MIA Lab Report

Hypothesis 4 - Is the random forest classifier the best suiting for brain segmentation?

Julian Rösch
BME student
julian.roesch@students.unibe.ch

Timo Kofmehl
BME student
timo.kofmehl@students.unibe.ch

***Abstract*—The Hypothesis of this project defines the random forest classifier as the best suiting classifier for brain segmentation. After several tests with different methods and parameters, random forest can be confirmed as one of the best choices. The gradient boost classifier is very promising too. With some more time fine tuning the parameters, even better results are expected.**

I. Introduction

The aim of this project was to automatically detect and determine the different brain regions in a MRI scan. An typical MRI scan of a brain contains of a cube of aprox. 200x200x200 voxels. This results in 200 images each for the sagital, coronal and transverse plane. For a doctor it can take several hours to look through all this images and to classify the different brain region manually. The goal of the current research in the MIA field is to automate this process.
A typical medical image analysis pipeline consists of the following 5 tasks. Registration – Pre-processing – Feature Extraction – Classification – Post-processing. The first two steps prepare the image for the following classification. Skull-stripping, a part of the pre-processing, for example reduces the data to be classified by masking the images with an atlas brain image. Our group focused on the classification part, the segmentation of the brain regions. Given at the beginning of the lab is the random forest classifier. Is this the best suiting classifier for brain segmentation? Are there any more accurate or faster alternatives?
In this project the brain was segmented into 5 parts:
Amygdala, Grey Matter, Hippocampus, Thalamus, and White Matter

II. Evaluation methods

The first step of the project was to complete the pre-processiong. A general structure of the code was already given. We completed the given ToDo's and implemented the calculation of the Dice and Hausdorff distance. Because our job is the comparison of different classifiers, we decided to store the results of the pre-processing as pickle files. The files data_train and labels_train are stored in the folder ..\bin. This saves on one hand a lot of calculation time for each new test. On the other hand there is a much better comparison if the input data is always the same.
Furthermore there is a file in this folder called "ShowResults". This file was developed to visuallize and plot our results.
To evaluate the different classifiers we used four methods; the Dice coefficient, the Hausdorff distance, the runtime, and a visual evaluation of the MRI images with the generated segmentation. These methods are commonly used for the evaluation of segmentation.

A. Dice-coefficient

The Dice coefficient, also known as the Sørensen–Dice coefficient, is a similarity metric commonly used in segmentation evaluation to measure the overlap between two segmented regions.

For two sets of points A (predicted segmentation) and B (ground truth segmentation) in an image, the Dice coefficient $D(A, B)$ is computed as:

$$D(A, B) = \frac{2 \times |A \cap B|}{|A| + |B|}$$

Here, $|A|$ and $|B|$ denote the cardinalities of sets A and B, respectively. $|A \cap B|$ represents the number of common points between A and B. The Dice coefficient ranges from 0 to 1, where 1 indicates perfect overlap between the two segmentations.

The Dice coefficient is particularly useful for evaluating segmentation tasks where accurate delineation of boundaries or regions is crucial. Its range from 0 to 1 allows for intuitive interpretation, making it a widely adopted metric for comparing and benchmarking segmentation algorithms. [1]

B. Hausdorff distance

The Hausdorff distance serves as a metric to quantify dissimilarity between two sets of points, often employed in segmentation evaluation to assess the similarity between predicted and ground truth segmentations.

Given two sets of points A (representing the predicted segmentation) and B (representing the ground truth segmentation) in a metric space X, the Hausdorff distance $H(A, B)$ is calculated as:

$$H(A, B) = \max \left\{ \sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b) \right\}$$

Here, $d(a, b)$ denotes the distance metric used to measure the distance between points a and b. The calculation involves

finding the maximum distance between the closest points in A and B, and determining the greatest dissimilarity between the two sets.

The Hausdorff distance, by measuring the maximum discrepancy between corresponding points in the segmentations, serves as a valuable tool for objectively evaluating the performance of segmentation algorithms. [2]

C. Runtime

An other value to compare the classifiers is the time to run the program. Runtime in segmentation algorithms is vital for timely diagnosis and swift patient care in clinical settings. Optimizing for runtime enhances practical applicability.

III. Classifier methods

In this section, the tested classifiers are described more precisely.

A. Random Forest Classifier

The initial classifier of the hypothesis is the random forest classifier. The Random Forest algorithm is an ensemble learning technique used for classification and regression tasks. It operates by building multiple decision trees during training and aggregating their predictions to make a final prediction.
Let X represent the training dataset with N samples and M features. To create a Random Forest, the following steps are performed:

1) Randomly select K samples from the dataset with replacement (bootstrap sampling) to create each tree in the forest.
2) For each tree:
   a) Randomly select a subset of features of size m ($m < M$).
   b) Construct the decision tree using the selected samples and features.
3) Aggregate the predictions from all trees for classification (or average predictions for regression).

During prediction, the Random Forest collects the output from each tree and determines the final class by a majority vote (for classification) or averaging (for regression).
Random Forests are robust against overfitting and perform well on large datasets. They can handle missing values and maintain accuracy even with unbalanced datasets. [3]

B. K-Nearest Neighbor Classifier

The k-Nearest Neighbors algorithm, further written as kNN, is a non-parametric supervised learning method. The user defined constant k is the most important value of this algorithm. Starting at a query point, the k value describes how many training samples are taken into account for the description of the query point. In the used sklearn function, the value k is defined as "n_neighbors". Further input parameters for the function we changed were n_jobs, p, and leaf_size. The p parameter defines the used distance measurement from the green dot toe the red and blue objects (talking for the example above). If p is equal to 1, the manhattan-distance will be used. If p is not defined the euclidean-distance is used.
The leaf_size parameter can affect the speed of the construction and the required memory.
N_jobs defines the number of parallel jobs that are running for the neighbors search. Default value is 1. If n_jobs is set to -1, all possible processors are used. [4]

C. Decision Tree Classifier

A Decision Tree classifier is a non parametric supervised learning method. The values of a target variable is predicted with the help of simple decision rules. Starting at the first decision, for each of the two possible solutions a new decision is set. Graphically this resembles an genealogical tree. The function stops either after a defined maximum depth of the tree (aka maximum of generations) or until all leaves are pure.
In the used sklear function, again, many parameters can be set. The criterion can be set to "gini", "enropy" or "log_loss" ,each of them is a different function to measure the quality of a split. The default value is "gini". The strategy to chose the split at each node can be done in two different ways, "best" which is the default or "random".
Some more parameters were tested, like the "max_depth" or the "randstate". [5] [6]

D. Gradient Boost Classifier

The Gradient Boost classifier is an ensemble method that sequentially builds decision trees to correct errors made by the previous trees. It focuses on minimizing prediction errors by fitting new trees to the residuals of the previous ones. This creates a powerful model by combining the predictions of multiple trees, resulting in robust classification performance while handling complex data and minimizing overfitting.

E. Naive Bayes Classifier

The Naive Bayes classifier is a probabilistic classifier based on Bayes' theorem. It's often used in segmentation tasks such as document classification.
The algorithm assumes that the features are conditionally independent given the class label. For a document D represented by a set of features $F = \{f_1, f_2, ..., f_n\}$, and classes $C = \{c_1, c_2, ..., c_k\}$, the probability of a document belonging to a class c_i can be calculated using Bayes' theorem:

$$P(c_i|D) = \frac{P(D|c_i) \cdot P(c_i)}{P(D)}$$

where:

- $P(c_i|D)$ is the posterior probability of class c_i given document D.
- $P(D|c_i)$ is the likelihood of observing document D given class c_i.
- $P(c_i)$ is the prior probability of class c_i.
- $P(D)$ is the probability of observing document D.

The naive assumption of feature independence allows us to simplify the probability term:

$$P(D|c_i) = P(f_1, f_2, ..., f_n|c_i) \approx \prod_{j=1}^{n} P(f_j|c_i)$$

To make predictions, the class with the highest posterior probability is selected for the given document. Several variations of Naive Bayes exist, such as Multinomial Naive Bayes or Gaussian Naive Bayes, suitable for different types of data distributions. [7]

F. others

Beside the methods above described in detail some other functions, for example the Gaussian Process Classifier, had been implemented and tested. The results of this method was not as good as expected, wherefore there is no closer focus on this method. The implementation of the RGM SVM was not successful and did not lead to any results.

IV. Results

Multiple Tests were made with each of the listed methods above. In this section, the best result for each classifier is displayed together with the used parameters to achieve these results.

A. Random Forest Classifier

Using the grid-search to adjust the hyper-parameters did not perform marginally better than the already given set of parameters. The best set of hyper-parameters from the grid-search were: ('max_depth'= None, 'min_samples_leaf'= 1, 'min_samples_split'= 2, 'n_estimators'= 100)
As for performance, the Random Forest classifier performed overall as good or better than others, especially for smaller regions (see Figure 3 and 4)

B. K-Nearest Neighbor Classifier

The best result for a kNN found in this project was with the following parameters:
KNeighborsClassifier(n_neighbors=2, n_jobs = -1, leaf_size = 4)
The Thalamus achieved a Dice coefficient of almost 0.68 as a median value. The White Matter was around 0.5 and the other three brain regions had a dice coefficient in median of 0.4. The Hausdorff-distance of all five regions was with this parameters not larger than 7.
The runtime for tests with this method was around 30-35 minutes.

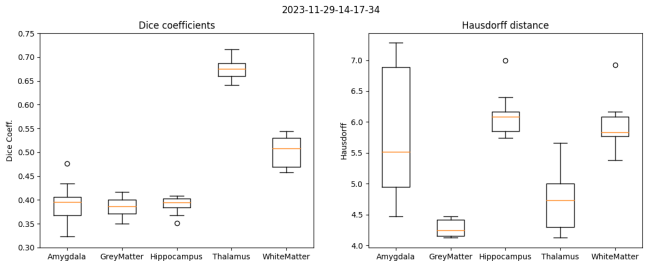

Fig. 1. Dice Coefficients of kNN Method

C. Decision Tree Classifier

The best result for the Decision Tree classifier was achieved with the criterion = 'entropy' and all the other parameters set to default values. Both, the dice coefficient for the thalamus and the white matter, scored in this example between 0.65 and 0.7. The other three regions were at values around 0.4. A big difference with this classifier was the Hausdorff-distance. Only the white and grey matter had values of under 7, comparable to the kNN classifier. Amygdala, Hippocampus, and Thalamus were all way larger.
The runtime to execute one test was on ubelix [8] below 4 minutes.

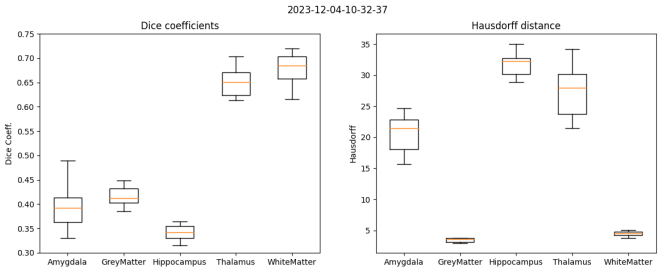

Fig. 2. Dice Coefficients of Decision Tree Method

D. Gradient Boost

Running a gridsearch to find the best hyper-parameters resulted in this set of parameters: ('learning_rate'= 0.1, 'max_depth'= 7, 'min_samples_split'= 2, 'n_estimators'= 150). Using these parameters, the Dice coefficients and the Hausdorff distances were in the same range as for the Random-Forest Classifier. But performing much worse considering the Hausdorff distance at the Amygdala, as seen in Figure 4.

E. Naive Bayes

The naive Bayes algorithm has no hyper-parameters to adjust. Considering the run-time, Dice coefficient and Hausdorff distance, the results perform similar compared to the other algorithms. Naive Bayes works best for larger areas, as seen in Figure 3.

F. Comparison

For a better overview of the results, figure 3 and figure 4 are showing the Dice coefficient respective the Hausdorff distance for the following four classifiers: Naive Bayes, kNN, Random Forest, and Gradient Boost. These four classifiers were chosen because of their most promising results. Figure 5 shows the measured runtime for three individual tests with each classifier. The Decision Tree and the Naive Bayes were all under 5 minutes. The other three displayed classifiers needed between 30 and 40 minutes to complete the calculations.

V. Discussion

The hypothesis of this project was to check if the Random Forest classifier is the best suiting classifier for brain segmentation. In figure 3 are the dice coefficients of the four

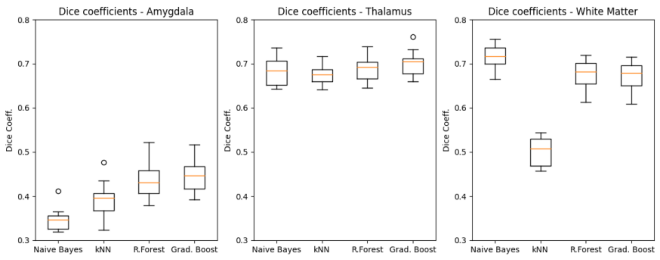

Fig. 3. Dice coefficients of different methods compared

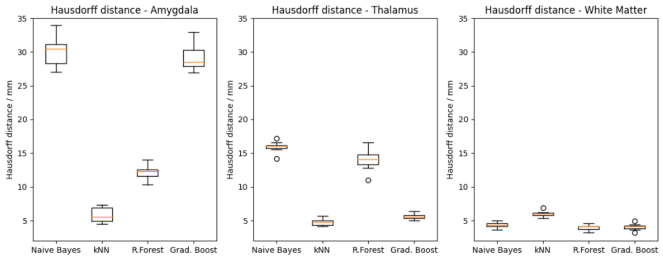

Fig. 4. Hausdorff distances of different methods compared

Fig. 5. Run-time of the different classifier algorithms

best methods displayed side by side. Our tests showed that the Random Forest classifier is certainly comparable and as good as the other three displayed methods. The segmentation of the thalamus was for all four classifiers identically good. The kNN classifier has a bad output in the classification of white matter. The amygdala segmentation was for each tested classifier strictly lower, Random Forest and Gradient Boost still with the best results.

Only looking at the runtime, the Decision Tree and the Naive Bayes classifier were by far the fastest method with only 3–4 minutes for the whole process. But the results of the Decision tree were not sufficient. Naive Bayes achieved good results with its fast calculation time. KNN, Random Forest, and Gradient Boost had all more or less the same process time of 30 to 40 minutes. This measurement was therefore not considered as important as the dice-coefficient and the Hausdorff-distance.

Overall, we saw the most promising results for the Gradient Boost classifier. With some more parameter optimization, much better results could certainly be achieved.

VI. Outlook

By taking a closer look at the ground truth, we detected a lot of voxels, that do not appear to be connected to the larger segments. This seems to be a problem occuring in smaller brain regions (e.g. Amygdala). All of our chosen classifiers tried to combine larger areas to form segments. This resulted in over-segmentation in many places. One question we had was whether this ground truth was really correct. For us it seemed almost more realistic that the individual regions in the brain should be more coherent. For further research on this project, the ground truth would certainly have to be examined and tested again.

In this project, we excluded the post-processing part of the image analysis pipeline. With an optimal and well adapted post-processing, improved results for the dice score and the Hausdorff-distance could be certainly expected.

The use of neural networks was excluded from this project. The workload to implement this methods would have been beyond the scope of this project.

Another point we figured out too late was the data management for this project. All the results from the individual tests were stored in folders only containing the date. The files itself, inside the folders, were all the same. We then created a text file to get an overview of the methods and parameters used in the individual tests. For a future projects it would be certainly advisably to define a better system beforehand to get a better overview of our results.

References

[1] Zou, K. H., Warfield, S. K., Bharatha, A., Tempany, C. M., Kaus, M. R., Haker, S. J., Jolesz, F. A. (2004). Statistical validation of image segmentation quality based on a spatial overlap index. *Academic Radiology*, 11(2), 178-189.

[2] Jeff Henrikson. Completeness and Total Boundedness of the Hausdorff Metric. *Journal of Mathematical Analysis and Applications*, 234(1):71-79, 1999.

[3] Breiman, L. (2001). *Random forests*. Machine Learning, 45(1), 5-32.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html, 11.12.2023

[5] https://scikit-learn.org/stable/modules/tree.html, 16.12.2023

[6] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier, 16.12.2023

[7] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[8] https://hpc-unibe-ch.github.io/, 29.12.2023