Comparing Machine Learning and Deep Learning Models in the Detection of Brain Tumors

Amina Isse, Concordia University
Github Link: github.com/isse-amina/brain-tumor-detection

Abstract

Medical image processing is an ever-growing field in which a database of either CT scans or MRI images are used in the training of various machine learning and deep learning models. In this paper, we compare the performance of three learning models in their detection of brain tumors. Specifically, we opted to train the following models: the k-means cluster, the k-nearest neighbor and the multilayer perceptron. Each model is designed to output the integer "1" if it predicts that the brain is tumorous, and zero otherwise. To assess their performance, we compared their classification reports as well as their computational times.

I. Introduction

Over the years, an increasing amount of medical image databases have been made available to the public. These databases usually contain either CT (computed tomography) scans or MRIs (magnetic resonance imaging), which capture black and white images within our body, albeit in different ways. [1] These databases have thus allowed machine learning and deep learning models to be trained to detect abnormalities within the human body. These tools have not been created to replace radiologists, but instead to work alongside them.

In this paper, machine learning and deep learning models were trained to predict whether a brain was tumorous. The MRIs display the tumors as white patches that, when large, are easily seen by the human eye. However, sometimes, these patches are more subtle. It's in those cases that machine learning and deep learning can be especially useful in classifying tumorous and non-tumorous brain MRIs.

Classification works by training a model to output a specific label depending on the features. [2] With image classification, the features are the image pixels, and the label represents the class to which the image belongs to. For colored images, there are three channels that need to be worked with: red, blue and green. However, considering that MRIs are black and white, we will only be working with one channel. Furthermore, in some image classification problems, there are various classes to which the image can belong to. For instance, with proper training, a model would be able to classify MRIs of brain tumors, multiple sclerosis, and dementia. [3] Nonetheless, in this paper, the image classification problem is binary.

To accomplish this binary image classification, three models were trained: a k-means cluster, a k-nearest neighbor algorithm and a multilayer perceptron. The accuracy of some of these models depends on their parameters; hence, for the k-nearest neighbor algorithm and the multilayer perceptron, parameters were chosen by tuning the hyperparameters.

II. Related Work

One of the most commonly used neural network architectures is the convolutional neural network. Indeed, it is considered to be the most powerful supervised learning model. [4] It is especially practical in the analysis of images, thus it is often used in medical image processing. The main focus of such papers is to use convolutional neural networks (CNNs) to achieve the highest accuracy possible. For example, in a research paper published to the Biomedical and Pharmacology Journal, brain tumor classification using a CNN achieved an accuracy of 97.5%. [5] Furthermore, in another paper published to the Iranian Journal of Science and Technology, multi-classification of brain tumors was completed using three different CNN models, in which an accuracy of 99.33% was achieved with one of the models. [6]

The goal of this paper is not to build a model with greater accuracy than the convolutional neural networks available in literature, but instead, our objective is to determine how significant the differences in performance are between other machine learning and deep learning models, in which performance is defined by various evaluation metric scores, as well as computational time. To accomplish this goal, we will be using an unsupervised and a supervised machine learning model, as well as a deep learning model.

III. Proposed Methodology

A. Dataset

The dataset obtained from kaggle contains 1500 brain MRI images that are tumorous, and 1500 brain MRI images that are non-tumorous. These images are not uniform in size, hence they were all resized. For samples of tumorous and non-tumorous brain MRIs, refer to Figure 1.

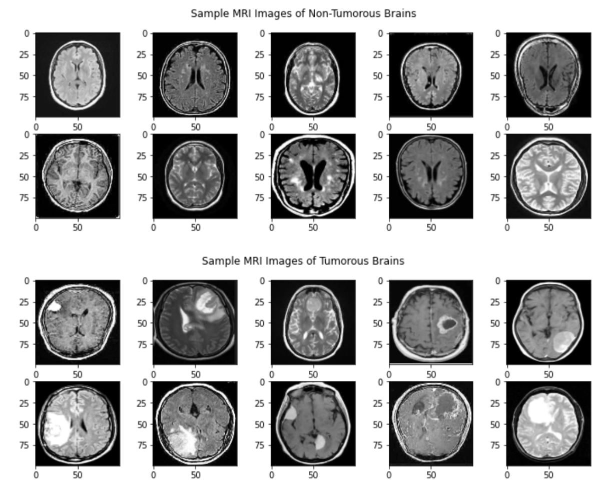

Sample MRI Images of Non-Tumorous Brains
Sample MRI Images of Tumorous Brains
0
25
50
75

Figure 1. Sample of tumorous and non-tumorous brain MRIs

B. Preprocessing

The images need to be modified for them to work with the various models. Hence, the data was preprocessed before being used in the models.

1) *Image Resizing*: In order to have images of uniform size, all of the images were resized to be 100x100 pixels.

2) *Labeling*: Labels were added to each image, which indicated whether the brain was tumorous, represented with the label "1", or non-tumorous, represented with the label "0".

3) *Shuffle Data*: The first 1500 rows of the dataset all had the label "0", whereas the last 1500 rows of the dataset all had the label "1". Therefore, the rows needed to be

shuffled in order to not hinder the model's training process.

4) *Feature and Label Separation*: Each image has features, which are its pixels, and a label, which is its class. Hence, a variable "X" was used to save image features, and a variable "y" was used to save image labels.

5) *Feature Normalization*: The features were normalized to only have values from 0 and 1. This normalization was accomplished by dividing each feature by 255, considering that pixel values range from 0 to 255. Normalization is important because it speeds up the learning rate of the network and it may boost its accuracy. [7]

6) *Testing and Training Split*: The data was split into training and testing sets with a 80/20 split. The class distribution of brain tumors in both sets were almost equal. To see the class distribution in each set, refer to Figure 2.

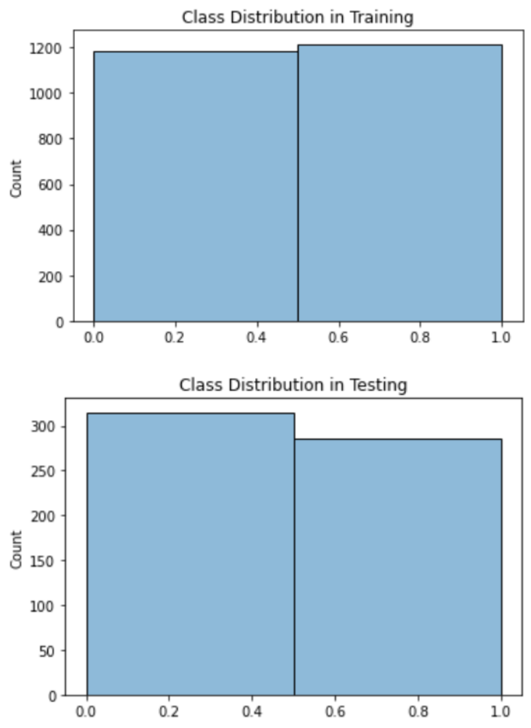


Figure 2. Class distribution of brain tumors in training and testing sets

C. Dependencies

This project relies heavily on the scikit-learn library. The clustering and classification algorithms used throughout the code have been imported from that library. Scikit-learn thus needs to be installed for the code to run without errors. A variety of other modules, such as numpy, pandas and matplotlib, must also be imported.

D. Model Architecture

1) *K-means cluster*: The k-means clustering is an unsupervised learning model. This algorithm is not provided with labels, and thus must detect patterns and create clusters without the help of the user. In order to accomplish its task, it goes through various steps:

- **Step 1:** Select the number of clusters k. In the case of binary classification, k = 2.
- **Step 2:** Randomly select k distinct starting points.
- **Step 3:** Measure the distance between point A and each starting point.
- **Step 4:** Assign point A to the nearest cluster.
- **Step 5:** Repeat Step 3 and 4 for each point.
- **Step 6:** Calculate the mean of each cluster.
- **Step 7:** Repeat Steps 3 through 6 using the mean of the clusters. This process is complete once the resultant clustering is the same as the clustering of the previous iteration.

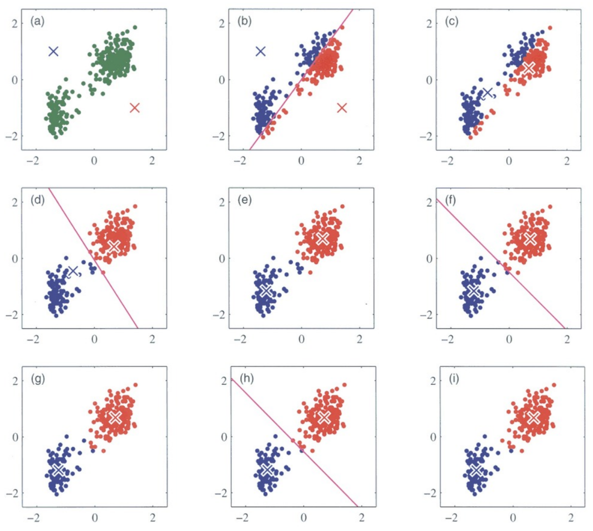


Figure 3. Illustration of K-means Algorithm [8]

To better visualize these steps, refer to Figure 3. The total variation within the clusters is used to access the quality of the clustering. The clustering process then repeats from Step 2 with new starting points. The amount of times the clustering process repeats depends on the value to which "n_init" is initialized to. [9]

2) *K-nearest neighbor*: The k-means neighbor is a supervised learning model. Hence, we start with a dataset with known classes. This model stores all available cases obtained during the training phase, and classifies new cases from the testing phase based on a similarity measure.

To classify data, k nearest neighbors are evaluated. For instance, in Figure 4, if k = 3, then there are more labels belonging to class B than to class A in the data point's neighborhood. Hence, the point is predicted to belong to class B. However if k = 7, then there are more labels belonging to class A than to class B in the data point's neighborhood. Hence, the point is predicted to belong to class A. [10]

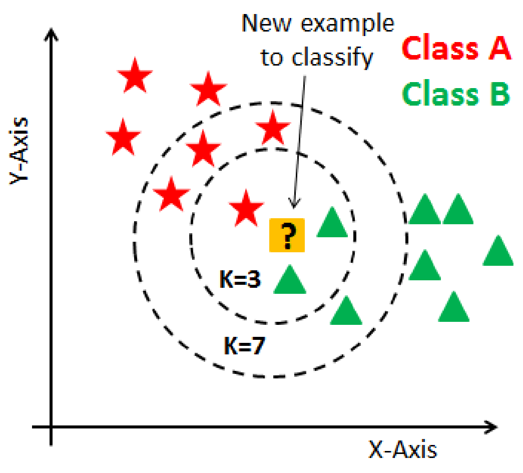


Figure 4. K-Nearest Neighbor Classification Example [11]

3) *Multilayer perceptron*: A multilayer perceptron is an artificial neural network that has an input, a set number of hidden layers, and an output layer. This model is a feedforward algorithm because the result of each layer is being "fed" to the next layer. In order to classify data, the model goes through various steps:

- **Step 1:** Connections between neurons in one layer and neurons in the next all have weights associated with them. These weights are initialized to random numbers, whereas the bias is initialized to 1. [11]
- **Step 2:** The input of a neuron is equal to the weighted sum of the neurons in the previous layer, plus the bias.
- **Step 3:** The neuron applies an activation function to the input it received.
- **Step 4:** When the forward pass is completed, the backward pass begins. The goal of backpropagation is to update the weights in order to improve the performance of the neural network.

The calculation of step 2 and 3 for the first hidden layer neuron in Figure 5 can be summarized with the following formula: $h_1 = \sigma(w_1 * i_1 + w_2 * i_2 + b_1)$, in which σ is an activation function.

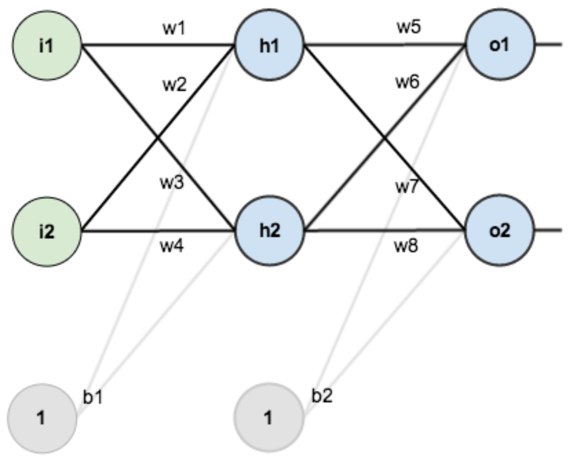


Figure 5. Example of a Basic Structure of a Multilayer Perceptron [13]

IV. Evaluation and Results

A. Hyperparameters

To optimize the training of our selected supervised learning models, we tested relevant hyperparameters using grid search. Grid search "fits" the training dataset with all combinations of the parameters in the parameter grid, and it outputs the best combination. [14] The "base" k-nearest neighbor (base_knn) is the version of the model that uses default parameters, and the "best" k-nearest neighbor (best_knn) is the version of the model using hyperparameter tuning. Similarly, the "base" multilayer perceptron (base_mlp) is the version of the model using default parameters, and the "best" multilayer perceptron (best_mlp) is the version of the model using hyperparameter tuning..

These are the hyperparameters the k-nearest neighbor model and the multilayer perceptron model used:

1) *Random state*: Having the random state be equal to some number ensures the reproducibility of the results. Hence, the random state was set to 1.

2) N*umber of neighbors*: This represents the value of k; in other words, it determines how many neighbors will be taken into account in the similarity measurement for classification. For the base_knn, the number of neighbors was equal to the default of 5. For the best_knn, the number of neighbors was modified to 1.

3) *Hidden layer sizes*: The hidden layer size determines the number of neurons per hidden layer and the number of hidden layers. The base_mlp used the default of 100 neurons per hidden layer, with 1 hidden layer. For the best_mlp, the number of neurons per hidden layer was 10, with 1 hidden layer.

4) *Activation*: This represents the activation function. The base_mlp used the default of reLu, which stands for rectified linear unit model. It returns:

$$f(x) = max(0, x)$$

The best_mlp opted for the same function.

5) *Solver*: Solvers are used during weight optimization. The base_mlp used the adam solver, which is a stochastic gradient-based optimizer, while the best_mlp used the sgd solver, which refers to stochastic gradient descent. [15]

B. Evaluation Criteria

Evaluation metrics are used to evaluate the performance of a model. The metrics compare the classes that the models predicted to the true classes to which the data belongs. When the classification reports are displayed, they show the various evaluation metrics with their scores. The main metrics are the following:

1) *Precision*: Precision is useful in determining whether there are many False Positives. It's formula is the following:

$$Precision = \frac{True\,Positive}{True\,Positive + False\,Positive}$$

2) *Recall*: Recall can be used to select the best model when the cost associated with False Negatives is high. [16] It's formula is the following:

$$Recall = \frac{True\,Positive}{True\,Positive\ + False\,Negative}$$

3) *F1 Score*: F1 is a function of precision and recall. Hence, we examine that metric when we wish to have a balance between precision and recall. It's formula is the following:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

4) *Accuracy*: Accuracy is the ratio of correctly predicted observations to the total number of observations.

$$Accuracy\ = \frac{True\,Positive\ + True\,Negative}{Total}$$

C. Confusion Matrix

Confusion matrices are used to visualize the accuracy of a classification. [17] In the case of binary classification, they display the prediction's true positives, true negatives, false positives and false negatives, as Figure 6 demonstrates.

D. Results

Let us begin by examining the model with the lower accuracy: the k-means clustering. Considering that it uses unsupervised learning, the model had no way of determining whether a cluster should be classified with a "0" for non-tumorous, or with a "1" for tumorous. Hence, the model predicted that "0" meant tumorous, and that "1" meant non-tumorous. Fortunately, there is a simple fix to this problem: all of the model's 0s were changed to 1s, and all of its 1s were changed to 0s. This swap boosted the accuracy from 37% to 63%, and all of the other evaluation metric scores increased as well. Considering that the model could not train with labels, and that k-means clusters are oftentimes not used for binary classification problems, it is to be expected that the model's predictions would yield a low accuracy.

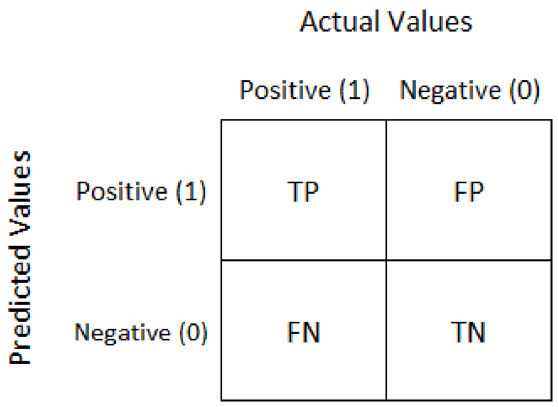
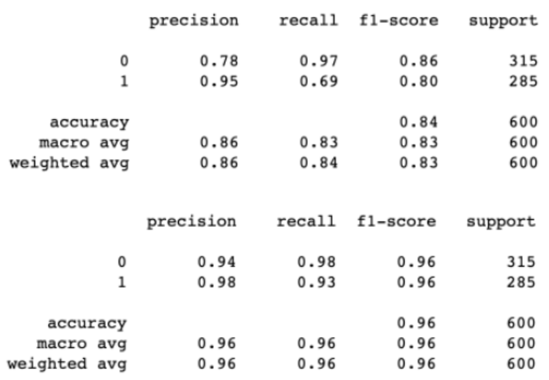


Figure 6. Significance of the Cells in a Confusion Matrix for Binary Classification [18]

The next model we will look into is a machine learning network that uses supervised learning: the k-nearest neighbor algorithm. When the classifier used default parameters, it had an accuracy of 84%. Although that accuracy is not bad, the precision for the "0" label was 78% and the recall for the "1" label was 69%. These scores could be improved by using hyperparameters tuning. With optimal parameters, the model's accuracy increased to 96%, the precision for the "0" label went from 78% to 94%, and the recall went from 69% to 93%, as Figure 7 demonstrates.



```
              precision    recall  f1-score   support

           0       0.78      0.97      0.86       315
           1       0.95      0.69      0.80       285

    accuracy                           0.84       600
   macro avg       0.86      0.83      0.83       600
weighted avg       0.86      0.84      0.83       600

              precision    recall  f1-score   support

           0       0.94      0.98      0.96       315
           1       0.98      0.93      0.96       285

    accuracy                           0.96       600
   macro avg       0.96      0.96      0.96       600
weighted avg       0.96      0.96      0.96       600
```

Figure 7. Classification Report of the K-Nearest Neighbor Classifier with Default Parameters (Above) vs with Hyperparameter Tuning (Below)

The final model that we will go over is the only deep learning model of this project: the multilayer perceptron. Even without hyperparameter tuning, the multilayer perceptron has an accuracy of 96%. This increase in accuracy does come at a computational cost. While all other other models take under 20 seconds to run, the base_mlp takes over a minute. Although it could be argued that once the training has been completed, the model no longer needs to be trained again, making that computation time is irrelevant, the extra computational cost is still not worth the effort, especially considering that the best_knn and the base_mlp have almost identical confusion matrices, as you can see in Figure 8, but the best_knn is drastically faster than the base_mlp. The same could be said about the multilayer perceptron with hyperparameter tuning. Grid search improved the accuracy by only 1% – going from 96% to 97% – while it increased the computational time by over 30 minutes.

It is interesting to note that the optimized models seem to have accuracies around 96%. This might mean that there is a threshold accuracy that the models cannot pass, which can explain why convolutional neural networks are so popular: they are able to surpass this threshold to reach an accuracy of almost 100%.

In summary, although the model with the highest accuracy was the multilayer perceptron with hyperparameter tuning, the model with best overall performance, which includes computational time, is the k-nearest neighbor algorithm with hyperparameter tuning. To see all of the model's accuracies and computational times, please refer to Figure 9.

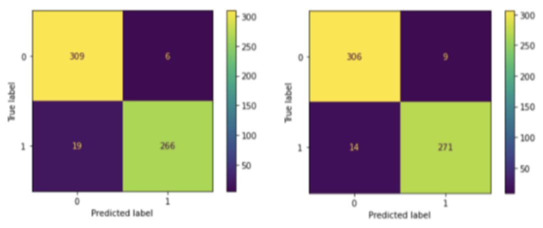


Figure 8. Confusion Matrix of the K-Nearest Neighbor Classifier with Hyperparameter Parameters (Left) vs of the Multilayer Perceptron (Right)

Model	Accuracy (percentage)	Time (seconds)
k-means clustering	63	0.5825
k-nearest neighbor with default parameters	84	0.6880
k-nearest neighbor with tuned parameters	96	17.6550
multilayer perceptron with default parameters	96	66.0000
multilayer perceptron with tuned parameters	97	3352.5100

Figure 9. Accuracies and Computational Times of Different Models

REFERENCES

[1] Healthline. 2022. CT Scans vs. MRIs: Differences, Benefits, and Risks. [online] Available at: <https://www.healthline.com/health/ct-scan-vs-mri> [Accessed 15 April 2022].

[2] Google Developers. 2022. ML Practicum: Image Classification | Google Developers. [online] Available at: <https://developers.google.com/machine-learning/practica/image-classification> [Accessed 15 April 2022].

[3] Mayfieldclinic.com. 2022. MRI, Magnetic Resonance Imaging Mayfield Brain & Spine Cincinnati, Ohio. [online] Available at: <https://mayfieldclinic.com/pe-mri.htm#:~:text=MRI%20can%20be%20used%20to%20detect%20brain%20tumors%2C%20traumatic%20brain,and%20the%20causes%20of%20headache.> [Accessed 15 April 2022].

[4] Medium. 2022. Top 6 Deep Learning Models You Should Master for Killer AI Applications. [online] Available at: <https://towardsdatascience.com/top-6-deep-learning-models-you-should-master-for-killer-ai-applications-13c7dfa68a3#:~:text=One%20of%20the%20most%20powerful,neurons%20with%20weights%20and%20biases.> [Accessed 15 April 2022].

[5] Seetha, J. and Raja, S., 2018. Brain Tumor Classification Using Convolutional Neural Networks. Biomedical and Pharmacology Journal, 11(3), pp.1457-1461.

[6] Irmak, E., 2021. Multi-Classification of Brain Tumor MRI Images Using Deep Convolutional Neural Network with Fully Optimized Framework. Iranian Journal of Science and Technology, Transactions of Electrical Engineering, 45(3), pp.1015-1036.

[7] Medium. 2022. Why Data should be Normalized before Training a Neural Network. [online] Available at: <https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d#:~:text=Among%20the%20best%20practices%20for,and%20leads%20to%20faster%20convergence.> [Accessed 15 April 2022].

[8] dendroid. 2022. K-means Clustering algorithm explained - dendroid. [online] Available at: <http://dendroid.sk/2011/05/09/k-means-clustering/> [Accessed 15 April 2022].

[9] www.javatpoint.com. 2022. K-Means Clustering Algorithm - Javatpoint. [online] Available at: <https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning> [Accessed 15 April 2022].

[10] 2022. [online] Available at: <https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn> [Accessed 15 April 2022].

[11] Medium. 2022. k-NN on Iris Dataset. [online] Available at: <https://towardsdatascience.com/k-nn-on-iris-dataset-3b827f2591e> [Accessed 15 April 2022].

[12] Matt Mazur. 2022. A Step by Step Backpropagation Example. [online] Available at: <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/> [Accessed 15 April 2022].

[13] Matt Mazur. 2022. A Step by Step Backpropagation Example. [online] Available at: <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/> [Accessed 15 April 2022].

[14] scikit-learn. 2022. 3.2. Tuning the hyper-parameters of an estimator. [online] Available at: <https://scikit-learn.org/stable/modules/grid_search.html> [Accessed 15 April 2022].

[15] scikit-learn. 2022. sklearn.neural_network.MLPClassifier. [online] Available at:

<https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html> [Accessed 15 April 2022].

[16] Medium. 2022. Accuracy, Precision, Recall or F1?. [online] Available at: <https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9> [Accessed 15 April 2022].

[17] scikit-learn. 2022. sklearn.metrics.confusion_matrix. [online] Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html> [Accessed 15 April 2022].

[18] Medium. 2022. Taking the Confusion Out of Confusion Matrices. [online] Available at: <https://towardsdatascience.com/taking-the-confusion-out-of-confusion-matrices-c1ce054b3d3e> [Accessed 15 April 2022].