, which is similar to the standard C-supported vector classification (SVC) model provided by sklearn but performs better on larger models. For a neural network I chose MLPClassifier. Multilayer perceptron (MLP) neural networks use hidden layers between the input and output ones and have the ability to be used nonlinear models [7]. As shown in Figure 6, the accuracy of each varied significantly between each other, and even between the two datasets as I will go into mnore detail in the section “Model Evaluation and Validation”.

## Model Training

Rather than choose what model to use manually, I decided to train my machine model using AutoML tools. AutoML refers to the automated and iterative process of—among other functionalities—finding the most optimal model and hyperparameters to use for a given dataset. In terms of choosing the best model, this removes the need for data sciences to do what I did for demonstration purposes in the previous section, which was run data manually through several different models in order to find the one with the best accuracy. While there does exist folklore on what values for the number of epochs, batch size, and other hyperparameters for certain types of data are the most optimal, they are not necessarily always accurate and thus having AutoML tools to find hyperparameter values as well is also convenient.

For this project, I chose to use the tpot Python library for the AutoML process. This involved creating a TPOTClassifier object and setting its given parameters, such as the maximum evaluation time for each pipeline tested and the number of generations to be done. The main drawback of using AutoML tools is that they are computationally intensive and can take a long time to perform. To mitigate this fact during the training process, I ended up setting the number of generations for the two TPOTClassifier objects to one, meaning they would only go through one generation of optimization (alternatively I could have stopped the process as TPOTClassifier objects will give the most optimal result not just after finishing training but also when the program is halted). From experimentation, I found that each generation on the datasets took around 10-15 minutes, meaning that with five or so generations for two datasets I would have to wait two hours for what could ultimately turn out to be suboptimal results. By performing only one generation at a time, I could more frequently see the results and adjust the other TPOTClassifier objects parameter values. Only after finding the best values did I then run the two datasets through five generations each.

Mutation and crossover rates are important values to set for the AutoML process, representing the percentage of pipelines to apply random changes to and the percentage of piplines to mix characteristics between respectively [8]. The two almost act as opposites, with the mutation rate introducing an element of randomness that prevents stagnation or unwanted convergence in the piplines, and the crossover rate increasing homogenity between piplines and thus increasing convergence. Mutation rates tend to be set low (<0.05) as otherwise the AutoML process becomes too random, while convergence rates are set in the range of 0.5-1 [9]. These were the main parameters for the TPOTClassifiers that I tinkered around with. Despite trying several different pairs of values from (0.9, 0.1) to (0.05, 0.95), there seemed to be no significant difference between different set values for mutation and crossover rate. For my final trained model, I thus stuck with a mutation rate of 0.05. and a crossover rate of 0.75.

## Model Evaluation and Validation

For the manually-picked example models, I tested their accuracy by using the .score() function, printing out separately training and testing scores as seen in Figure 6. Although I did not end up choosing the model to use manually, looking at these preliminary accuracy scores still provides much valuable insight. From them, it is apparent that df2—the dataframe with less datapoints but more fields—outperformed df1 in most situations, the only exception being in Linear SVC, which is quite unusual considering SVM models tend to perform better on datasets with a large amount of features but relatively low datapoints [10]. df1’s lack of input features lead to the Logistic Regression model performing especially poorly most likely due to—as I hypothesized earlier—there not being enough fields to tweak the predicted output with. Another interesting observation is that in both scenarios the Decision Tree model had a 100% accuracy score, which is not too surprising considering they have a tendency to overfit data [11].

During the AutoML training process, accuracy scores are automatically printed out for each optimal pipeline, as shown in Figure 7. Despite the fact that df1 did perform noticeably worse in the manually-picked models, I still trained it using AutoML in the off chance the TPOTClassifier would have found a more optimal model I missed. Unfortunately, from Figure 7, it is apparent that this was not the case. The final accuracy for df1 was 86%, and only around 89% for df2, with df1 using a GradientBoostingClassifier and df2 using an XGBClassifier—both of which I had not tested during the manual model picking. As I stated earlier tweaking any of the parameters for the TPOTClassifiers did not make a noticeable difference on the results, meaning 86% and 89% were the most I was able to achieve. Going forward in assessing accuracy and deploying my model, I decided to only stick with df2 as it was the dataframe with the higher accuracy. Whether or not these low scores were due to a lack of correlation between pH and the input features of the dataset, a lack of datapoints, or some other reason I am unsure.

## Model Deployment

After having trained and evaluated my model, I saved the AutoML result for df2 using the .export() function, which created a specified file called “pipeline.py” in my project directory showing the Python code—including imported libraries and function parameters—for the XGBClassifier model, as shown in Figure 8. After this, I copied the code into my Jupyter notebook file in order to fit it to my data and save it to a .joblib file, which allows functions on the model to be called by programs using it without having to refit it.

I ended up deploying my model using Streamlit, which not allow allows data scientists to create apps through the streamlit Python library, but also allows them to host them for free. The main purpose of my web app is to allow users to upload their own data for my model to compare its predicted values with those of the uploaded data and subsequently generate an accuracy score; however, I implemented several other features both practical and for vanity, such as the ability for users to control input features via sliders to test single chemistry value combinations (the results being shown real time in a confidence-level bar graph as shown in Figure 9), or the ability to read information about each input feature should they be uninitiated on its identity or nature (as of writing this report, this feature is technically completed as not all information has been written—I am not a chemist and probably do not have the knowledge to accurately provide such information regardless; for those who really want to learn about them, alternatively Wikipedia excerpts for most are available however).

Files are uploaded through a streamlit file\_uploader element, and processed in a way not too dissimilar to what I did in order to process the data used to train my machine model in my Jupyter notebook. After a .csv or .xlsx file is uploaded, it is converted to a pandas dataframe and then processed over the course of several different steps. I wanted to make my peers’ experience testing the model in the future during the later evaluation phase as seemless and easy as possible, and thus the web app does most of the dataset processing for users (at the cost of running a bit slower). This includes culling any unused columns, replacing nulls with mean values, and splitting the pH values into bins. While my peers did not end up using this function, I also implemented the ability for users to map column names in their datasets to those matching the ones used by model (for example, mapping “Ca” to “CALCIUM”). After uploading a file, if the given dataset has its own included pH values, an accuracy score will be generated. The web app uses the trained machine model by opening the .joblib file discussed earlier in order to make predictions on user-uploaded data, and then compares it with the pH values of the user-uploaded data if present in order to find this score. Users will also be able to download the results and the processed version of their own file run through my model to generate the accuracy value.

I tested this app by running the command “py -3.9 -m streamlit run app.py” in the project folder, which hosted the Python file on my local machine. After thoroughly testing my app’s main functionality, I created a new Git repository and pushed all of the project’s files onto it. By clicking on the sidebar on the app as it ran on my local machine, I was able to access the option to deploy it via Streamlit directly, as shown in Figure 10. All I had to do was link my Google account to Streamlit, fix a few dependency issues by updating my project’s “requirements.txt” file, and wait for a few minutes in order to deploy the app.

# Results

## Mode Performance in Evaluation

Just before exporting and deploying my model, I also assessed the accuracy of the final df2 model on a testing set of data. As shown in Figure 11, for both the manually-picked models and for the AutoML process I did not train the models using all of the data available in the dataset. Instead, I split the data up into two distinct training and testing sets using the train\_test\_split() function from the sklearn library. The test\_size paramater (in all cases, I used 0.2) represents the percent of the total dataset that will be chosen to be part of the testing set, while the rest (80%) will be set as the training set. The random\_state parameter takes an integer input acting as the seed for how the dataset will be split up—It was important for me to keep it at the same arbitrary number every time I split the dataset up so all manual models and the AutoML process would be training on the training set for a fairer comparison.

The validation process involves running the testing set through the trained/fitted machine model in order to assess the accuracy. Having two distinct sets is important as sometimes the accuracy in the training set may not accurately reflect its accuracy in a real world scenario, such as with the Decision Tree Classifier and how it had a 100% accuracy on the training set but only 86% for the testing set on df2, as shown in Figure 6. This can be done simply by using the score() function on both sets for sklearn models. For validating the AutoML TPOTClassifier object’s chosen optimal pipeline, a score() function does exist in the tpot library as well. Alternatively, one can also use the predict() function to generate predicted values on the testing input features and compare the predicted results to the actual testing target feature values the using the sklearn accuracy\_score() function. I evaluated my model through this method, and furthermore plotted the details of its performance using an sklearn confusion matrix, as shown in Figure 12. With an accuracy score of 90%, the optimal XGBClassifier pipeline appeared to perform just about the same on the testing set as it did on the training set. Based on Figure 12, it had more of a tendency to overpredict the pH range than underpredict, although the difference is not that noticeable.

## Feature Importance

Similar to the properties of water in real life, some of the input features of my final model contribute to the target pH value more than others. I was able to easily and visually measure how much each feature affected the results by using the sliders I had implemented during the app deloyment and creation process discussed earlier.

Each of the measured elements and compounds aside from ammonia (sulfate, potassium, nitrogen and nitrate/nitrite, calcium, magnesium, sodium, and phoshorus) increased the predicted pH level, although only up to a certain point before their affect was capped (without necessarily even making the model near 100% certain of the water being alkaline). When lowering their amounts below their mean values, the pH lowered by noticeable amounts and to a greater degree compared to raising them. With the exception of sulfate, this is peculiar to note as must of them—such as sodium—are neutral in pH and thus one would expect the pH to become more neutral (trending towards seven) than alkaline (increasing); however, such a discepretency would require more scientific knowledge to verify. Sulfate and ammonia notably perform the most unexpectedly, as despite being acidic [12] sulfate tended to increase the alklanity, and despite being basic [13] ammonia tended to increase the acidity. Non-elemental measurements varied in their effect, with conductivity notable having little effect, and acid-neutralizing capacity having arguably the greatest affect of any of the features. The latter’s propensity to lower the acidity when lowered aligns scientifically, as it represents the ability for a body of water to resist becoming acidic [14].

It is important to note that the trends listed above are only broad generalizations. Aside from linear regression and other similar models (which my model was not), target feature prediction is not always a question of assigning constant weights to each target feature. As I stated earlier, the affect of each one on the result tended to cap off well before the max value. In some cases—such as with sulfate—they furthermore did not have a straight positive or negative correlation with the input feature, but tended to alternate increasing and decreasing it every so often.

## Runtime Monitoring

As I had no classmates with the scientific knowledge or access to equipment needed to measure the chemical composition of water, there was no way for any of my peers to create their own datasets to test my deployed model. The two classmates who helped me in testing instead found water chemistry data from different sources other than the EPA website. One of them eventually found data from the National Park Service Datastore website [15] regarding water chemistry in Alaskan streams, while the other found data about downstream water chemistry from an online dataset library [16].

One of the biggest issues with testing my model using datasets from other websites is that they often did not have all of the features my model required. Although this was by far not a perfect solution, I mediated this by having my web app automatically fill in missing columns with the mean values from the EPA datasets alongside all of the other processing steps, as shown in Figure 13. As stated earlier, this and several other features were implemented in my web app to make it as easy as possible for my peers to test. For the two datasets tested by my classmates, all they needed to was rename some of the columns to match the feature names used by my machine model.

The results of my peers’ testing can be seen in Figures 14 and 15, respectively being the results for the Alaskan streams and downstream water chemistry tables respectively. Although my model did have a 90% accuracy when testing within the EPA datasets as shown in Figure 12, attempting to predict pH based on data from other sources revealed that, in the end, my model was not very sound. The degree to which my model’s inherent training compared to the lack of certain features within the datasets my peers chose affected the level of accuracy is uncertain, however. Although the model’s predictions for the downstream water chemistry dataset were only barely better than random guessing, it is important to note that that dataset did not provide—among other features—the feature for acid neutralizing capacity, which as I discussed earlier was perhaps the most important one. Generalizing such a column to only singular mean values is bound to lead to inaccurate results. Comparatively, the Alaskan streams dataset did include acid neutralizing capacity values and thus—while it did not perform too stellar still—was a lot more accurate.

# Discussion

## Answers to Research Questions

Unfortunately, due to how inaccurate my final model turned out despite my efforts, I was not able to sufficiently answer the research question I had posed. Even if a water chemist had the available chemical data for a body of water and wanted to find the pH value based on it using my model, the prediction would not necessarily be accurate. The model not only is inaccurate when it comes to making predictions, but also the manner in which its input features contribute to predicted result are—at least from my preliminary analysis—not necessarily accurate to scientific reality.

Even if my model had been 100% accurate, the nature of my research question is still far too broad to thoroughly answer based on the success of a single student-made model. As I stated earlier in this paper, the target feature of pH is just a placeholder for any other important feature about a body of water—more steps would have to be done to further investigate the question. It is due to this that I can optimistically say that, while it is so that a single machine model cannot answer the question, it can also be said that the inability of a single machine model to make accurate predictions does not outright invalidate the question. Perhaps it is simply so that—at least based on my input features, which were procedurally chosen not out of any scientific basis but merely out of availability—pH genuinely is impossible to predict, while other important features such as temperature, oxygen content, or organic density would be.

Overall, despite leading to failure, I still consider my research to be a fruitful experience. I was able to learn much about training and deploying machine learning models in much further depth than previously explored in class (as I will discuss in a later section), and in some cases my machine model did display some accurate behaviour, such as being heavily affected by the acid neutralizing capacity of the water. This suggests that perhaps had I had access to more data and more time in my academic schedule to dedicate to this pursuit, I would have been able to achieve a more accurate model in the end.

## Limitations of your research.

As it stands, there are many different aspects to this research question which I understand I alone and in a limited time frame was not able to answer or even take into account. There are several reasons for this, one of them being that when it comes to natural sciences in general there are several different factors one must take into account, possibly even beyond the initial scope of the question. As I stated earlier, there are many other factors beside pH level that need to be taken into account when it comes to assessing the overall quality of a body of water. Furthermore, these factors may require even more input data about their body of water in order to fully assess. For example, if instead of pH one were to try and predict the level of pollution in water, such a target feature may not best be viewed in a vacuum of its chemical components, but rather under the contexts of the surrounding cities and population centers around it which may have contributed to said pollution. For an environmental scientist, gaining access or personally measuring such a factor would be hard to accomplish, which leads into another reason answering the question may be difficult, that being a lack of data.

Considering how many rivers, lakes, and so on exist in the US and how they exist all around the map (not to mention all of the lab testing required to process water samples), gathering comprehensive data on the US’s water sources is difficult, and so it is unfortunate but understandable that the amount of data that currently exists may be sparse compared to other fields. For example, the EPA website which I got the data used to train my machine learning model only took measurements on each type of body of water every three to five years rather than, say, monthly ot even annual reports. Due to me being merely a student outside of the field of environmental science, I had to rely on this data from these and other similar reports solely, whereas professionals in the field would have access to more comprehensive, non-public data as well as the tools to more frequently and automatically monitor chemical values.

The final largest limiter to my research was just the amount of time required to gather data and train machine models in the first place. Had I had a more powerful machine, much time could have notably been saved when it came to the AutoML process, which as I stated earlier took around two hours to complete five generations of for both tested datasets. Further experience beyond a single academic course in the field of data science could have also helped me avoid mistakes and make decisions the best answers to which were only viewable in hindsight. For example, if I had known that the ratio of features to datapoints in df2 was better suited for a machine model to learn from than df1, I could have saved time by simply not performing any preprocessing or training on df1, dropping it from the beginning. In terms of the accuracy of the model, had I known that pH would have performed so poorly in the first place, I would have probably created an automated function to test the performance of all features as target features rather than stick to it solely to the point it was too late to turn back after I found out how suboptimal predictions on it were.

## Lessons Learned

While I did learn much about the machine learning process in class and by doing the homeworks, I found there was a large difference between simply running and tweaking values in an already existing project with pre-gathered data than training a machine model from scratch. Training and deploying my model taught me the significance of the data gathering process when it comes to data science, and in trying to create a machine model specifically tailoured around my gathered data I encountered many problems and obstacles unique to my dataset and research question that, in eventually overcoming, gave me greater insight into the training process that will no doubt come in hand for any future data science pursuits of mine, whether it be academic or professional.

During this project, I learned about several different types of machine models available on the sklearn website by using them on my datasets, examining the results, and performing further research in order to determine their differences to see which would possibly be the best for my model. Furthermore, in creating my Streamlit web app I also learned much about the streamlit library and how to host a web app using something other than Heroku.

When it came to validating the results of my machine model and having my peers test it, I learned from hindsight many possible areas I could improve on when it comes to training machine models and which I should have spent more time in. As I discussed earlier, there were several limitations to my research which I now know from experience how to circumvent or at the very least keep in mind in order to migitate their effect on the machine model training process.

##### Acknowledgment

I would like to acknowledge Jacob Caudill and Jonah Sussman—two other computer science graduate students attending the University of Alabama and taking the CS551 data science course this paper was written for—for spending the time to look up and format datasets to be tested by my machine model web app, as well as simply making sure the web app worked correctly on other computers. Their work was instrumental in helping me validate my model

##### References

1. “National Lakes Assessment,” *EPA*, 07-Sep-2022. [Online]. Available: https://www.epa.gov/national-aquatic-resource-surveys/nla. [Accessed: 01-Dec-2022].
2. M. Zhu, J. Wang, X. Yang, Y. Zhang, L. Zhang, H. Ren, B. Wu, and L. Ye, “A review of the application of machine learning in water quality evaluation,” *Elsevier Enhanced Reader*, Jun-2022. [Online]. Available: https://reader.elsevier.com/reader/sd/pii/S2772985022000163?token=3584440E9D78ACD58B6024ADABCB9670F48ED47C7E39590C4C022CF6BF09E50057158058844E69287125FE32E8B61E10&originRegion=us-east-1&originCreation=20221202033141. [Accessed: 01-Dec-2022].
3. R. Gandhi, “Support Vector Machine - introduction to machine learning algorithms,” *Medium*, 05-Jul-2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. [Accessed: 01-Dec-2022].
4. “Linear regression vs logistic regression - javatpoint,” *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/linear-regression-vs-logistic-regression-in-machine-learning#:~:text=The%20output%20for%20Linear%20Regression%20must%20be%20a%20continuous%20value,independent%20variable%20must%20be%20linear. [Accessed: 01-Dec-2022].
5. N. S. Chauhan, “Decision tree algorithm, explained,” *KDnuggets*, 09-Feb-2022. [Online]. Available: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html. [Accessed: 01-Dec-2022].
6. IBM Cloud Education, “What are neural networks?,” *IBM*, 17-Aug-2020. [Online]. Available: https://www.ibm.com/cloud/learn/neural-networks. [Accessed: 01-Dec-2022].
7. “1.17. neural network models (supervised),” *scikit*. [Online]. Available: https://scikit-learn.org/stable/modules/neural\_networks\_supervised.html. [Accessed: 01-Dec-2022].
8. R. S. Olson, “TPOT API,” *TPOT API - TPOT*. [Online]. Available: http://epistasislab.github.io/tpot/api/. [Accessed: 01-Dec-2022].
9. M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," in *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, pp. 656-667, April 1994, doi: 10.1109/21.286385.
10. Y. Kinha, “An easy guide to choose the Right Machine Learning algorithm,” *KDnuggets*, 17-Feb-2022. [Online]. Available: https://www.kdnuggets.com/2020/05/guide-choose-right-machine-learning-algorithm.html. [Accessed: 01-Dec-2022].
11. S. Kumar, “3 techniques to avoid overfitting of decision trees,” *Medium*, 21-Sep-2021. [Online]. Available: https://towardsdatascience.com/3-techniques-to-avoid-overfitting-of-decision-trees-1e7d3d985a09#:~:text=Decision%20Trees%20are%20a%20non,decision%20trees%20can%20easily%20overfit. [Accessed: 01-Dec-2022].
12. F. Liang, Y. Xiao, and F. Zhao, “Effect of pH on sulfate removal from wastewater using a bioelectrochemical system,” *Chem. Eng. J.*, vol. 218, pp. 147–153, 2013.
13. “Bases - ph values,” *Engineering ToolBox*. [Online]. Available: https://www.engineeringtoolbox.com/bases-ph-d\_402.html. [Accessed: 01-Dec-2022].
14. M. Weber-Shirk, “Acid Neutralizing Capacity,” *Environmental Engineering Laboratory Research Textbook v.0.0.50 documentation*. [Online]. Available: https://monroews.github.io/EnvEngLabTextbook/Acid\_Neutralizing\_Capacity/Acid\_Neutralizing\_Capacity.html. [Accessed: 01-Dec-2022].
15. C. O'Donnell, “Stream Chemistry Data for Alaska's Arctic Network,” *NPS Datastore*, 2020. [Online]. Available: https://irma.nps.gov/DataStore/Reference/Profile/2225760. [Accessed: 01-Dec-2022].
16. B. Johnson, E. Smith, J. W. Ackerman, S. Dye, R. Polinsky, E. Somerville, C. Decker, D. Little, G. J. Pond, and E. D. Amico, “Spatial Convergence in Major Dissolved Ion Concentrations and Implications of Headwater Mining for Downstream Water Quality,” *Wiley Online Library*, 10-Jan-2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/1752-1688.12725. [Accessed: 01-Dec-2022].