Water Potability Prediction Using a Feedforward Neural Network

SEP 769 Cyber Physical Systems
Deep Learning Project

AUTHORS

Juliusz Gasior - 400490100

2025-08-03

Contents

1 Introduction

1.1 Background

Potable water is a fundamental need for all. Safe drinking water is essential for good health and development, geopolitical stability, and public services and industries.

I currently work as a product engineer at a plumbing goods company and this topic is of great importance. At every stage of the water cycle, there are many possible contaminants that can enter the water supply, such as air pollution mixing with rain, phthalates from water bottles leaching into water, or contaminated groundwater feeding into cracks in municipal water mains.

In my work, we strive to reduce or eliminate all sources of chemicals and molecules that may be present in our waterway materials. There are industry standards like NSF 61, NSF 372, Proposition 65, etc. that measure and set limits for allowable contaminants in products. However, some of the biggest causes of contamination come from much earlier than the faucet. Because of this, we are interested in low-cost, low-energy sensors that can detect non-potable water to help prevent contaminant exposure.

1.2 Problem Statement

There are many factors that affect the potability of water, and they are generally not linearly separable and may have complex interactions, making it difficult to create a universal equation that determines the potability. There are organizations that set limits or acceptable ranges for measurable parameters that can help determine potability. These parameters generally do not measure the presence of biological components that can affect health, like E. Coli.

Can a predictive model learn these limits of water makes water potable and understand the interactions between parameters?

1.3 Objectives

The objective of this project is to train a Feedforward Neural Network (FNN) model that can classify if a water sample is potable or not. This model would be intended to be used in water quality assessments. The main goal is to optimize for safety.

Specifically to this project, the objectives are to:

1. clean the water potability dataset by handling missing values
2. perform feature engineering to improve the model's ability to recognize patterns
3. design an FNN that is maximizes precision
4. compare the final FNN with other machine learning models
5. interpret and discuss model behaviors

Due to the objective of this project, the following metrics will be considered:

1. Precision: we want to minimize the amount of false positives. This will be our primary metric to maximize
2. accuracy: we want a model that generally predicts well
3. recall: we are OK with a model that predicts false positives in favor of higher precision. this metric is more for our information.
4. ROCAUC: this metric is for our information to allow us to judge how well the model generalizes. We want models that are better than randomly guessing.
5. F1-score: this is more for our information for us to see how well precision and recall are balanced. However, our objective is to maximize precision

1.4 Scope

The scope of this project is limited to understanding a deep learning model's binary classification behavior on a public water potability dataset.

What is not in scope of this project is real-time data acquisition, detection of biological elements (like bacteria and viruses), and deploying the model to a production state.

This model and work is intended to be for proof-of-concept only and is not meant to replace regulatory testing methods or standards.

1.5 Report Structure

Section 1 contains introductory materials to give context and define the problem, objectives and scope.

Section 2 will be detailing theories behind the models used as well as describing the dataset and its source.

Section 3 will dive deep into the implementation details. This section will systematically go through the project workflow from cleaning the dataset, feature generation and selection, and model experimentation.

Section 4 will explain the code used to tune the model.

Section 5 will show and discuss the results.

2 Theory and Datasets

2.1 Theory

This project primarily focuses on an tuning an FNN but also uses popular machine learning algorithms to compare. All models made are using supervised learning for a binary classification problem.

2.1.1 FNN

An FNN passes input data through one or more hidden layers before producing an output. Each layer applied a series of transformations using weights, biases, and activation functions to expose non-linear patterns. During training, the model adjusts its parameters using backpropagation to minimize prediction errors, measured by a loss function.

2.1.2 Logistic Regression

This type of algorithm models the probability that an input belongs to a certain class using a sigmoid function. There is an assumption of a linear relationship between features.

2.1.3 SVM

This type of algorithm tries to find a boundary line or hyperplane that finds that maximum distance between the closest data points from each class.

2.1.4 KNN

This is a non-parametric type of algorithm that predicts a data sample's class based on a majority-rules system of the K number of known data samples closest to the data being inferred.

2.1.5 Gaussian Naive Bayes

This type of algorithm applies Bayes' theorem assuming features are independent and normally distributed within each class.

2.1.6 Decision Tree

This algorithm splits data into subsets based of feature values that maximize information gain. This models a decision-making process flow base don the training data.

2.1.7 Random Forest

This algorithm is an ensemble of multiple decision tree classifiers. The prediction of the random forest is based on the majority prediction of all of the individual decision trees in the forest.

2.1.8 Gradient Boosted Decision Tree

This algorithm looks to sequentially build decision trees that attempt to correct the mistakes that the previous decision tree had made by minimizing a loss function using gradient descent.

2.2 Dataset

The dataset used is the Water Quality dataset from Kaggle[1]. This dataset contains 3276 data samples with the following nine features:

- pH - the acidity or basicity of water
- Hardness - concentration of calcium and magnesium ions in water
- Solids - total concentration of dissolved substances in water
- Chloramines - a common treatment chemical added to water
- Sulfate - concentration of sulfate ions
- Conductivity - electrical conductivity of water
- Organic Carbon - concentration of organic compounds in water
- Trihalomethanes - concentration of natural byproducts between chlorine and organic matter
- Turbidity - cloudiness of water due to suspended solids

These features are continuous numerical values. The label is Potability where 0 means no, the water is not safe or potable and 1 means yes, the water is safe or potable.

2.3 Data Analysis

There are 3276 data samples in the original dataset but many features are missing values. To deal with this, one could either impute a value to fill or opt to drop the entire sample. I chose to drop all data samples missing values. The final dataset was reduced by 38.6% to 2011 data samples. As shown in figures 1 and 2, the distribution of all features in both classes to continue to be normal.

Our class balance is leaning towards imbalanced where 59.6% of data is labeled as 0 and the remainder is labeled as 1.

According to figure 3, there does not seem to be any features that have strong linear relationships with potability. There also does not seem to be strong feature-to-feature linear relationships, as well as no strong collinearity issues. Table 1 shows the how poorly each feature correlates with potability. Since there aren't collinear issues to consider, we should still be able to safely use all features.

Feature	Correlation with Potability
Solids	0.040674
Turbidity	0.022682
Chloramines	0.020784
pH	0.014530
Trihalomethanes	0.009244
Hardness	-0.001505
Sulfate	-0.015303
Conductivity	-0.015496
Organic Carbon	-0.015567

Table 1: Correlation between potability and water quality features, sorted from highest correlation to lowest

Since there does not seem to be strong linear relationships within the dataset, measuring correlation may not be appropriate. Instead, let's analyze the dataset using Mutual Information (MI) which can be more useful for non-linear data. Table 2 also shows that the original features within the dataset have low predictive power

Feature	Mutual Information Score
Sulfate	0.016167
pH	0.011711
Organic Carbon	0.009949
Hardness	0.007390
Conductivity	0.004854
Chloramines	0.000000
Solids	0.000000
Trihalomethanes	0.000000
Turbidity	0.000000

Table 2: Mutual Information scores between each feature and water potability, sorted from highest to lowest

With the current features and their relationships, it would be expected that created a highly accurate predictive model will be difficult.

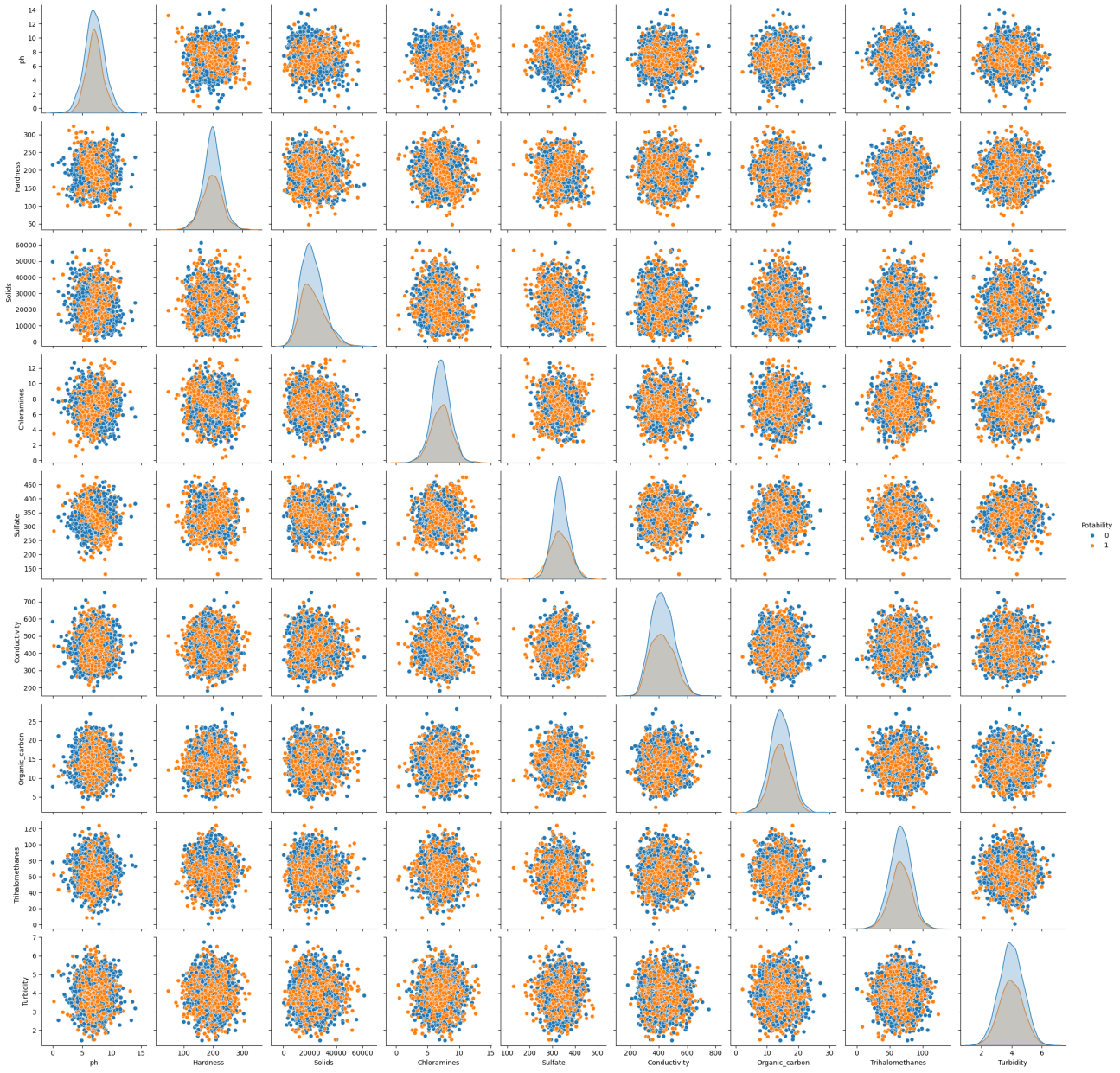

Figure 1: Pairplot of original dataset

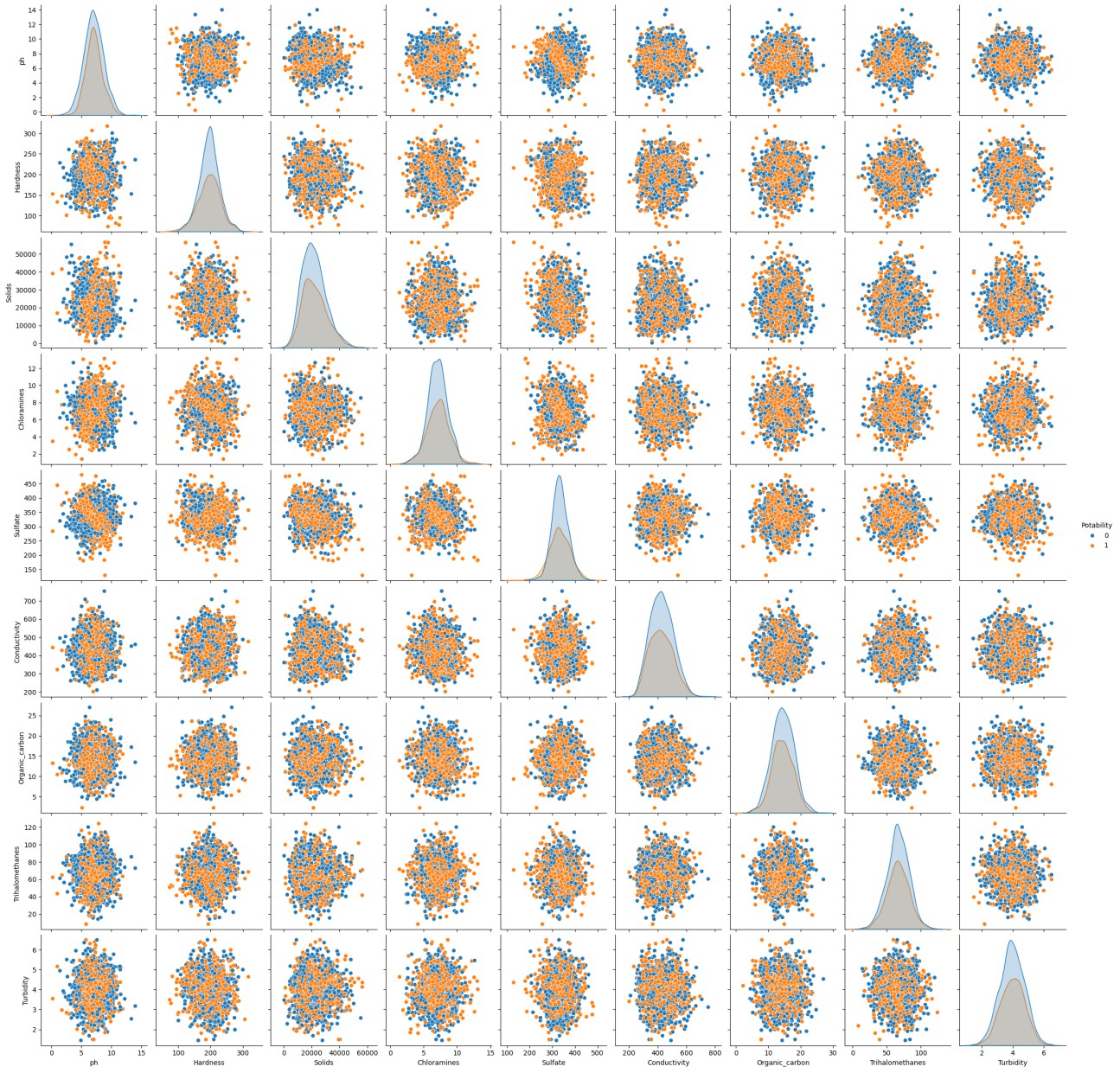

Figure 2: Pairplot of cleaned dataset where missing values were dropped

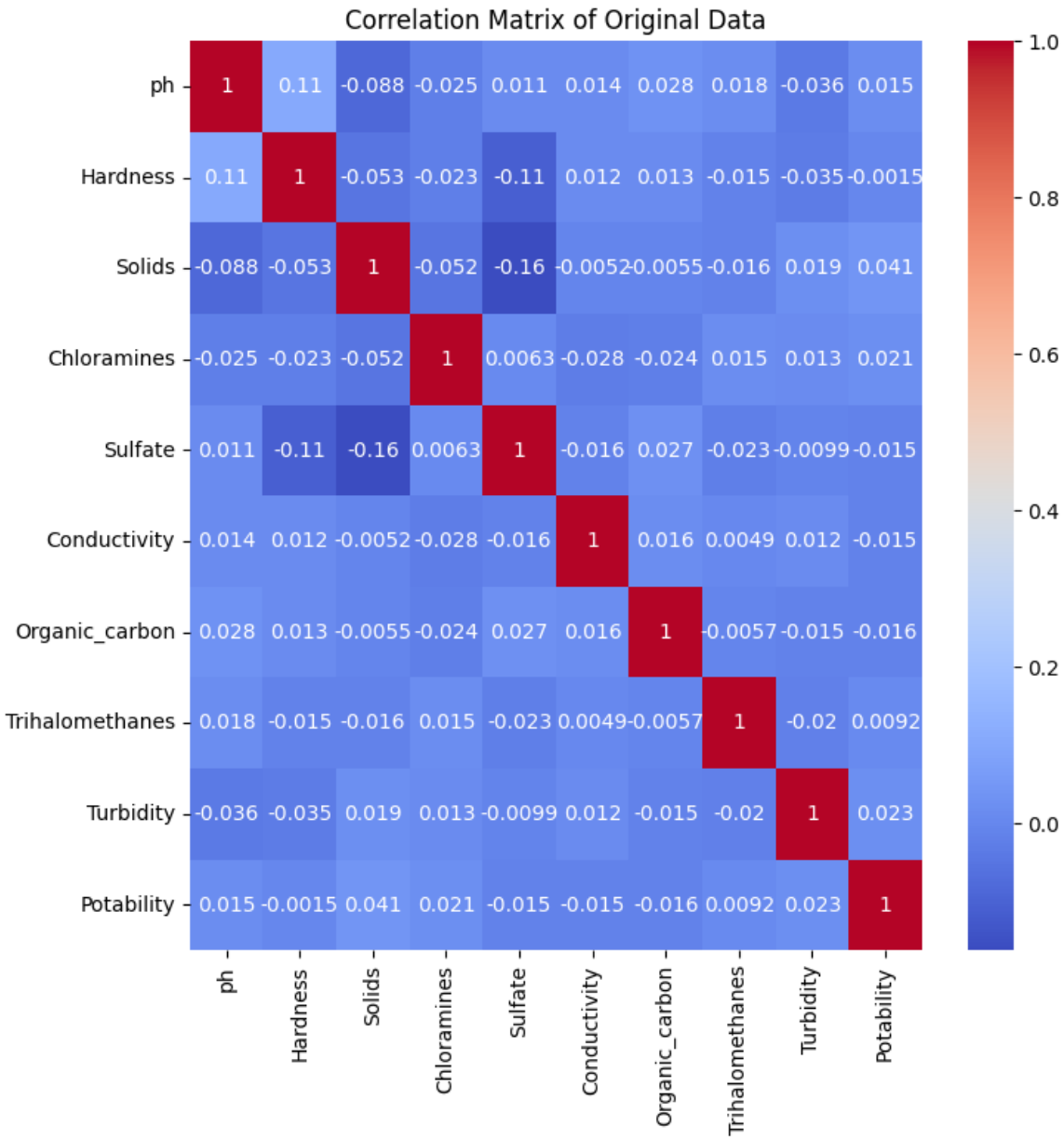

Figure 3: Correlation matrix of dataset

2.4 Feature Engineering

As per figure 3, it seems like solids and sulfates have the strongest linear relationship with a correlation value of -0.16. let's add a ratio feature that describes Solids and Sulfates to the dataset.

We will also create polynomial combinations of our data to see if we can improve the predictive

power of the data. In total, the dataset will now have sixty six features versus the original nine. The data was exported as a CSV and uploaded to my personal Github for future use in model creation.

Tables 3 and 4 show improved correlation and MI scores with the newly created features. The Solids/Sulfate ratio feature looks to be a good addition as it and its feature combinations show stronger relationships with potability. Looking at MI scores specifically, we have added five additional features that have a higher MI score than the highest MI feature from the original dataset (Sulfates).

Feature	**Correlation with Potability**
Solids/Sulfate pH	0.108745
Solids/Sulfate^2	0.088910
Solids/Sulfate Trihalomethanes	0.076359
Solids/Sulfate Turbidity	0.075691
Solids/Sulfate	0.074579
Solids/Sulfate Solids	0.072820
pH Solids	0.070659
Solids/Sulfate Conductivity	0.065829
Solids/Sulfate Hardness	0.063010
Solids/Sulfate Chloramines	0.060508

Table 3: Correlation of engineered features with water potability. Showing top ten features

Feature	**MI Score**
Sulfate Organic_carbon	0.019716
Hardness Turbidity	0.018726
Solids/Sulfate Organic_carbon	0.016927
Sulfate^2	0.016208
Sulfate	0.016167
Solids Turbidity	0.015153
Hardness Sulfate	0.014780
Solids/Sulfate^2	0.014234
Solids/Sulfate	0.014203
Hardness Conductivity	0.014190

Table 4: Mutual Information scores of engineered features with water potability. Showing top ten features

3 Implementation Details

This section outlines the key steps undertaken to develop, train, and evaluate the FNN models for predicting water potability. Reference FNN.ipynb for more details on the execution.

3.1 Environment Setup

The implementation utilized Python with popular libraries such as NumPy, pandas and Scikit-learn for data manipulation, Matplotlib and Seaborn for visualization, and Tensorflow (Keras) for deep learning activities. Random seeds were set in NumPy, random, and Tensorflow modules to ensure reproducibility of results.

3.2 Data Acquisition

A preprocessed and feature-engineered version of the water potability dataset was sourced from my personal Github. repository. The dataset contains the original dataset features, as well as fifty seven additional features that should provide higher predictive power.

3.3 Preprocessing and Feature Scaling

To prepare the data for training and testing, features were normalized using Scikit-learn's MinMaxScaler. This scaling ensures all features are on a comparable scale, between 0 and 1.

3.4 Initial Model

The initial model made to test the dataset was a simple 3 layer FNN as described by figure 4. The initial accuracy of this model is only 58.3%. The generalization of the model looks to be overfit when reviewing figure 5

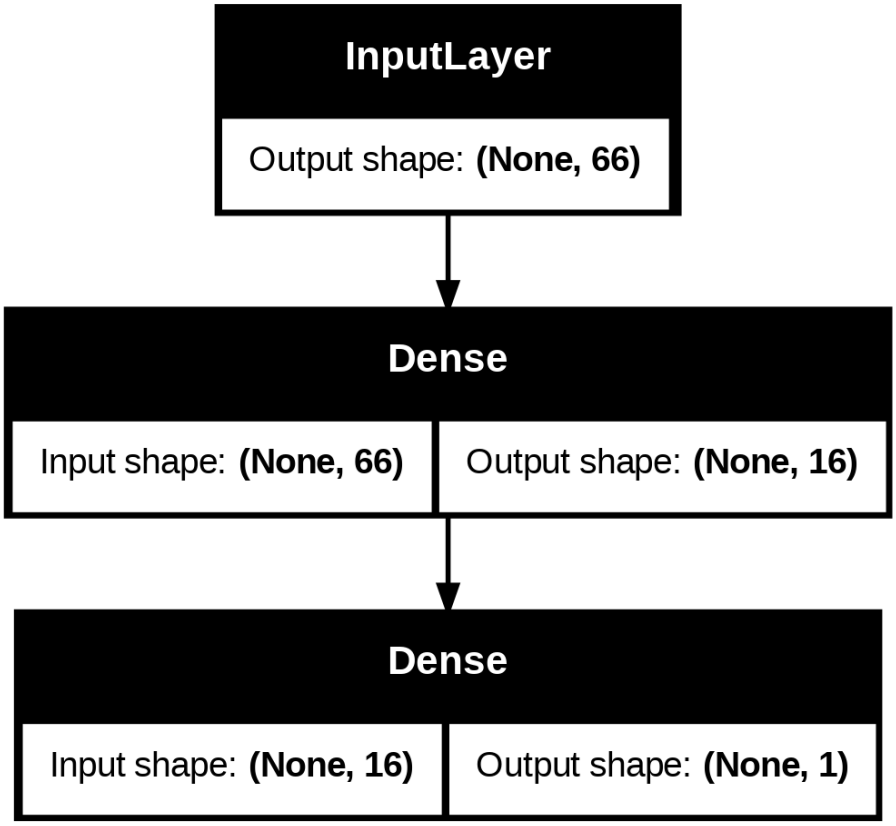

Figure 4: Initial model architecture

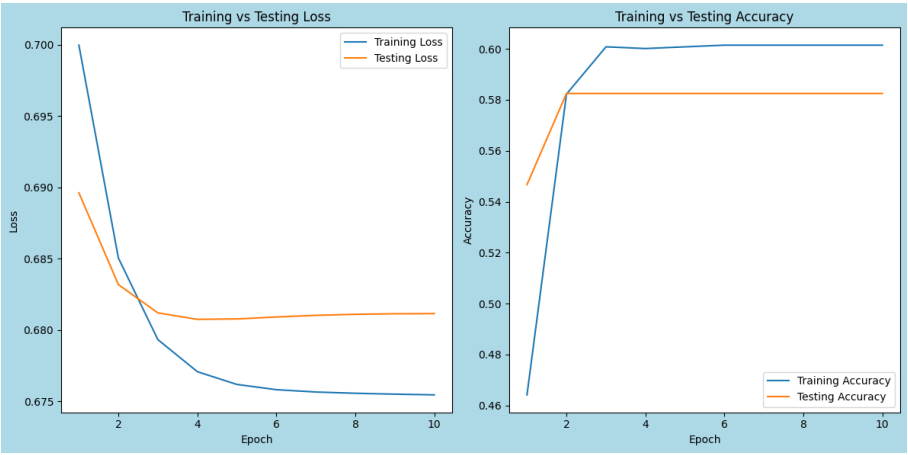

Figure 5: Initial model performance

3.5 Hyperparameter Experiments

The first set of experiments runs two hundred different models with random combinations of values for the following hyperparameters:

- model breadth from 1 to 7
- model depth using 16, 32, 64, 128, 256, or 512 neurons
- activation functions using either ReLU, Sigmoid, Tanh, or Leaky ReLU
- optimizers using either stochastic gradient descent, AdaM, or RMSProp
- loss function always using binary cross-entropy
- dropout rates of 0, 0.25, or 0.5
- epochs of 10, 50, or 100
- batch sizes of 32, 64, or 128
- test split sizes of 0.1, 0.25 or 0.5
- learning rates of 0.0001, 0.001, 0.01, and 0.1

With every run, the program records the model information and the evaluation metrics. This method of running experiments was inspired from Kroepfl's FNN research using the same water potability dataset[2].

Figure 6 shows that most runs in this experiment had relatively low accuracies, where the mean is about 59.5%. The models we are interested in are at the higher accuracy range.

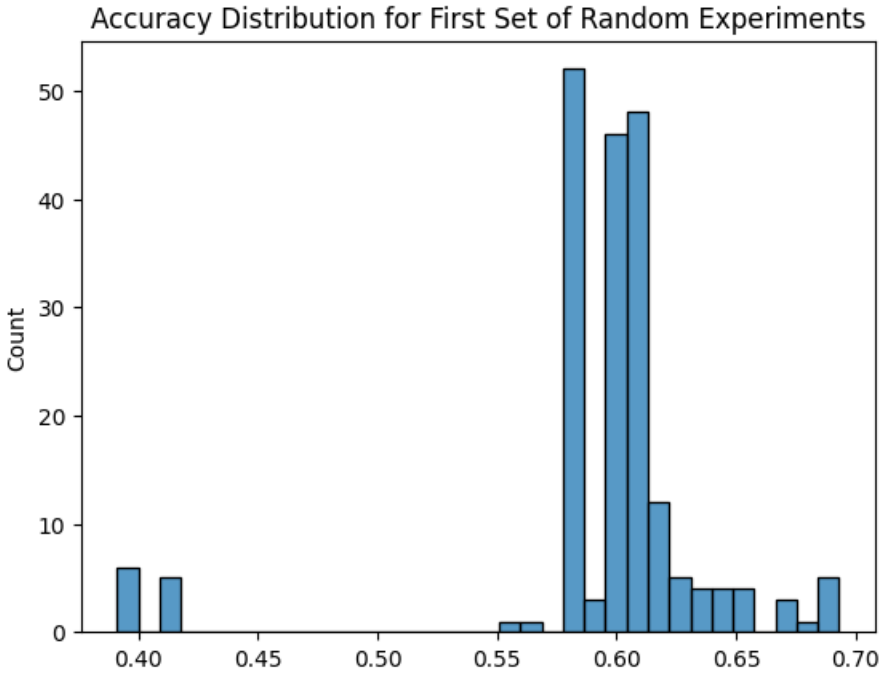

Figure 6: Accuracy distribution of first run of experiments

3.6 Final Architecture and Implementation of FNN

After running the experiments, it was shown that the following hyperparameters provide a model that is robust with decent accuracy:

- one dense layer with 256 neurons
- a dropout rate of 0.25
- a split size of 0.1
- a batch size of 128
- Leaky ReLU as the activation function
- a learning rate of 0.01
- RMSprop as the optimizing function
- 100 epochs

After manually experimenting a bit more on this architecture, a second hidden layer was added but with 128 neurons. This second layer greatly improved precision and figure 8 shows good generalization.

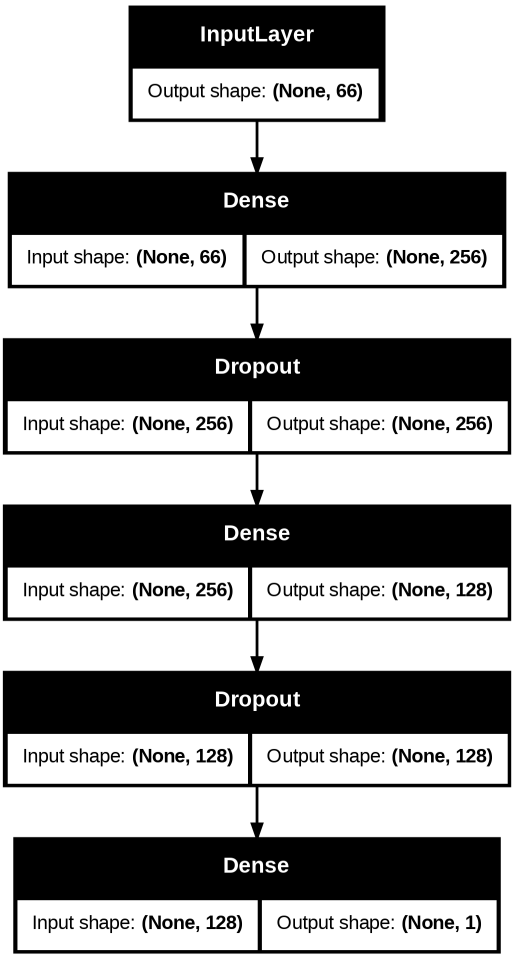

Figure 7: Model architecture of final FNN

Figure 8: Performance behavior of final FNN architecture

3.7 Data Normalization Effects

Another experiment that was looked at was how different types of scaling/normalization effect the model performance. When running the final FNN on data that has been transformed using Scikit-learn's StandardScaler, the model seems to overfit as shown in figure 9.

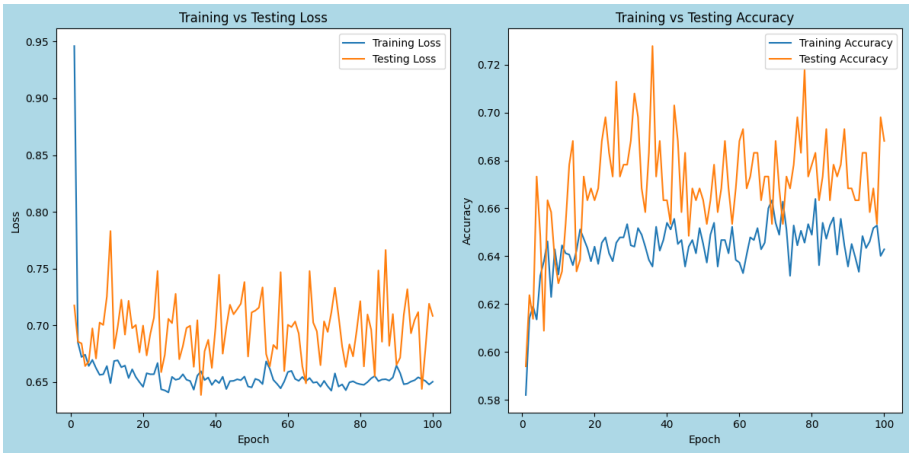

Figure 9: Performance behavior of final FNN training on StandardScaler transformed data

3.8 Dataset Effects

With the final architecture set, we are also interested in seeing how much additional performance we had gained from the feature engineered dataset. Figure 10 shows that the final FNN model does not generalize well with the original dataset. There is a lot of overfitting to the training data occurring.

Figure 10: Performance behavior of final FNN architecture using original dataset without feature engineering

3.9 Machine Learning Comparison

To compare the results of the FNN model being tuned, I also ran experiments on popular machine learning algorithms using Scikit-learn.

Similarly to the deep learning experiments, the data was brought in via Github and normalized and split into the same training and test sizes.

Scikit-learns GridSearch was used to tune each model parametrically and their validation results were reported. This method was also inspired by Kroepfl[3].

4 Explanation of Source Code

The location of all files can be found in this location:
Github Repository

The implementation is organized as a Python Jupyter Notebook and follows a readable structure. This section refers to FNN.ipynb. Key components include:

1. Library imports
2. Data processing
3. Train-Test Split
4. Tensor conversion and batching
5. Initial model definition and training

6. Model evaluation
7. Performance plotting
8. Hyperparamter experimentation

5 Results and Discussion

5.1 Results

Comments	Model	Accuracy	Precision	F1-score	Recall	ROC-AUC
Initial	FNN	0.583	-	-	-	0.507
Final	FNN	0.668	0.929	0.280	0.165	0.659
Tuned	KNN	0.624	0.552	0.296	0.203	0.548
Tuned	Logistic Regression	0.707	0.717	0.528	0.418	0.656
Tuned	SVM	0.718	0.729	0.551	0.443	0.669
Tuned	Gaussian Naive Bayes	0.658	0.619	0.430	0.329	0.600
Tuned	Decision Tree	0.604	0.494	0.506	0.519	0.589
Tuned	Random Forest	0.713	0.784	0.500	0.367	0.651
Tuned	Gradient Boost Decision Tree	0.688	0.667	0.504	0.405	0.638

Table 5: Comparison of Model Performance

5.2 Discussion

This project explored the performance of FNNs against a set of traditional machine learning models for binary classification of water potability.The evaluation focused on accuracy, precision, recall, F1-score and ROC-AUC to assess both correctness and robustness of each approach. Precision was chosen as the metric to maximize under the context that false positives (non-potable water classified as potable) could lead to serious health risks.

The final FNN achieved a precision of 0.929, meaning 93% of water samples predicted to potable were indeed safe (or 7% of non-potable samples were misclassified as potable). However, this model is highly conservative as the high precision came at a cost of low recall and thus F1-score. The strong precision score of the final FNN indicates that it was able to learn a narrow but confident decision boundary.

From a health standpoint, this behavior may be seen as justifiable, especially in areas that have poorer water treatment practices and infrastructure. It may be argued that being safe is better than being sorry.

Of the machine learning models, none were able to boast such a high precision. However, the Random Forest and SVM models were able to provide more balanced performance which may be more desirable is maximizing accuracy was more important.

6 Recommendations for Future Work

The final FNN model was made to be highly precise at the cost of accuracy. This may be good in some instances but most application may want to maximize both precision (for safety) and accuracy (to lower waste/cost).

Future work on the FNN may include:

- tuning the loss function to favor higher precision
- Implement threshold tuning on the FNN output to optimize for desired operating points on the precision-recall curve
- Evaluate ensemble methods that combine the FNN's conservative behavior with other models' higher accuracy

List of Figures

List of Tables

Nomenclature

AdaM Adaptive Momentum

FNN Feedforward Neural Network

KNN K Nearest Neighbors

MI Mutual Information

ReLU Rectifier Linear Unit

RMSprop Root Mean Square Propagation

SVM Support Vector Machine

References

[1] A. Kadiwal, "Water Qaulity: Drinking Water Potability Dataset." Kaggle Dataset. Accessed: 2025-07-20.

[2] J. Kroepfl, "Water quality fnn classification." `https://github.com/johannes-kroepfl-97/Water-Quality-Classification/blob/main/fnn_pytorch.ipynb`, 2022. Accessed: 2025-07-23.

[3] J. Kroepfl, "Water quality machine learning classification." `https://github.com/johannes-kroepfl-97/Water-Quality-Classification/blob/main/water_quality.ipynb`, 2022. Accessed: 2025-07-23.

7 Appendix A