Brain Tumor Classification with Stacking Method

Cristian Daniel Rodríguez Vázquez, A01209306, Tecnológico de Monterrey Campus Querétaro

***Abstract* – This document presents the implementation and explanation of an analysis of the use of stacking method and its performance with a brain tumor classification problem.**

I. INTRODUCTION

A brain tumor, known as an intracranial tumor, is an abnormal mass of tissue in which cells grow and multiply uncontrollably, seemingly unchecked by the mechanism that control normal cells. To date, more than 150 types of brain tumors have been detected; however, they can be grouped into two main groups called primary and metastatic [1].

The incidence of brain tumors has been increasing in all ages in recent decades. Metastatic tumors to the brain affect nearly one in four patients with cancer, or an estimated 150,000 people a year.

There are various techniques used to obtain information about tumors. The most used is Magnetic Resonance Imaging (MRI), which produces a large amount of 2D images. The detection and classification of brain tumors generated by manual procedures is costly in both effort and time. Therefore, it is worthwhile to generate an automatic detection and classification procedure in order to obtain an early diagnosis and thus have a faster response in the treatment to improve the survival rate of patients [2].

II. STATE OF THE ART

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model. By combining the output of different models, ensemble modeling helps to build a consensus on the meaning of the data. In the case of classification, multiple models are consolidated into a single prediction using a frequency-based voting system. Ensemble models can be generated by using a single algorithm with numerous variations, known as homogeneous ensemble, or by using different techniques, known as a heterogeneous ensemble [3].

As shown in figure 1, the idea of stacking method is to train several different weak learners and combine them by training a meta model to output predictions based on the multiple predictions returned by these weak models [4].

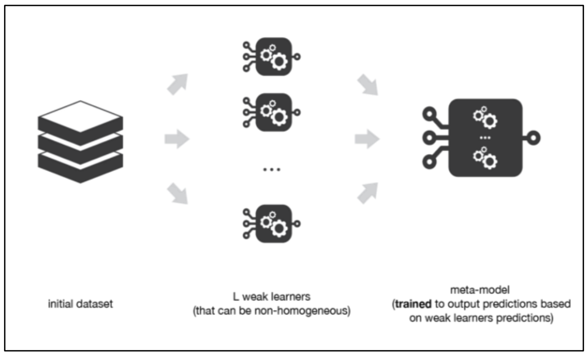

initial dataset
L weak learners
(that can be non-homogeneous)
meta-model
(trained to output predictions based on weak learners predictions)

Fig 1. Stacking model implementation diagram [4].

III. DATASET

The dataset comes from Kaggle [5], that contains a database of 3206 brain MRI images, which are separated in four different categories: no tumor, glioma tumor, meningioma tumor, and pituitary tumor. Figure 2 shows a sample image for each category, respectively.

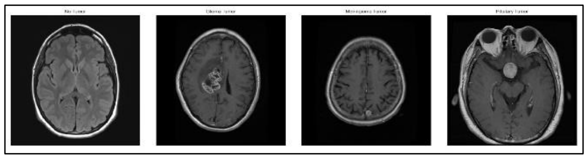


Fig 2. Sample images from each category.

In order to obtain our final dataset to train the models, it was necessary to do an image preprocessing. Machines store images in the form of a matrix of numbers, the size of this matrix depends on the number of pixels we have in any given image. The pixels values denote the intensity or brightness of the pixel, smaller numbers represent black and larger numbers represent white. For gray scale images, as in this case, the matrices are two- dimensional.

After obtaining the pixel matrices, five first order and seven second order features were obtained for each image. For the first order features, basic statistical analysis was implemented in the pixel's matrices:

- *Mean*: is the average or the most common value in the pixel's matrix.
- *Variance*: measures the average degree to which each point differs from the mean.
- *Standard Deviation*: looks at how spread out a group of numbers is from the mean.
- *Skewness*: measures the lack of symmetry.
- *Kurtosis*: defines how heavily the tails of a distribution differ from the tails of a normal distribution.

In order to obtain the second order characteristics, the grey level co-occurrence matrix (GLCM) was used. GLCM is a matrix that represents the relative frequencies of a pair of grey level present at a certain distance apart and a particle angle. In this case one pixel of distance and angles of 0°, 45°, 90°, and 135° were used. In figure 3 is shown how the GLCM is determined.

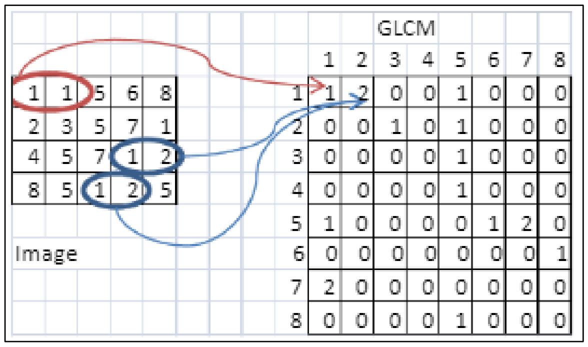


Fig 3. GLCM calculation example [6].

The second order features obtained from the greycomatrix are the next ones:

- *Contrast:* represents the difference in luminance across the image.
- *Entropy:* measure of randomness.
- *Dissimilarity*: is a numerical measure of how different two data objects are.
- *Homogeneity*: expresses how similar certain elements of the image are.
- *ASM*: is a measure of textural uniformity of an image.
- *Energy*: the rate of change in the brightness of the pixels over local areas.
- *Correlation*: gives information about how correlated a pixel is to its neighboring pixels.

IV. MODEL PROPOSAL

As mentioned previously, stacking runs multiple models simultaneously on the data and combines those results to produce a final model. The previously mentioned, can be schematically illustrated in figure 4.

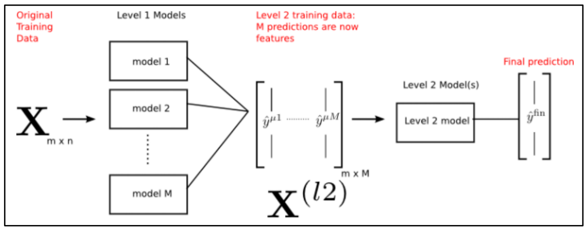


Fig 4. Stacking model implementation example [7].

The general idea of how the model works is as follows [7]:

1. Initial training data has 2565 observations, and 12 features.
2. There are three different weak learner models that are trained on the training data.
3. Each weak learner provides predictions for the outcome which are then cast into a second level training data, which is now 2565 x 3.
4. A meta model is trained on this second level training data to produce the final predictions.

For this implementation the three weak learner models used were k-nearest neighbors, decision trees and naive bayes, and for the meta model k-nearest neighbors were used again.

K-Nearest Neighbors

The KNN algorithm assumes that similar things exist in proximity, so it classifies new data points based on their position to nearby data points.

In figure 5, the data points have been classified into two classes, and a new data point with unknow class is added to the plot. Using KNN algorithm the category of the new data point can be predicted based on its position to the existing data points. For example, if *k* is set to 3, the outcome of selecting the three nearest data points returns two class B and one class A, so the prediction for the new data point will be class B. On the other hand, if *k* is set to 6, the prediction will be class A. The chosen number of neighbors identified is crucial in determining the results [3].

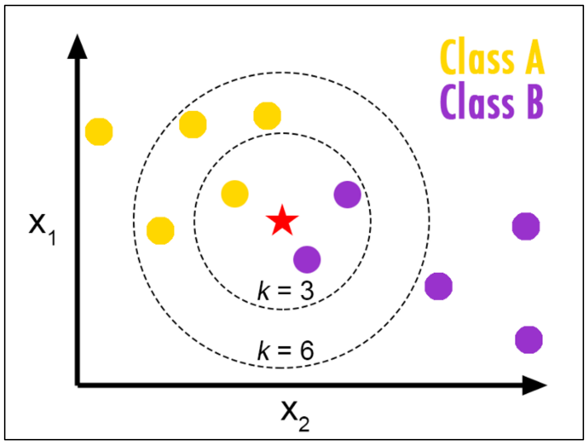


Fig 5. Stacking model implementation example [8].

In general, the Euclidean distance is used to determine the distance between each data point, which is calculated as follows:

$$d(q,p) = \sqrt{\sum_{i=1}^{k}(q_i - p_i)^2}$$

Decision Trees

Decision trees are transparent and easy to interpret. Classification trees predict categorical outcomes using numeric and categorical variable as input. The trees start with a root node that acts as a starting point and is followed by splits that produce branches. The branches then link to leaves which form decision points, and this process is repeated using the data points collected in each new leaf. The final categorization is produced when a leaf no longer generates any new branches and results in what is called a terminal node [3]. In the figure below a decision tree which predicts how to make the journey to work is shown.

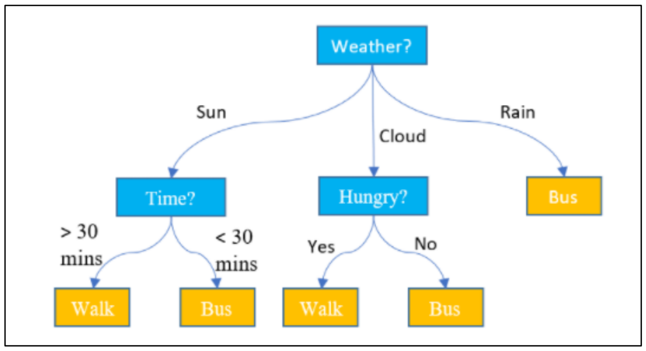


Fig 6. Example of a decision tree [9].

To build the decision tree in an efficient way entropy is used for selecting the best splitter. The objective is that at each layer the data can be more homogeneous than the previous partition. The mathematical formula of entropy is as follows:

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i$$

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain. The information gain is calculated as follows:

$$Gain(S,x) = E(S) - E(x)$$

In resume, in order to obtain the best split for our decision tree we need to follow the next steps:

1. Determine the system entropy.
2. Determine sub entropy for each feature.

3. Calculate the information gain of each feature.
4. Choose as the next node the feature that obtain the highest information gain.
5. Repeat until get the desired number of nodes.

Naive Bayes

A Naive Bayes classifier is a probabilistic machine learning model that is used for classification task. This algorithm is based on the Bayes theorem.

Using Bayes theorem, we can find the probability of y happening, given that X has occurred. Being X the evidence and y the hypothesis. The assumption is that the features are independent, meaning that the presence of one feature does not affect the other. The Bayes theorem is described with the next formula [10]:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Where y represents the label and X the features. By substituting the features for X and expanding using the chain rule we get:

$$P(y|x_1, \dots, x_n) = \frac{P(x_1|y) \dots P(x_n|y)P(y)}{P(x_1) \dots P(x_n)}$$

For all entries in the dataset, the denominator does not change, therefore, it can be removed, and a proportionality can be introduced:

$$P(y|x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$$

Finally, it is needed to find the class y with maximum probability:

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

V. TEST & VALIDATION

The initial dataset was divided into training and testing data, being 80% for training and 20% for testing. The models were created from scratch for this implementation.

The first step was to train the weak learners, for the k-nearest neighbor model a k of 5 was used, and for decision tree the maximum depth assigned was 7. The obtained decision tree is seen on figure 7.

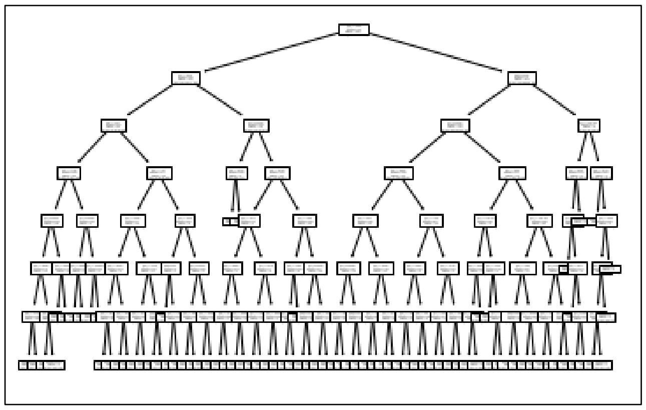
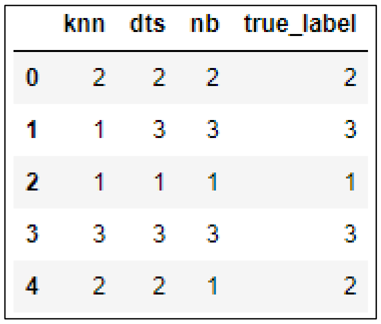


Fig 7. Obtained decision tree with max_depth=7.

After training the weak learners, predictions were made to create our second dataset with which the meta model was trained. Figure 8 shows the 5 first rows of the second dataset.



```
   knn  dts  nb  true_label
0    2    2   2           2
1    1    3   3           3
2    1    1   1           1
3    3    3   3           3
4    2    2   1           2
```

Fig 8. First 5 rows of the meta model training dataset.

Finally, the meta model was trained, and our stacking model is ready for making new predictions.

To evaluate the performance of the models implemented by hand, cross validation was implemented. The dataset was divided into five different K folds and the average of the recorded scores were saved. In the next table is shown the accuracy of the models:

Model	Accuracy
K-Nearest Neighbor	58.28%
Naive Bayes	55.65%
Decision Tree	73.21%
Stacking	66.98%

Table 1. Models accuracies.

Finally, models from scikit-learn library were used to make a comparison between these and the ones elaborated from scratch. The comparison is shown in the following table:

Model	From scratch	Scikit-learn
K-Nearest Neighbor	58.28%	59.55%
Naive Bayes	55.65%	54.02%
Decision Tree	73.21%	68.75%
Stacking	66.98%	65.21%

Table 2. Models comparison.

VI. CONCLUSIONS

The final stacking model obtained an accuracy of 67%, while the decision tree model obtained 73% accuracy, this is because the other two weak models fed into the meta model have a very low accuracy and the meta model has an accuracy of approximately the average accuracy among the three weak learner models. Possibly a better implementation would have been with Random Forest.

Although slightly better accuracy was obtained with the models implemented from scratch, the library models are better as they take much less time to train and predict new results.

A better implementation for brain tumors detection and classification could be using deep learning, as these models generally have better accuracy and would help identify the location of the tumor.

VII. REFERENCES

[1] AANS. (N.D.). *Brain Tumors*. Retrieved from https://www.aans.org/en/Patients/Neurosurgical-Conditions-and-Treatments/Brain-Tumors#:~:text=A%20brain%20tumor%2C%20

known%20as,mechanisms%20that%20control%20normal%20cells.

[2] Díaz, F., Martínez, M., Antón, M., & González D. (2021). *A Deep Learning Approach for Brain Tumor Classification and Segmentation Using a Multiscale Convolutional Neural Network.* Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7912940/

[3] Theobald, O. (2017). *Machine Learning for Absolute Beginners Second Edition.* Oliver Theobald.

[4] Rocca, J. (2019). *Ensemble methods: bagging, boosting, and stacking.* Retrieved from https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

[5] Kaggle. (N.D.). *Brain Tumor Classification (MRI).* Retrieved from https://www.kaggle.com/sartajbhuvaji/brain-tumor-classification-mri

[6] Singh, S., Agarwal, S., & Srivastava, D. (2017). *GLCM and Its Application in Pattern Recognition.* Retrieved from https://www.researchgate.net/publication/320176123_GLCM_and_its_application_in_pattern_recognition

[7] KDnuggets. (N.D.). *Stacking Models for Improved Predictions.* Retrieved from https://www.kdnuggets.com/2017/02/stacking-models-imropved-predictions.html

[8] Beisel, A. (2020). *KNN (K-Nearest Neighbors Classifier from Scratch).* Retrieved from https://amybeisel.medium.com/knn-k-nearest-neighbors-classifier-from-scratch-2087561010dc

[9] Hoare, J. (N.D.). *What is a Decision Tree?* Retrieved from https://www.displayr.com/what-is-a-decision-tree/

[10] Gandhi, R. (2018). *Naive Bayes Classifier.* Retrieved from https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

Code References

AskPython. (N.D.). *K-Nearest Neighbors from Scratch with Python.* Retrieved from https://www.askpython.com/python/examples/k-nearest-neighbors-from-scratch

Scikits-image. (N.D.). *Module: feature.texture.* Retrieved from https://scikit-image.org/docs/0.7.0/api/skimage.feature.texture.html

Brownlee, J. (2021). *Stacking Ensemble Machine Learning With Python.* Retrieved from https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/

Normalized Nerd. (2021, February 26). *Naïve Bayes Classifier in Python (from scratch!)*

[Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=3I8oX3OUL6I

Python Engineer. (2019, November 22). *Decision Tree in Python Part 2/2 – Machine Learning From Scratch 09 – Python Tutorial* [Video]. YouTube. Retrieved from https://www.youtube.com/watch?v=Bqi7EFFvNOg