Detecting Brain Tumours using Machine Learning

Samar Dikshit, Jerry Adams Franklin

Abstract

A brain tumour is an uncontrollable cell proliferation within the brain, and they are either benign or malignant. They can be detected through magnetic resonance imaging, however, this technique cannot be used for smaller tumours. In this project, we aim to develop machine learning classification models that can detect the presence of brain tumours given an MRI scan. The models include logistic regression, SVMs, Naive Bayes classifiers, decision trees, AdaBoost, and convolutional neural networks.

1 Introduction

Brain tumours are detected by analysing MRI scans. Each image needs to be individually analysed by oncologists or neurologists. This is an expensive, time-consuming process with a chance of human error. In the United States alone, over 80,000 brain tumours are diagnosed annually [1], out of which almost 24,000 are malignant [2].

Brain tumours have an overall five-year survival rate of only 36% [2]. However, early detection and diagnosis have been linked with a higher chance of survival [3]. Due to the advancements made in the field of computer vision and machine learning, it is now possible to build a system that can extract features from MRI scans, and predict whether a patient has a tumour or not.

The goal of our project is to build and compare a set of classifiers that can serve as an inexpensive, rapid, and accurate way to detect the presence of a brain tumour, hence increasing the likelihood of survival.

2 Technical Approach

To achieve our goal, we used various supervised learning classification techniques:

1. Logistic regression
2. Support vector machines (RBF kernel, sigmoid kernel, and linear kernel)
3. Decision trees
4. Naive Bayes classifiers
5. Adaptive Boosting
6. Convolutional neural networks

The reasons we added a CNN to our set of classifiers are:

1. A CNN convolves learned features with input data and uses 2D convolutional layers [4], hence making it a suitable model to classify MRI scans
2. CNNs need relatively less data preprocessing
3. CNNs work by extracting features from images, removing the need for a feature extraction pipeline, hence they are also independent of prior knowledge in feature design

To judge our models, we used the following scoring metrics:

1. Accuracy: A measure of the number of correctly predicted data points out of all data points.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives

2. Sensitivity: Sensitivity is a measure of the true positive rate. For a medical test such as diagnosing a tumour, it refers to the test's ability to correctly detect the patients who have that condition. Therefore, it is a more important metric than accuracy for our purpose.

$$Sensitivity = \frac{TP}{TP + FN}$$

3. ROC Curves and AUC: An ROC curve is a graph showing the performance of a classifier at all classification thresholds. It is a plot of the true positive rate (sensitivity) versus the false positive rate. Area Under the Curve (AUC) is a measure of the 2D area under the ROC curve. This value lies in the range [0, 1]. Scores closer to 1 are an indication of better prediction performance.

3 Experimental Results

3.1 Dataset

The data used to train our models consisted of 3,762 samples, each with 13 features and a binary label. The samples were well balanced, and had a 55-45 distribution (55% of the samples were class 0 i.e. not a tumour, and 45% were class 1 i.e. a tumour is present).

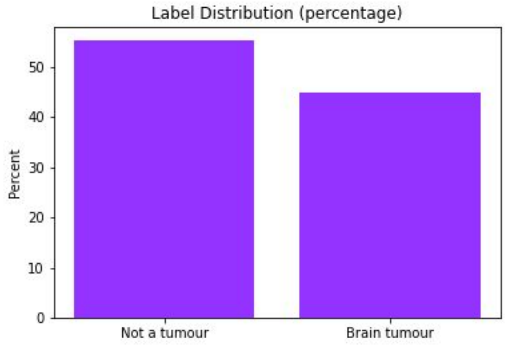

Figure 1: Label Distribution

The following features were present in the dataset:

1. Mean: The contribution of individual pixel intensity for the entire image
2. Variance: How each pixel varies from neighbouring pixels
3. Standard deviation: The deviation of measured values
4. Skewness: The measure of the lack of symmetry
5. Kurtosis: The peakedness of a distribution
6. Contrast: The difference in colour across the image
7. Energy: The rate of change of the colour of pixels over local areas
8. ASM (Angular Second Moment): The textural uniformity of an image
9. Entropy: The statistical measure of randomness
10. Homogeneity: How similar certain pixels of the image are
11. Dissimilarity: The numerical measure of how different two data objects are
12. Correlation: The result of moving a mask over the image and computing the sum of products at each location
13. Coarseness: The roughness of a texture

All features are continuous. Also included in the dataset were MRI scans corresponding to each sample.

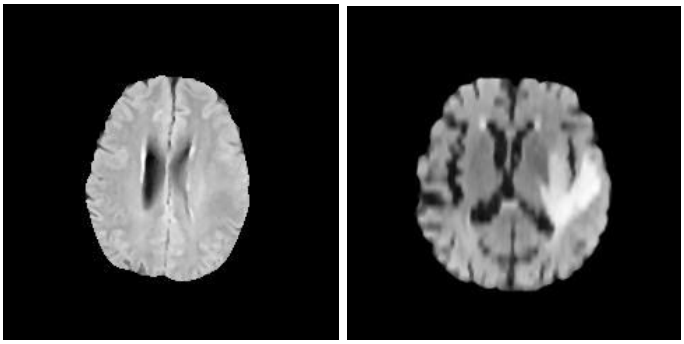

Figure 2: MRI scans of brains. Left: no tumour; Right: tumour present

3.2 Preprocessing and Exploratory Data Analysis

Before we could build models, we needed to preprocess our data to ensure that it can be used for machine learning. As a part of the preprocessing, we normalised the data so that each feature was on the same scale. This was done to ensure that no feature creates a bias in the models due to excessively large or small values. Using min-max normalisation, each feature was re-scaled to range from [0, 1].

After preprocessing the data, we conducted an exploratory data analysis with the following steps:

1. Feature value distributions: Plotting the overall distributions of each feature
2. Correlation analysis: To analyse the relationship between each pair of features
3. Class-wise distributions and density estimates: Plotting the class-wise distributions of each feature along with the kernel density estimates (KDE) for each distribution

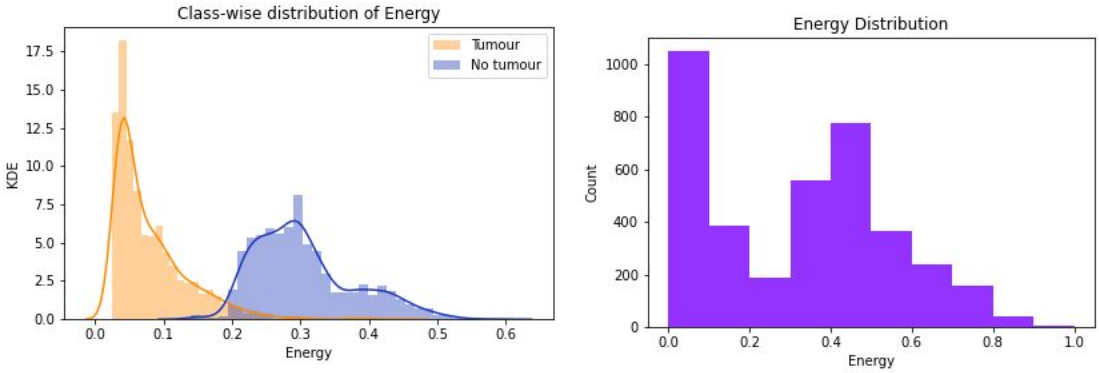

Figure 3: Distribution for *Energy*. Left: class-wise distribution with KDE; Right: overall distribution

3.3 Basic Models

For our first set of models, we used the following classifiers:

- Logistic regression
- SVMs: RBF and linear kernels
- Naive Bayes classifier

We split our dataset into a training set and a test set using an 80-20 split, and then trained each model using the training set. Using the test set, we obtained predictions and determined the accuracy of each model. Although each model had an accuracy of over 99%, we noted the following:

1. The models could be overfit
2. There might have been an imbalance in the split of the samples with and without tumours

3. We didn't use all the data available to us for training and testing, the 80% used for training was never used for testing, and similarly for the test set

As a result of these observations, we proposed the following solutions:

1. Add the following classifiers to our system: SVM with a sigmoid kernel, decision trees with Gini impurity, adaptive boosting models, and a convolutional neural network
2. Adjust the train/test split
3. Implement cross-validation
4. Implement feature selection
5. Hyperparameter tuning: Adjust hyperparameters such as depth of decision trees and number of estimators in adaptive boosting

3.4 The Effect of the Size of the Training and Test Sets

To observe the effect of the train/test split on each model, we trained and tested each classifier after adjusting the split size, and found the best performing model.

Starting from a test set size of 5% (i.e. training set size of 95%), we increased the test set size to 95% over 20 intervals (5%, 10%, … 90%, 95%) and measured the accuracy and sensitivity. We also set a threshold of 45%. Using this threshold, we found the best performing model without a restriction on the test set size, and with a restriction where test set size must be less than the threshold. This is because we cannot realistically use a model which has been trained on less than 50% of the data available.

The best model was the decision tree (depth = 15), with sensitivity and accuracy of 98.0% and 98.339% respectively, with a train/test split of 76% and 24%.

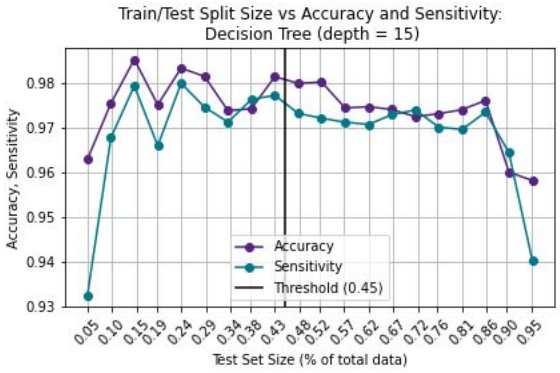

Figure 4: Decision Tree (depth = 15) Performance when Adjusting Train/Test Split

3.5 Implementing Cross-validation and Hyperparameter Tuning

By using cross-validation, we can use all the data available for training and testing. We implemented 10-fold cross-validation without repetition, and with 3 repetitions on our set of models.

Apart from letting us use all the data available, the other main advantages of cross-validation are that it reduces overfitting, and allows us to tune hyperparameters such as decision tree depth [5].

From Section 3.4 onwards, we have specifically used decision trees of depths 6 and 15. These were obtained by changing the depth parameter of the decision tree and comparing their performances during cross-validation. These trees had the best performance in terms of sensitivity and accuracy (Figure 5).

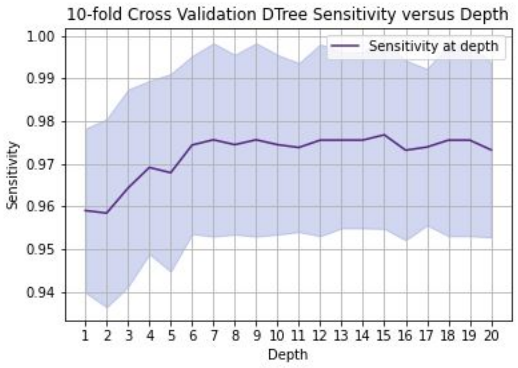

Figure 5: 10-fold Cross-validation Sensitivity vs Decision Tree Depth

The best model was a decision tree (depth = 15), with a sensitivity of 97.68%, and accuracy of 98.33%

3.6 Implementing Feature Selection

Since all 13 features are continuous, we used ANOVA F-value to get the 6 best features (~ 50% of the features). The 6 best features were *energy*, *homogeneity*, *entropy*, *ASM*, *dissimilarity*, and *skewness*.

We then applied 5-fold cross-validation to the models and plotted the ROC curve.

The best model was still a decision tree (depth = 15), however, its sensitivity and accuracy fell to 96.89% and 97.39% respectively, with a mean ROC AUC of 0.973 ± 0.003. Hence, reducing the number of features doesn't lead to better performance for our data and classifiers.

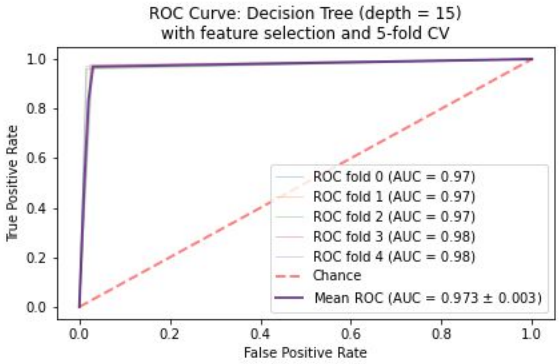

Figure 6: ROC Curve for Decision Tree (depth = 15) with 6-best Feature Selection and 5-fold Cross-validation

3.7 Adaptive Boosting

AdaBoost is a boosting technique that is used as an ensemble method in machine learning. A classifier is fit on the data, after which additional copies of the same classifier are fit on the same dataset with the weights of incorrectly classified samples adjusted such that the subsequent classifiers focus more on the harder cases. The SAMME àlgorithm is used for boosting [6].

The two primary hyperparameters in AdaBoost are the number of estimators, and the ensemble learner used. Our classifier uses a decision tree as the ensemble learner. The performance of AdaBoost was compared for models with a varying number of estimators and decision tree ensemble depths.

3.7.1 AdaBoost and Cross-validation

Similar to what was done in Section 3.5 with the basic models, we implemented cross-validation (without repetition, and with 3 repetitions) for our default AdaBoost model (50 decision stump i.e. depth = 1

estimators). Cross-validation with repetition led to better performance, with a sensitivity of 97.9%, and accuracy of 98.75%.

3.7.2 AdaBoost and Feature Selection

Using the same default model described in Section 3.7.1, we trained our model using the 6 best features and 5-fold cross-validation (Section 3.6). As earlier, the performance dropped, resulting in a sensitivity of 96.84%, accuracy of 97.89%, and mean ROC AUC of 0.990 ± 0.001.

3.7.3 Changing the Number of Estimators

Sections 3.7.1 and 3.7.2 used the default AdaBoost model. The first hyperparameter we adjusted to improve performance was the number of estimators i.e. number of decision trees/stumps. Using 3 repetitions of 10-fold cross-validation, we compared the behaviour of models that use between 10 and 350 decision stumps as the ensemble learner.

The model with the highest accuracy of 98.85% had a sensitivity of 98.02% and 100 estimators, while the model with the highest sensitivity of 98.1824% had an accuracy of 98.85 % and 350 estimators. However, the latter classifier took long to train, hence we selected the model with 200 estimators. This model had similar performance scores (98.1818% sensitivity, 98.83% accuracy), and took less time to train.

3.7.4 Changing the Ensemble Learner

The default AdaBoost model uses 50 decision stumps as the ensemble. From Section 3.5, we know that trees of depths 6 and 15 have the best performance. From Section 3.7.3, we also know that using 200 estimators leads to a well-performing AdaBoost model. Putting these together, we compared models with 50 (the default) and 200 estimators, with each using trees of depths 1 (the default), 6, and 15. Each model was trained using 10-fold cross-validation with 3 repetitions. The results can be seen in Figure 7.

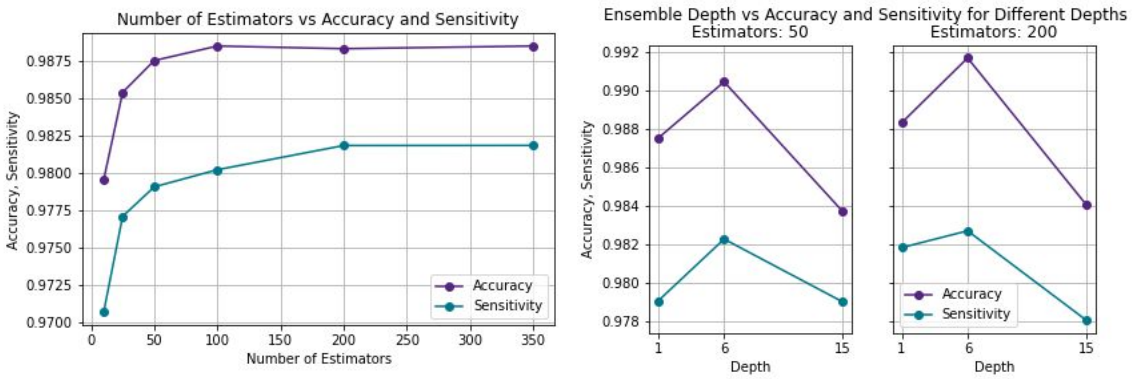

Figure 7: Performance Effect by Changing Number of Estimators and Ensemble Depth

The best model had 200 estimators, each using decision trees of depth 6. The sensitivity of the model was 98.27%, with an accuracy of 99.17%. This is the best performing model out of all so far.

3.8 Convolutional Neural Network

The final model we created was a CNN. Unlike the previous models in Section 3, we don't need to extract features and feed them to the network. MRI scans (Figure 2) can be converted to NumPy arrays, which are then used as the input of the network.

The first CNN we created had 7 layers: 2 convolutional, 2 pooling, 2 dense, and 1 flattening layer. This resulted in heavy overfitting.

To overcome overfitting, we added dropout regularisation for our next model. The second model had 12 layers: 3 convolutional, 3 pooling, 2 dropout, 3 dense, and 1 flattening layer. Using the ReLU activation

function, classification cross-entropy loss, and Adam optimisation, we achieved a sensitivity of 96.6%, and accuracy of 94.9% during training over 50 epochs.

Cross-entropy calculates a score that summarises the average difference between predicted probabilities for class 1 (tumour present), after which the score is minimised. Adam optimisation computes adaptive learning rates for each parameter of the model, and also stores an exponentially decaying average of past gradients [7].

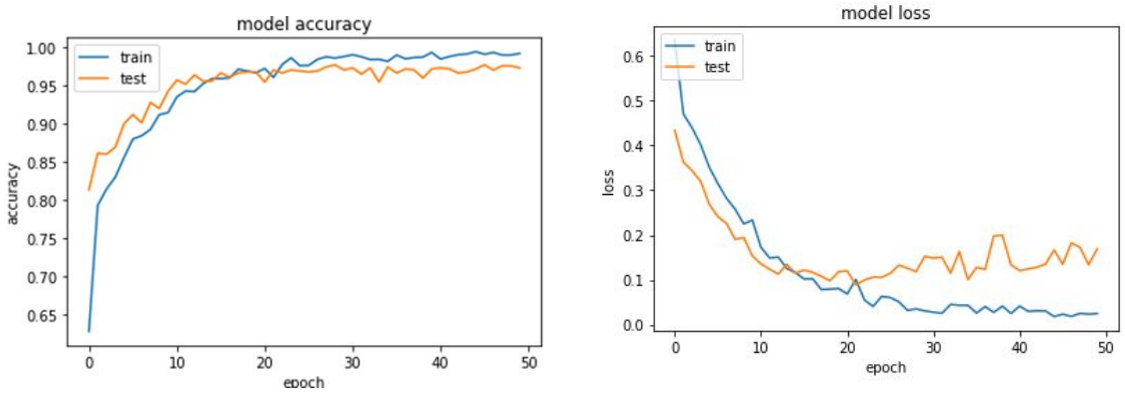

Figure 8: CNN Accuracy (left) and Loss (right) over 50 Epochs

The final CNN we tried to create was with applying batch normalisation, however, we did not have the computation power required to implement this. While running it for 100 epochs, its accuracy had only reached 70%, and was increasing very slowly, before crashing. However, we believe that theoretically, this should perform better than our best CNN.

In each CNN, we started off with 64 filters in the first layer, followed by 128 filters in the second layer. This took days to train, and gave only a slightly higher accuracy. We replaced this setup with 32 filters for the first layer, and 64 filters for each subsequent layer. We were then able to train the model within a few hours, with an accuracy difference of only 0.05%, hence we kept this configuration.

3.9 Results

The AdaBoost classifier with 200 estimators and decision trees of depth 6 (Section 3.7.4) was the best model in terms of performance only (98.27% sensitivity, 99.17% accuracy). However, its drawbacks are that it requires feature extraction and processing, and a grid search for hyperparameter tuning. The CNN has a slightly lower sensitivity (96.6%) and accuracy (94.9%), but doesn't need any feature engineering. However, it takes longer to develop as compared to AdaBoost.

4 Statement of Contributions

1. Samar Dikshit: Preprocessing; EDA; Tuning basic models; Adaptive Boosting
2. Jerry Adams Franklin: EDA; Creating initial basic models; Convolutional neural net

References

[1] “Brain Tumour Facts & Figures” [Online]. Available: https://blog.braintumour.org/brain-tumour-facts-figures-may-2018-incidence-mortality-and-survival-in-2018 [Accessed: November 2020]

[2] “Brain Tumour” [Online]. Available: https://www.cancer.net/cancer-types/brain-tumour/statistic [Accessed: November 2020]

[3] “Early Detection Can Be Key to Surviving Brain Tumour” [Online]. Available: https://weillcornellbrainandspine.org/early-detection-can-be-key-surviving-brain-tumour [Accessed: November 2020]

[4] B. Darane, P. Rajput, Y. Sondagar and R. Koshy, "Recognizing Presence of Hematological Disease using Deep Learning," *2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India, 2019, pp. 310-315, doi: 10.1109/I-SMAC47947.2019.9032639

[5] S. Yadav and S. Shukla, "Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification," *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, Bhimavaram, 2016, pp. 78-83, doi: 10.1109/IACC.2016.25

[6] J. Zhu, S. Rosset, H. Zou and T. Hastie, “Multi-class AdaBoost”

[7] “An overview of gradient descent optimization algorithms” [Online]. Available: https://ruder.io/optimizing-gradient-descent/index.html#adam [Accessed: December 2020]