MSE | MASTER OF SCIENCE IN ENGINEERING

TSM – Machine Learning

PW 09: Decision Trees

19 November 2023

Daniel Ribeiro Cabral – Bastien Veuthey

MSE | Data Science and Computer Science

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

1. Classification Trees

Q1.1: At which frequencies does the electrical activity mostly occur?

The electrical activity, as represented by the amplitude of the spectrum, predominantly occurs at the lower frequencies has we can see in the graph. Specifically, we can observe significant activity in the range up to around 2 to 10 Hz. Each state shows a concentration of amplitude within specific frequency bands. For instance, the non-REM sleep (n) state appears to have activity at the very low-frequency range, while REM sleep (r) and awake (w) states show activity extending into slightly higher frequencies.

Q1.2: Can you easily distinguish between classes visually? What can you say about the inter- vs intra-class variability?

While it can be a bit difficult to visually differentiate between the states based on the EEG spectra, certain distinctions are noticeable:

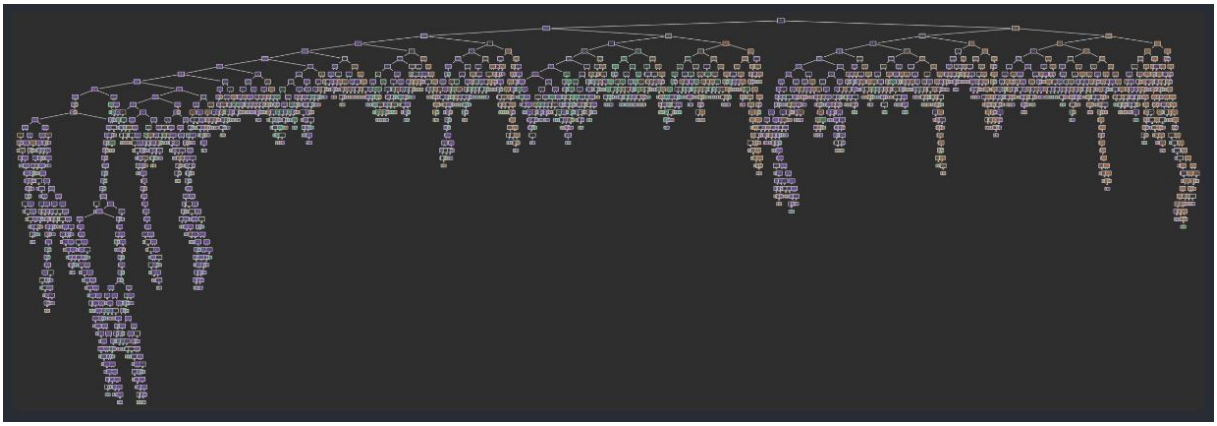
- **The non-REM sleep (n)** spectra exhibit a uniform pattern, with a pointed peak at lower frequencies, implying a low variability within this state (low intra-class variability). This consistency in the non-REM sleep state's spectrum could serve as a visual cue for identification.
- **The awake (w)** state also demonstrates a consistent spectral pattern, similar to the non-REM sleep, marked by low variability within this state (low intra-class variability). However, discerning the awake state from the non-REM sleep solely on visual inspection is complex (high inter-class variability), as both states lack significant peaks and maintain a lower amplitude throughout the spectrum.
- In contrast, **the REM sleep (r)** state shows a much higher degree of variation in its spectral profile, signifying greater variability among different instances of REM sleep (high intra-class variability). Visually, this state is more distinct compared to the other two, with notable and more prominent peaks in the spectrum (clear inter-class differentiation)

Q1.3: Describe both of these criteria. What does a gini impurity of 0 means? What does an entropy of 1 mean?

Has seen in the theory class:

- **Gini impurity:** It's a measure that expresses the likelihood of incorrect classification if you randomly pick an item and classify it according to the distribution of classes in the dataset. It ranges from 0 to 0.5 (where 0.5 denotes a 50/50 split in a binary classification), with 0 being the most desirable score. A Gini impurity of 0 means that the set is perfectly homogeneous, that is, all elements belong to the same class.
- **Entropy:** Almost the same concept has gini. It is a concept that measures the level of disorder or uncertainty in the data. It ranges from 0 to 1 in the case of binary classification (the range is 0 to log2(n) for a classification problem with n classes). An entropy of 0 means that there is no surprise in the data (perfect purity), while an entropy of 1 (in binary classification) indicates that the data is evenly split between the classes, representing the maximum level of disorder or uncertainty. An entropy of 1 would signify that the data within a node is perfectly split between the classes, meaning the information provided by the node is no better than a random guess.

Q1.4: This problem suffers from a common issue in machine learning. What is this problem called? What could be its causes? How can it be resolved?



The problem found in this system is the **overfitting**. Overfitting occurs when a model learns the training data too well, capturing noise and details that do not generalize to new, unseen data. Generally, the system arrives to a 100% of accuracy on the train set. Then in the test set the accuracy drops significantly.

Some causes of this problem:

- Training the model for too many iterations
- Having a model that is too complex relative to the simplicity of the data we have.
- Having a small data

Solving this problem:

- Collecting more data or augmenting the dataset artificially (data augmentation)
- Using techniques like cross-validation
- Implementing early stopping during training, so the model doesn't continue to learn the noise in the training data.

Q1.5: Use the visualization of this tree to show and explain:

What is a node? What is an edge? What is a leaf?

- **Node**: Is a point in the decision tree where the data is split. In this tree, each rectangle represents a node. Each node will test an attribute and split the data according to the outcome of the test.
- **Edge**: An edge is the line that connects one node to another.
- **Leaf:** A leaf is a terminal node that does not split any further. This is where a prediction is made.

What are the two additional hyperparameters doing? Do you think that both are necessary in this particular case (min_samples_leaf = 20, max_depth= 4)? Why?

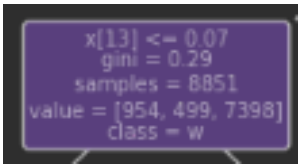
- **min_samples_leaf :** This parameter specifies the minimum number of samples required to be at a leaf node. For example, if a leaf results in a result lower than the number specified it will not create it. That's a good way to avoid overfitting.
- **max_depth :** This hyperparameter controls the maximum depth of the tree. The depth of a tree is the length of the longest path from the root to a leaf.

In our case, the parameters are controlling the complexity of the tree. The example we had in the Q1.4 doesn't have any limit that will avoid overfitting. So in our case the parameters will do it and maximize our chances of doing good predictions on a new input. Both can be necessary, but their necessity values would usually be determined through a process of experimentation and cross-validation to see how changes in these parameters affect model performance.

What does the color of each node represent?

The color in each node represents the majority class in that node after the split, with different colors corresponding to different classes. In this case, they play with the intensity of the color to say if there are more or less portions of the class.

Q1.6: Choose one of the nodes. Explain precisely the information given on each line of text in this node.



The first value is the condition to split the data. If x[13] is lower than 0.07. The testing value will go left, if not it goes right. The second line explains the gini that we had (Q1.3). The samples line equal to the total number of samples that were thrown at this node. The value represents the distribution of the classes within the samples at this node (class 0, 1 and 2). The class line indicates the majority class among this node.

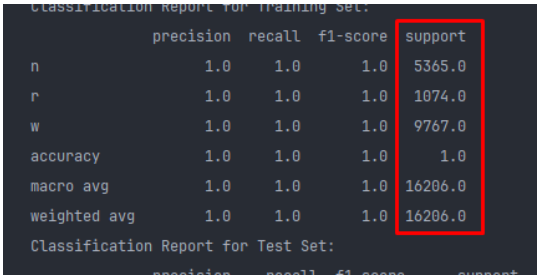
Q1.7: Does model 2 still have the same problem as model 1? Explain based on the classification reports and the confusion matrixes

In model 2, the accuracy on the training set is 0.813, and on the test set, it is 0.812, which are very close. The precision, recall, and f1-score for each class are also more consistent between the training and test sets compared to Model 1. This demonstrates that Model 2 does not suffer from the same overfitting problem as Model 1 since its performance generalizes better to unseen data. In the matrix, we can observe high values across the diagonal are pretty high but still not perfect.

Q1.8: One of the class seems more difficult to predict than others? Which one? Where could this difficulty come from in your opinion?

The class **'r'** has significantly lower precision and recall scores compared to the classes' 'n' and 'w' in both models. This is also reflected in the confusion matrices, where **'r'** has lower values on the diagonal compared to the other classes.

The main thing that comes to my mind is probably an "imbalanced Dataset" of this class compared to the other 2 classes. Has seen in the report given:


Classification Report for Training Set:

	precision	recall	f1-score	support
n	1.0	1.0	1.0	5365.0
r	1.0	1.0	1.0	1074.0
w	1.0	1.0	1.0	9767.0
accuracy	1.0	1.0	1.0	1.0
macro avg	1.0	1.0	1.0	16206.0
weighted avg	1.0	1.0	1.0	16206.0

Classification Report for Test Set:

Q1.9: What does this hyperparameter do? Explain giving examples from this dataset.

The **class_weight='balanced'** hyperparameter is used to address the issue of class imbalance in a dataset where the number of instances of one class significantly outnumbers the instances of another class. When set to 'balanced', this hyperparameter adjusts the weights of the classes inversely proportional to their frequencies in the input data. The weights are assigned to the classes such that smaller classes gain a higher weight and larger classes get a lower weight.

Using this parameter, the model gives more importance to correctly classifying instances of class 'r' (will probably multiply the number of values by the number it needs to equal the others), even though they are less frequent.

Q1.10: Compare results from model 2 and model 3. What are the pros and cons of each of them?

Pros Model 2 :

- Higher accuracy on both the training (0.813) and test (0.812) sets compared to Model 3.
- The f1-scores for classes 'n' and 'w' are higher in Model 2 than in Model 3, suggesting better overall performance for these classes.

Cons Model 2:

- The recall for class 'r' is significantly lower than in Model 3, demonstrating that Model 2 misses a larger number of actual 'r' instances. Like seen in the precedent exercices.

Pros Model 3:

- Significantly improved recall for class 'r' on both training and test sets

Cons Model 3:

- Lower accuracy overall on both training and test sets compared to Model 2

Summary:

Model 2 is generally more accurate and precise, but it struggles with recall for the minority class 'r'. Model 3, with balanced class weights, sacrifices overall accuracy and precision to improve the recall for the minority class 'r'.

2. Random Forest

Q2.1: For each of the hyperparameter: Is there a range of value giving particularly good results? Or particularly bad results?

- **max_depth**: The model's behavior for this hyperparameter seems to indicate a degradation/disparagement of performance (mean test score) as its value increases, but remains fairly homogeneous and consistent. One exception is a sample at the very beginning of the plot, which is more compact and less good than all the others.
- **n_estimators**: It starts off very poorly, then immediately stabilizes while remaining fairly dispersed. It is very stable throughout the plot.

- **max_features**: It all seems very stable and similar but with a slight improvement in the *worst* score. It is difficult to say if there are good or bad results for this hyperparameter.
- **min_samples_leaf**: This hyperparameter follows a real progression curve: as its value increases, the score becomes more precise and improves (while retaining a significant disparity).

Q2.2: These representations give valuable information about hyperparameters. It is nevertheless insufficient. What are/is the main problem(s) with those graphs in your opinion?

These graphs give great insight, but lack a little clarity. They don't allow us to define precisely whether the model performs better with a given hyperparameter and value. The overlap of data points doesn't allow us to understand our model's performance sufficiently.

Q2.3: What do the following plots represent?

These plots represent the combination of two different hyperparameters in the model. The lighter the color of the cell, the better the mean test score (=> good performance).

Q2.4: What do the white spots (=empty spots) in the heatmaps mean?

White spots in the heatmaps indicate combinations of hyperparameters for which there is no data available, either because:

- These combinations were not sampled during the hyperparameter tuning process.
- There was an error in the computation or the data collection process for these specific combinations.
- The combination was deliberately excluded, perhaps due to computational constraints or prior knowledge that certain ranges would not be effective.

Q2.5: How do those plots address the limitations of the previous visualizations?

These graphs make it much easier to observe patterns in the model, where it performs best (or not) thanks to the colors and the average score for each cell. It also makes it easier to deal with the problem of overlapping points, thanks to a unique representation for each value. The use of combinations between several hyperparameters also allows us to address other performance considerations for the model and understand how it can perform better.

Q2.6: What is grid search? Explain by giving real examples from this specific task.

Grid search is a hyperparameter optimization technique that methodically builds and evaluates a model for each combination of algorithm parameters specified in a grid. It is an exhaustive search because it goes through all possible combinations of the parameters.

- With our 4 hyperparameters max_depth with values [10, 20, 30], n_estimators [1 to 200], max_features [10, 12, 14, 16, 18] and min_samples_leaf with values [1, 2, 4], the grid search will evaluate the model with all combinations: (10, 1, 10, 1), (10, 1, 10, 2), (10, 1, 10, 4), (10, 1, 12, 1), (10, 1, 12, 2), etc. It can be very long.
- It will then use cross-validation to assess the performance of each model.
- The output would be the combination of parameters that gives the best performance according to a specified scoring metric (e.g., accuracy, balanced accuracy, etc.).

Q2.7: Use the plots above to narrow the range of hyperparameters you want to explore. Which values did you choose to test for each parameter? Justify your choices.

Based on the heat maps here is what we can say about the best choice for their values (yellowish areas):

- **max_depth**: the best scores are on their debut around the values 3-10. This can be explained by the fact that it can prevent overfitting when used as short values.
- **n_estimators**: There are no real ranges that stand out, except that we shouldn't test the value 1. However, we notice that the center-fine (100 to 200) is more sparsely populated than the rest. This could be a good research point (see course lecture).
- **max_features**: The same applies to max_features. The idea would therefore be to take mid-range values (12 to 20) to enable our model to offer a large number of functionalities without sacrificing the risk of overfitting and the diversity of decision trees.
- **min_samples_leaf**: Staying around the 20 samples limit is coherent and can prevent too much overfitting or even increase model stability.

Note: To save computation time, the plan is not to test all range values one by one, but to use steps such as 25 by 25 for the n_estimator or by 2 for features (non-exhaustive).

Q2.8: Which value did you choose for each hyperparameter? Why?

We chose the best parameters according to the GridSearch algorithm (thanks to *grid_search.best_params_*).

- **max_depth (7)**: it is important to not go too deep in the tree. Number 7 is a good way to capture important patterns in the data without overfitting.
- **n_estimators (175)**: it is a relatively high number of trees, which usually improves model performance by reducing variance. It can suggest that it provides a good balance between accuracy and computational efficiency.
- **max_features (16)**: it is a good compromise to find a balance in the model's ability to consider enough features at each split without overfitting.
- **min_samples_leaf (18)**: it aims to generalize over precision in individual leaves. This can help the model to be more robust and less sensitive to noise in the training data.

Q2.9: The test set should be used only at this stage, and it is theoretically important not to change the hyperparameters based on the performance on the test set. Why?

The test set is used only at the final stage of model evaluation to assess how well the model generalizes to unseen data. It is theoretically important not to change hyperparameters based on the test set performance because it can avoid overfitting easier, preserve the test set Integrity by keeping the data unseen to the model and therefore generate an objective evaluation of the model and data.

Q2.10: Comment on your results. -> How well does the model generalize on unseen data? Is a random forest better than a single classification tree in this case? What is the main challenge of this dataset? …

The Random Forest model has an accuracy of approximately 86.97% on the test set, which is slightly lower than its accuracy on the training set (about 88.88%). This indicates that the model generalizes well, with only a slight drop in performance on unseen data. It has about 10% more accuracy than the single classification algorithm (75 and 74%) which shows that

the random forest has better performance for its classes. It is the same with every other metric. The random forest seems to outperform the single classification algorithm.

The main challenge lies in the generalization of the 'r' class, which although better optimized for unseen data for random forest compared to single classification is still too sloppy. It shows that additional feature engineering, class rebalancing, or alternative modeling approaches might be needed to improve classification for this particular class.

Q2.11: How is this importance calculate?

The importance of each feature in the context of a Random Forest classifier is calculated using Mean Decrease in Impurity (MDI), also known as Gini importance. It measures how much each feature contributes to the homogeneity of the nodes and leaves in the trees of the forest. Here's how it's generally calculated:

- During the training of a Random Forest, each tree is constructed by recursively splitting nodes starting from the root. At each split, the feature that results in the largest decrease in impurity is selected.
- Impurity is measured by the Gini impurity or entropy for classification tasks. A decrease in impurity after a split means that the node's child nodes are more homogeneous in terms of the target variable than the node itself.
- The importance of a feature is then the sum of the impurity decreases over all trees in the forest, normalized by the total number of trees.
- This results in a score that represents the importance of each feature in the prediction process.

Q2.12: What can you conclude from this graph?

A few features stand out with significantly higher importance scores than the others, it seems that there is a sharp split between the first features (0 to 18) and the others (19 to 100). They certainly are the key drivers in the classification process. We can suggest that the dataset may contain a lot of features that don't contribute much to the overall predictive power of the model.

The error bars, which represent the Standard Error of the Mean (SEM), are relatively small for the top features, indicating that their importance scores are consistent across the different trees in the forest.

The SEM error bars provide an estimate of the uncertainty of the feature importance scores. Smaller error bars indicate more confidence in the importance score's stability across the trees in the forest. The features with larger error bars might have more variable importance across different trees, indicating less stability in their contribution to the model's predictions.

Conclusion:

There is a clear concentration of importance in a relatively small subset of features. The first few features have significantly higher importance scores compared to the rest, suggesting they have a stronger impact on the model’s decision-making process.

In the other hand, many features have a low importance score, which trails off into what appears to be a long tail. This implies that after a certain point, additional features contribute less and less to the model's ability to distinguish between classes.

3. Gradient Boosting for classification

Q3.1: Two additional hyperparameters were added compared to the RandomForestClassifier. What are these hyperparameters, and what roles do they play?

As we can see, in the code, there are two additional parameters. We are referring here to the parameters “learning_rate” and “subsample”. Here is an explanation of each:

- **learning_rate**[1] : This parameter scales the contribution of each tree to the final model. It essentially controls the step size at each iteration while moving toward a minimum of a loss function. Has seen until now in all the PW.
- **subsample**[2] : This parameter specifies the fraction of samples to be used for fitting the trees. If it's less than 1.0, then each new tree in the ensemble is trained on a random subset of the data, which introduces more randomness into the model and can help prevent overfitting.

Q3.2: Comment the results. Compare these results with the ones obtained with the RandomForestClassifier. Compare more specifically the precision, the recall and the f1-score of the 'r' class obtained with GradientBoostingClassifier and RandomForestClassifier. What are your conclusions?

Here is an overview of the results we had in the notebook :

- Overall Accuracy :
  - RF: Training: 88.88%, Testing: 86.96%
  - **GB: Training: 95.57%, Testing: 91.24%**

Let’s now dive into the precison of the “r” class:

- Precision (Class 'r'):
  - RF: Training: 48.59%, Testing: 39.75%
  - **GB: Training: 95.47%, Testing: 75.17%**
- Recall (Class 'r'):
  - **RF: Training: 81.93%, Testing: 59.41%**
  - GB: Training: 68.62%, Testing: 41.33%
- F1-Score (Class 'r'):
  - RF: Training: 61.00%, Testing: 47.63%
  - **GB: Training: 79.85%, Testing: 53.33%**

Conclusion :

Gradient Boosting emerges as a robust model, demonstrating superior overall performance characterized by higher accuracy and more balanced metrics across different classes. This superiority is especially pronounced in its precision in predicting instances of class 'r'. Precision, in this context, refers to the model's ability to correctly identify 'r' instances out of all instances it labeled as 'r'.

However, this high precision comes with a trade-off in recall, a measure of the model's ability to identify all actual instances of class 'r'. Here, Random Forest takes the lead, showcasing a higher recall for class 'r' compared to Gradient Boosting. This means that while Random

---

[1] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

Forest may incorrectly label more non-'r' instances as 'r' (lower precision), it is less likely to miss the 'r' instances (higher recall).

Choosing between these two models thus becomes a question of prioritizing precision over recall or vice versa, depending on the specific requirements and consequences associated with the predictive task.