Machine Learning-Based Detection of Cholesterol Presence in Foods Using Nutrient Composition Analysis

Peter Lahanas

[1]The University of Queensland

Introduction

The provided dataset is sourced from Food Standards Australia and New Zealand and contains various information about the nutrient content of Australian foods. The purpose of this report is to implement various machine learning models and evaluate their performance.

Problem Definition

The chosen procedure aims to solve a binary classification problem by using food nutrients (numeric variables) as input features to a model which determines whether the food contains cholesterol (categorical variable). The target variable fits into one of the two categories:

- Contains Cholesterol (encoded as 1)
- Does **NOT** Contain Cholesterol (encoded as a 0)

Several classification models will be constructed and their performance will be evaluated. Different ensemble methods will be used to help minimise error in the model.

Initial Data Preprocessing

Data Overview

Some summary output was generated for the dataset to understand what features were available.

```
Dataset shape: (1616, 293)
Non-numeric Features: ['Public Food Key' '
    Food Name']
Number of numeric features: 291
```

It is observed that the dataset contains 1616 rows and 293 columns. Of the available features, two of these are non-numeric.

Missing Data

For the purpose of this investigation, missing observations will be handled by either:

- Dropping the observation
- Dropping the feature

Firstly, several features contain more than 10% missing values. As there is no information about why these values are empty, the features were dropped from the dataset. In addition to this, any observations with 'NaN' values were dropped. The first three features of 'Public Food Key', 'Classification' and 'Food Name' are irrelevant to the modelling so were also removed from the dataset.

```
(1616, 293)
(1614, 60)
```

In total, 233 features are dropped including 'Galactose (g)' and 'C22:1w11 (%T)'.

Classification Modelling

Checking the Dataset

The first consideration was to check the balance of the data as an unbalanced dataset could lead to increased bias in the model. In addition, a categorical variable was constructed as our target variable:

- Contains cholesterol (Cholesterol (mg) > 0)
- Does not contain cholesterol (Otherwise)

In code:

```
# Create our data set for modelling
data_y = np.where(df_clean['Cholesterol  (mg)
    '] > 0, 1, 0)
data_x = df_clean.drop('Cholesterol  (mg)',
    axis=1)
```

```
Observations with cholesterol:  802
Observations without cholesterol:  812
```

The number of observations with and without cholesterol are relatively balanced (802 vs 812) however, the slight discrepancy should be noted as it may lead to bias in the training process.

Data Normalisation

As a variety of models will be tested (with some requiring normalisation), the input data will be Z-score normalised as it is usually preferred for input

data that is normally distributed. While there are too many features to individually check that this assumption is valid, this normalisation method should still be ok due to its scale invariance.

Model Evaluation

The models will be evaluated based on a misclassification error function $E(\hat{y}, y)$ and an estimation of E_{new} will be calculated using the hold-out validation approach ($E_{hold\text{-}out}$) [Lindholm et al., 2022]. Although estimating E_{new} with $E_{k\text{-}fold}$ allows for the model to be trained on all data points, $E_{hold\text{-}out}$ has lower computational complexity and allows for simple tuning of hyperparameters so will be used in this investigation [Lindholm et al., 2022].
A concern using $E_{hold\text{-}out}$ is that there is less data available to train the model. However, the relatively large dataset (1614 observations) allows for a reasonable training set even with a 70/30 train test split. The number of observation far exceeds the number of features.

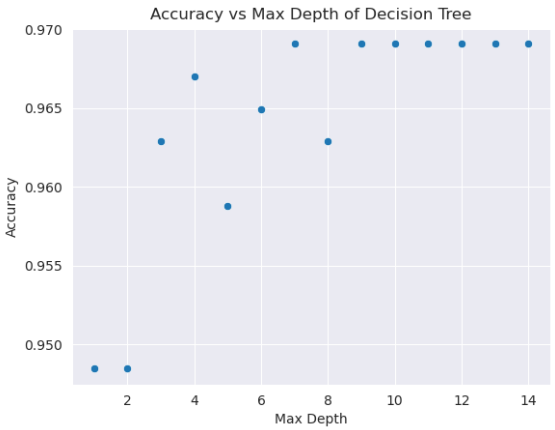

Figure 1: *How the maximum depth of the decision tree impacts on the accuracy score of the model*

Decision Tree Classifier

The first model created for this dataset was a decision tree classification model. The main considerations in the development of this model was the tuning of the hyperparameters:

- max_depth -> The maximum depth of the tree
- max_features -> The number of features considered at each split
- criterion -> The function that measures the quality of the split e.g. 'gini'
- splitter -> The strategy to choose the best split e.g. 'random'

There are several methods for tuning these hyperparameters however, the first approach was to graph the accuracy score of the model against different values for these hyperparameters.

As highlighted in Figure 1, the accuracy score of the model generally increased as the maximum depth of the tree increased. After a depth of 9, the accuracy remained constant as all the leaves were pure (contained less than the minimum number of samples to split). While accuracy increased, it must be considered that increasing the depth of the tree can increase the variance of our model through overfitting.
The number of features considered at each split of the tree also impacted on the accuracy score. However, as features are randomly selected, there was variability observed in the accuracy scores and it is difficult to determine which value is optimal. By randomly selecting features, the algorithm reduces

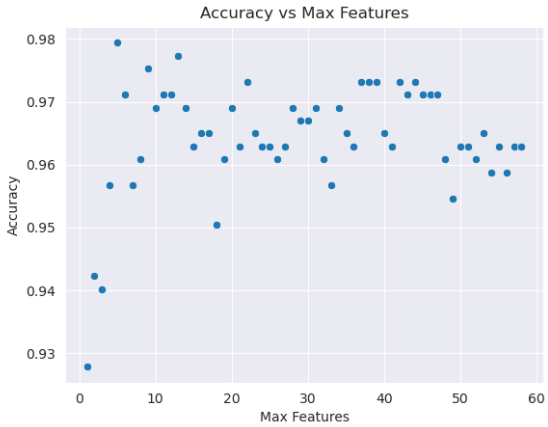

Figure 2: *How varying the number of features considered at each level of the tree impacts on the accuracy score of the model*

overfitting to our training set and therefore is more likely to have a smaller variance.

Hyperparameter Tuning

Rather than estimate values for these parameters from the graphs, the grid search implementation from sklearn was used. In addition, the sklearn method uses cross-validation to minimise overfitting. The GridSearchCV object was run and tested the following parameters:

```
params = {
  'criterion':  ['gini', 'entropy'],
  'max_depth':  [None, 2, 4, 6, 8, 10],
  'max_features': [None, 'sqrt', 'log2', 0.2,
     0.4, 0.6, 0.8],
  'splitter': ['best', 'random']
}
```

Based on the training set (with cross-validation), the optimal hyperparameters were:

```
Fitting 5 folds for each of 168 candidates,
    totalling 840 fits
{'criterion': 'entropy', 'max_depth': None, '
    max_features': 0.4, 'splitter': 'best'}
```

The decision tree classifier was reconstructed using these optimised parameters.

Model Evaluation

The first metric to consider is how well the model predicted the testing data. Using the accuracy score from sklearn, the calculated accuracy score was given as 0.9649484536082474. This indicates that on our testing dataset, the model correctly classified 96.5% of foods as containing cholesterol. A confusion matrix was constructed to determine the accuracy of each label. As highlighted in Figure 3, the model showed

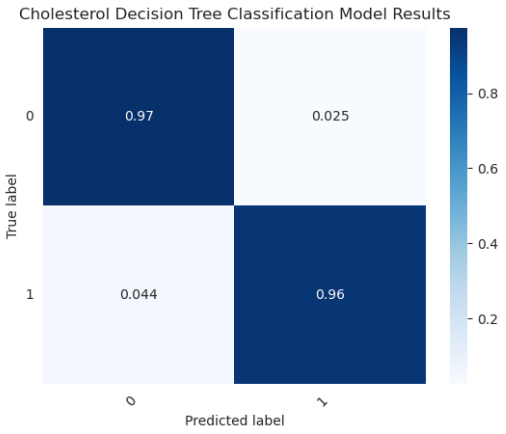

Figure 3: *Confusion matrix for the decision tree classifier model (values given as a percentage of test data label)*

minimal bias towards either label (contains or does not contain cholesterol) as both true positives and true negatives has a similar accuracy of 96% and 97% respectively. The reverse is true for the incorrectly identified labels with both having a similar misclassification percentage. This indicates that the discrepancy in the number of observations with and without cholesterol (802 vs 812) had minimal impact on the performance of our model in regards to its bias towards certain labels.

K-NN Classifier

Similarly to the decision tree classification model, several hyperparameters required tuning. The parameters of interest include:

- n_neighbours -> The number of neighbours considered when generating a predicted value
- weights -> The weighting function used for each neighbour e.g. 'uniform'

Again, the accuracy score of the model was compared for different values of these hyperparameters.

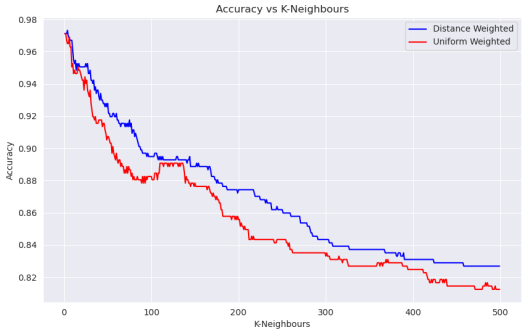

Figure 4: *Change in model accuracy for various hyperparameters*

Figure 4 highlights how increasing the number of neighbours used in the algorithm (K-Neighbours) increases the bias of our model by underfitting the training dataset. The two weighting functions showed a similar trend with the uniform weighting performing slightly worse. The distance function penalises further points more greatly and performed better for this dataset.

Hyperparameter Tuning

The sklearn GridSearchCV was utilised to generate the best combination of weighting function and n-neighbours.

```
params = {
'weights':  ['uniform', 'distance'],
'n_neighbors': np.arange(1, 500)
}
```

```
Fitting 5 folds for each of 998 candidates,
    totalling 4990 fits
{'n_neighbors': 3, 'weights': 'distance'}
```

Model Evaluation

The accuracy score of the tuned model yielded a value of 0.9711340206185567. This means that approximately 97.1% of the test foods were correctly classified as either containing or not containing cholesterol. The confusion matrix (Figure 5) showed minimal bias towards any of the labels in our tuned model with both the positive and negative labels having roughly even accuracy scores.

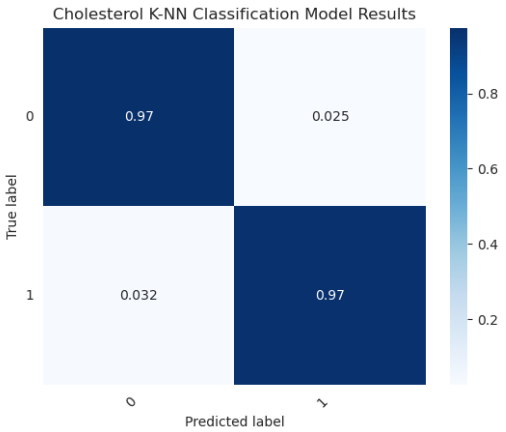

Figure 5: *Confusion matrix for the tuned K-NN model*

Logistic Regression Model

The logistic regression model modifies a linear model to be used for classification rather than regression. The hyper-parameters of this model include:

- solver -> The algorithm used to optimise the model weights e.g. Limited-memory Broyden-Fletcher-Goldfarb-Shanno ('lbfgs')
- max_iter -> The maximum number of iterations that the solver is given to converge

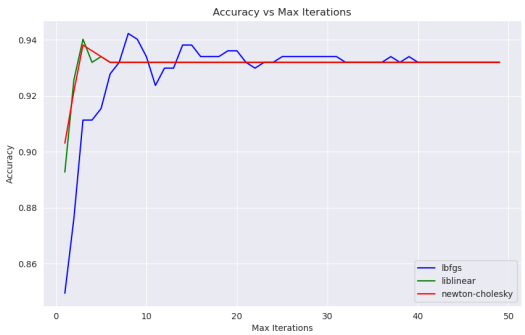

Figure 6: *Accuracy score of logistic model with varying hyper-parameters*

Figure 6 highlights the difference between the different solving algorithms. However, for this dataset, if enough iterations are allowed for all the solvers, the algorithm converges with few iterations. Due to this, the hyper-parameters used for the logistic regression can be left as the default from sklearn.

Model Evaluation

The tuned logistic regression model gave an accuracy score of 0.931958762886598 with the model correctly classifying food cholesterol approximately 93.2% of the time (for the test set). As with the other models, the logistic regression also had a relatively even split between false positives and false negatives indicating a relatively even split in the training set (Figure 7).

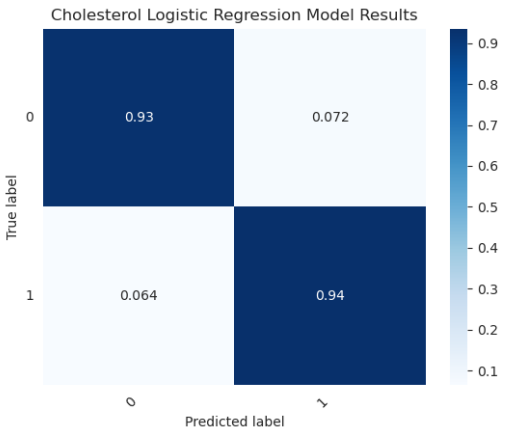

Figure 7: *Confusion matrix for the tuned logistic regression model*

Reducing Model Bias and Variance

Several ensemble based techniques can be utilised to minimise the error from the calculated models. The error term for our model can be broken down into the following equation.

$$\begin{aligned} Error(Model) = Variance(Model) \\ + Bias(Model) \\ + Variance(IrreducibleError) \end{aligned} \tag{1}$$

A technique to reduce variance (without increasing bias) is bootstrap aggregating (bagging) and boosting can be used to reduce bias in high-bias based models. The models from previous sections were further refined using these techniques.

Bagging

Using the sklearn BaggingClassifier, each of the models from above had bagging applied. The default parameters were used, with 10 base estimators and each sample containing the same number of elements as the training set. The results are summarised in Table 1.

Table 1: *Results after applying bagging to each model. Note that '*' refers to the bagged model*

Model	$E_{hold\text{-}out}$	$*E_{hold\text{-}out}$	Δ%
Decision Tree	3.51%	2.27%	-1.24%
K-NN	2.89%	3.51%	+0.62%
Logistic	6.80%	6.39%	-0.41%

The bootstrap process involves artificially creating new training datasets for each model by uniformly sampling from our current training set (with replacement). This is considered a reasonable way to estimate sampling from the population [Lindholm et al., 2022]. Due to the nature of each model, some benefited more significantly from bagging while other did not. Generally speaking, the model will benefit more significantly from bagging if it tends to overfit the training data while models that have a relatively low variance will have minimal benefit from bagging.

Decision Tree:
The decision tree model showed the greatest reduction in hold-out error (1.24%) after applying bagging. Decision trees can become very deep and complex when the tree's depth is large and as a result, they tend to overfit the training data. Bagging is an effective technique for tuning this model because it balances out the overfitting of each model in the ensemble by averaging the results.

K-NN:
Unlike the decision tree, the K-NN model increased in hold-out error after bagging. The results of K-NN can vary significantly depending on the number of neighbours (k) used when predicting in the model. A small k usually leads to lower bias but high variance through overfitting while a high k creates a smoother decision boundary but usually increases the bias of the model. While the k selected was relatively small (k=3), averaging the results of multiple K-NN models (trained on the bootstrapped datasets) did not assist with model generalisation.

Logistic Model:
Similarly to K-NN, the effects of bagging on the logistic model are minimal. While there was an improvement in the model (reduction in $E_{hold\text{-}out}$), it was a small change. Due to the nature of a logistic model (non-linear transformation of a linear model), it has low variance and tends to generalise well to new data. This means there is little to no benefit from averaging the results of several models trained on bootstrapped data unless the dataset is highly imbalanced or contains outliers. If these conditions are met, bagging will likely help smooth out and reduce the variability from any outliers.

Boosting

Similarly to bagging, each model was boosted using the sklearn AdaBoostClassifier with the default values for learning rate and number of estimators.

Table 2: *Results after applying boosting to each model. Note that '*' refers to the boosted model*

Model	$E_{hold\text{-}out}$	$*E_{hold\text{-}out}$	Δ%
Decision Tree	3.51%	4.33%	+0.82%
K-NN	NA	NA	NA
Logistic	6.80%	7.63%	+0.83%

Decision Tree and Logistic Model:
Boosting decreased the performance of both the decision tree model and the logistic model. The reason for this decrease in performance is likely due to the strength and high variance of the base model. As boosting aims to increase the flexibility of the model, it possible to overfit to the training dataset and typically, using deep classification trees (high-variance models) for the base classifier leads to deteriorated performance after boosting [Lindholm et al., 2022]. It is good practice to use a simple model with high bias and increase its flexibility through boosting.
To highlight this effect, new decision tree models were tested with depths of 1 and 3. The results are summarised in Figure 8.

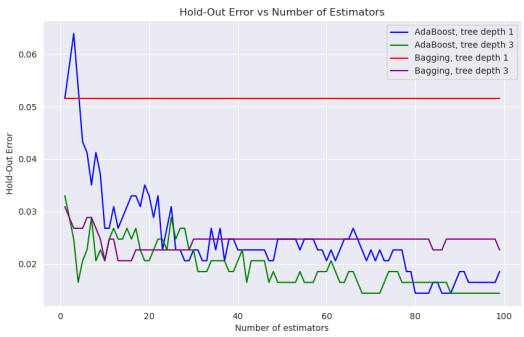

Figure 8: *How ensemble methods change the hold-out error for a varying number of estimators*

The bagged models showed little to no performance gains with the increased number of estimators.

This is because the weak base models already have a low variance so bagging does not improve their error score. However, boosting had a pretty significant improvement to model scores. Unlike boosting the more complex models from earlier, boosting the weak and highly bias models greatly improved their accuracy score by fitting them more tightly to the training data.

Neural Network Model

Neural networks are highly flexible models that can be applied to a range of modelling problems. Their main advantages include [Ellis, 2021]:

- Performs well on unstructured data
- No assumption of a linear relationship
- Detect and account for interactions
- Robust to outliers
- Can handle multi-class outcomes

The main disadvantages of neural networks are that they [Ellis, 2021]:

- Require large training sets
- Contain many hyperparameters that require tuning
- Require large amounts of computation
- No interpretable coefficients

In regards to the provided dataset, the relatively small sample size and simple nature of the problem means that it is unlikely a neural network is the best solution to the problem. However, neural networks usually perform well when several input features are available so this modelling technique will still be tested.

Types of Neural Networks

Several types of neural networks are available depending on the underlying problem. The uses of few neural networks are summarised in Table 3 [Brownlee, 2022].
As the cholesterol problem is a classification problem, a multilayer perceptron (MLP) will be used.

Table 3: *Types of neural network models and their main functions*

Network Type	Use Cases
Multilayer Perceptrons (MLPs)	• Classification tasks • Regression tasks • Pattern recognition
Convolution Neural Networks	• Image classification • Object detection • Image segmentation
Recurrent Neural Networks	• Natural language processing • Speech recognition • Time-series prediction

NN Design

NN Structure:
While there is no guide for designing the layers of the network, it was decided that an input layer of 59 (number of input features) would be used and the output layer would contain 1 node (classification) (Figure 9). The two hidden layers were kept relatively small for training.

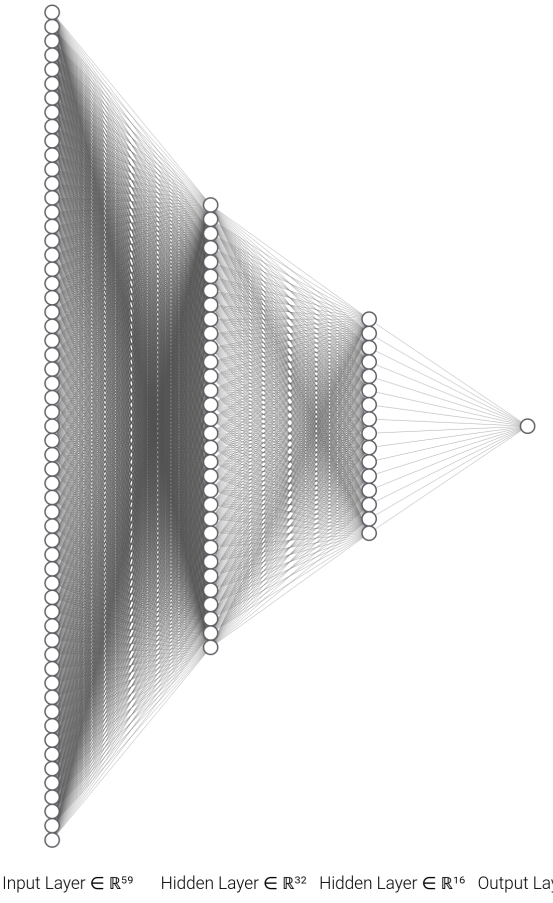

Figure 9: *MLP design with two hidden layers*

Activation Function:
For the hidden layers, it is common practice to use a ReLU activation function due to its simple implementation. The constructed model will use ReLU for its hidden layers. For the output layer, a sigmoid function will be utilised, which is appropriate for classification problems because it maps the output to a value between 0 and 1.

Model Construction:
The Keras API was utilised to build the model and dropout was added in the hidden layers to help reduce overfitting. The following model summary was generated.

```
_________________________________________________
 Layer (type)         Output Shape     Param #
=================================================
dense_66 (Dense)      (None, 59)        3540

dense_67 (Dense)      (None, 32)        1920

dropout_24 (Dropout) (None, 32)        0

dense_68 (Dense)      (None, 16)        528

dropout_25 (Dropout) (None, 16)        0

dense_69 (Dense)      (None, 1)         17

=================================================
Total params: 6,005
Trainable params: 6,005
Non-trainable params: 0
```

Model Evaluation

The model yielded an accuracy score of 0.9773 on the test data which is an $E_{\text{hold-out}}$ of around 2.3% which is on par with the bagged decision tree. As illustrated in Figure 10, the error of the validation set ($E_{\text{hold-out}}$) stopped improving after around 20 epochs while the training error kept improving. Using too many epochs can result in overfitting the model to the training set and has the potential to increase the error of unseen data.
Similarly to the other models, the neural network did not show much bias to positive or negative labels with a relatively even split in the confusion matrix (Figure 11). Again, this is likely due to the balanced number of each label in the training set.

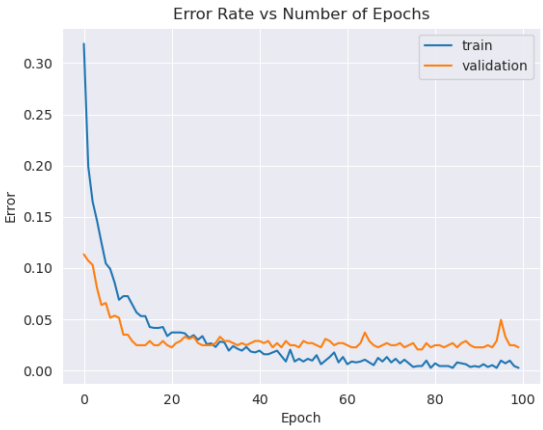

Figure 10: *How the error rate varies with epochs*

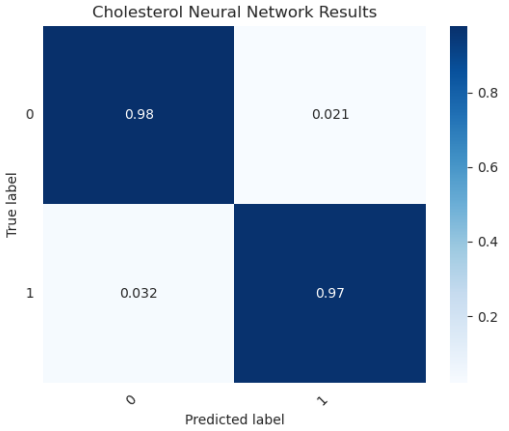

Figure 11: *Confusion matrix for the neural network*

Conclusion

In summary, each model performed relatively well after tuning any hyperparameters or implementing ensemble techniques. While the neural network had one of the lowest $E_{\text{hold-out}}$ errors (2.3%), its complexity and potential overfitting to the training set means

it is not necessarily the most suitable model. Rather, the bagged decision tree model had a similar $E_{\text{hold-out}}$ and is also a significantly simpler model. Several analytical techniques and processes were not investigated in this report and several future directions include:

- Feature selection -> Removing features is a type of regularisation and can improve model performance [Lindholm et al., 2022]
- Missing Data -> All missing data was just dropped, other techniques could be used such as imputation
- Outlier analysis -> The dataset was not checked for any outliers before generating a training set

Overall, the analytical techniques explored in this report provide an insight into machine learning techniques to solve a classification problem.

References

[Brownlee, 2022] Brownlee, J. (2022). Machine learning mastery: When to use mlp, cnn, and rnn neural networks. `https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/`.

[Ellis, 2021] Ellis, C. (2021). Crunching the data: When to use neural networks. `https://crunchingthedata.com/when-to-use-neural-networks/`.

[Lindholm et al., 2022] Lindholm, A., Wahlström, N., Lindsten, F., and Schön, T. B. (2022). *Machine learning: A first course for engineers and scientists*. Cambridge University Press.