Classifying Epileptic Seizures with Machine Learning Algorithms

FYS-STK4155

December 9, 2019

Teah Kaasa McLean
Marius Helvig Havgar
Åsmund Danielsen Kvitvang

Abstract

Extreme gradient boosting (XG) is a new and highly competitive algorithm for classification. In this paper, we compare XG to two other classification models, traditional random forests (RF) and K-nearest neighbors (KNN), to classify epileptic seizures. They all show very promising results, with accuracy scores in the range $(0.9365, 0.9772)$, but we find no clear indication that XG outperforms RF or KNN. The best sensitivity, 0.9930, was obtained by KNN, while RF recorded the highest specificity of 0.9745.

Contents

1 Introduction

It has become more and more popular to apply machine learning to healthcare and medical research. Machine learning has the wonderful ability to process datasets that are way too large for human capacity and to recognize patterns that are not obvious. For example, machine learning can be used to identify cancerous tumours in breast tissue or diagnose epilepsy.

In this paper we will apply three machine learning methods to a medical dataset. The dataset consists of electroencephalographic (EEG) signals, which record electrical activity of the brain and are most often used to diagnose epilepsy. We compare the performance of three popular machine learning models, and the goal is to determine whether they are suitable for diagnosing epilepsy.

In [Sub07] the author applies a neural network model and a mixture of experts-model with discrete wavelet transformation for feature extraction to classify epilepsy. They found that both approaches were suitable for this task.

The methods we will look at are K-nearest neighbors (KNN) and two ensemble methods, random forests (RF) and XGBoost (XG). RF and XG both have underlying tree structures, and are thus of interest to compare. Though RF and XG often give very accurate results, they can be computationally expensive and difficult to interpret, especially XG, while K-nearest neighbors is a highly intuitive method. We will therefore look into whether the gained performance justifies the complicated model.

We will start by introducing the methods and algorithms and give a brief description of the implementation. Further, we will look at the accuracy and ROC-AUC scores and the ROC curve plots. Finally, we will discuss the differences between the methods and compare them to determine their suitability.

2 Theory

In this section we will introduce the algorithms and methods used for classification in this paper. For a discussion on classification, see [MHK19a].

2.1 Metrics

The following section will describe the metrics we will use in this paper to determine the performance quality of the models.

2.1.1 Accuracy Score

The accuracy score is a metric for measuring the performance of a classifier. The accuracy is the number of correctly predicted targets t_i divided by the total number of targets,

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n},$$

where I is the indicator function, returning 1 if $t_i = y_i$ and 0 otherwise. A perfect classifier will have an accuracy score of 1.

2.1.2 ROC-AUC Score

The ROC-AUC score is a measurement of performance for classification as a function of thresholds. The score represents how capable the model is of distinguishing between classes.

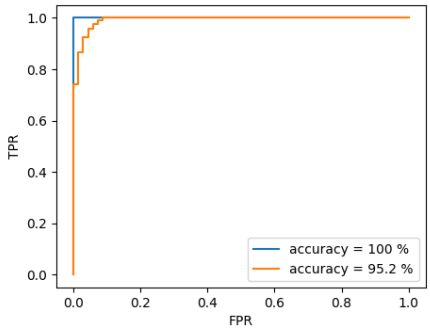


Figure 1: Figure showing an ideal ROC-AUC curve versus the ROC-AUC curve of a classifier with accuracy score of 95.2 %.

The receiver operating characteristic (ROC) curve is a probability curve that plots the true positive rate (TPR) against the false positive rate (FPR) for each decision threshold, as seen in fig. 1. TPR or *sensitivity* is defined as

$$\frac{\text{TP}}{\text{TP} + \text{FN}},$$

where TP is true positives and FN is false negatives. In other words, TPR is the ratio of correctly classified positives. A perfect sensitivity has the value of 1. *Specificity* is the ratio of correctly classified negatives,

$$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}},$$

where TN is true negatives and FP is false positives. A perfect specificity will have a value of 1. The FPR is then $1 -$ specificity,

$$\frac{\mathrm{FP}}{\mathrm{TN}+\mathrm{FP}}.$$

The AUC is the area under the ROC curve. The closer the AUC is to 1, the better the classification is. If the AUC is 0, the model classifies all negative instances as positive and all positive instances as negative, which is the exact opposite of what we want. The worst case scenario is if the model is not able to distinguish the classes at all, then the AUC score is 0.5 [Mar09, pp. 24-25].

2.2 K-Nearest Neighbors

K-nearest neighbors is a classification algorithm that uses the class of the nearest neighbors to predict the class of a data point [HTF09, pp. 16-17]. If $\|\cdot\|$ is the norm on the feature space, the algorithm considers for a fixed $K \in \mathbb{N}$ the class of the K neighbors which are closest in norm, and infers a class e.g. by majority voting. In our case, this involves classifying whether or not a person has an epileptic seizure.

Formally, let $\boldsymbol{x}$ denote the datapoint we wish to classify, and let $\{\boldsymbol{x_i}, y_i\}_{i=1}^N$ be a set of datapoints $\boldsymbol{x_i}$ and their associated classes y_i. Order $\{\boldsymbol{x_i}, y_i\}_{i=1}^N$ such that

$$\|\boldsymbol{x} - \boldsymbol{x_1}\| \leq \|\boldsymbol{x} - \boldsymbol{x_2}\| \leq \cdots \leq \|\boldsymbol{x} - \boldsymbol{x_N}\|.$$

Choose $y_1, y_2, \ldots, y_K$ associated with $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_K}$, respectively. Then, the predicted class of $\boldsymbol{x}$ is

$$\hat{y} = \frac{1}{K}\sum_{k=1}^{K} y_k,$$

rounded by some threshold. KNN might also be implemented with a scaling rule. This is done by associating to each $\boldsymbol{x_i}$ a weight λ_i proportional to $\frac{1}{\|\boldsymbol{x}-\boldsymbol{x_i}\|}$, such that $\sum_{i=1}^{k} \lambda_i = 1$. Then,

$$\hat{y} = \frac{1}{K}\sum_{k=1}^{K} y_k \lambda_k,$$

rounded by some threshold [HTF09, pp. 479-487]. We will refer to the variant where the neighbors are weighted by their distances as *distance weighting*, and the variant where each neighbor is weighted equally as *uniform weighting*.

2.3 Decision Trees

Before we can dive into the theory of random forests, we must first have a basic understanding of their building blocks - decision trees. The following theory is based on the lecture notes on decision trees by M. Hjorth-Jensen [Hjo19].

Decision trees are, as the name implies, tree structures that make a decision based on some training data. They can be used for both regression and classification problems, but we will focus on classification. The trees have a *root node* that splits into *branches*, and each branch could potentially split several times and will end in a *leaf node*.

The main idea of a decision tree is to find the predictors that contain the most information regarding the target value. These predictors will be used to form the splitting nodes. Growing a decision tree is done by performing recursive binary splitting, which is a greedy algorithm. In recursive binary splitting, all the features are considered and different split points are tried and tested using a cost function. The split with the best cost is chosen. Splitting is performed recursively on the resulting subtrees.

There are two common recursive binary splitting algorithms, CART for both regression and classification cases, and ID3 for classification. The CART algorithm is used by scikit-learn [Ped+11] and will be used in this paper. A brief description of the CART algorithm can be found in algorithm 1. For a description of the ID3 algorithm, see [Hjo19].

As mentioned, a cost function is used to determine which feature to choose for the splitting. A common choice for the cost function is the *Gini index*,

$$G = \sum_{k=1}^{K} p_k(1 - p_k),$$

where K is the number of observations and p_k is the proportion of observations of class k in a particular group. The Gini index gives an idea of how good a split is by measuring how mixed the classes are in the two groups created by the split. A perfect separation of the classes would yield a Gini score of 0, whereas the worst case separation would yield a score of 0.5.

Another choice for the cost function is the *entropy* or *information gain*,

$$E = \sum_{k=1}^{K} -p_k \, log(p_k).$$

The entropy of a feature tells us how much extra information we would get from knowing the value of that feature. If the entropy is 0, we gain no information from that feature. If the feature separates examples perfectly between positives and negatives, then the entropy is maximized. The feature that yields the highest entropy is chosen to create a split [Mar09]. Typically, the CART algorithm uses the Gini index as the cost function and the ID3 algorithm uses the entropy.

A big concern in machine learning is whether our models will overfit the data. Decision trees are very data sensitive and can become very large, and can therefore easily overfit. In order to try to prevent this, we need to install a stopping criteria for the growth of the tree. An example would be to set a minimum number of observation inputs in a leaf. Another example is to set a maximum depth, i.e. a maximum length of the longest path from the root to a leaf [Hjo19].

Algorithm 1 CART

Let K be the number of features.
for feature $k = 1, \ldots, K$ **do**
Compute the cost function for the pair (k, t_k) where t_k is a threshold

$$C(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right},$$

where $m_{left/right}$ is the number of instances in the corresponding subsets and $G_{left/right}$ is the Gini score of the corresponding subsets.

Choose the pair (k, t_k) with the lowest cost and split the tree accordingly. Repeat process recursively for the resulting subtrees until stopping criteria is reached.

2.4 Random Forests

The theory in this section is based on the lectures notes on decision trees and random forests by M. Hjorth-Jensen [Hjo19].

The random forest algorithm is an ensemble machine learning algorithm that can be used for either regression or classification. The focus in this paper will be on random forest classifiers.

A random forest is a collection of individual decision trees that operate as a committee or an *ensemble*. Each individual tree in the random forest predicts a class, and the class with the most votes becomes the prediction for the forest. For example in a binary classification case, if a random forest contains ten trees, and seven trees predicted class 1 and three predicted class 0, then the forest would predict class 1.

Having several relatively uncorrelated trees operating as a committee will outperform any of the individual trees, because this will handle overfitting and reduce variance. The trees will "protect each other" from their individual errors. In other words, if there are some trees that predict the wrong class, then there will also be many trees that predict the right class. In general, increasing the number of trees will increase the chance of predicting the right class.

The key to the advantages mentioned is having trees with low correlation. The RF algorithm ensures that the individual trees are not too correlated by implementing *boostrap aggregation* or *bagging*. Decision trees are very sensitive to the data they are trained on. Thus, randomizing the training data will decrease the correlation between the trees. Bagging randomizes the data set a tree in the forest is trained on, by drawing a bootstrap sample with replacement from the original dataset.

Because each tree has a random set of datapoints, it may not have access to all the original features and must pick from a random subset of the features. As seen in section 2.3, single decision trees consider all possible features and pick the one that yields the most separation between the observations. If the trees in a forest have access to different features, the features they choose as their root node will differ. Thus, this feature randomness allows for even less correlation between the trees.

The RF algorithm can be summarized as in algorithm 2 [Hjo19].

Algorithm 2 Random Forests

Let K be the number of trees and p be the number of predictors.
for $k = 1, \ldots, K$ **do**
Draw a bootstrap sample from the training data.
Grow a random forest tree T_k based on the bootstrapped data by repeating the following:
while depth of tree is less than max depth **do**
Select $m \leq p$ predictors at random.
Pick the best split point among the m features using the CART or ID3 algorithm.
Split the node into daughter nodes.
Output the ensemble of trees $\{T_k\}_1^K$ and make a prediction.

2.5 XGBoost

XGboost stands for *extreme gradient boosting* and is a gradient boosted tree model proposed in 2016. It has since won several machine learning competitions, and has proven to be a highly successful model both in terms of performance and scalability. We refer to [CG16] for the full details on the implementation and particularities of XG, but we proceed to give a brief outline of gradient boosted tree models.

2.5.1 Gradient Boosted Trees

The focus of discussion of gradient boosted trees will be classification models, but only minor altercations are needed to extend the results to the problem of regression. Let $X \subseteq \mathbb{R}^n$ be the space of predictors and $Y \subseteq \mathbb{N}$ the space of associated classes. Let $\boldsymbol{x} \in X$ be a vector of predictors, and $y \in Y$ be the class associated with $\boldsymbol{x}$. Let $\mathcal{L}(y, \hat{y})$ be a convex and differentiable function that measures the loss between the true class y and the predicted class $\hat{y}$ associated with $\boldsymbol{x}$. The idea of gradient boosting is simple. We begin with a weak tree classifier. Then, over a number of iterations we calculate the negative gradient of the loss-function for each $\boldsymbol{x_i}, i = 1, \ldots, N$, denoted r_i with respect to the prediction y_i. Then, for each iteration, fit a weak learner to the set $\{\boldsymbol{x_i}, r_i\}_{i=1}^N$. We then continue additively, updating the model by adding the new weak learner to the previous model for each iteration. This can be expressed in pseudocode as follows.

Algorithm 3 Gradient boosted trees

Let M be the number of iterations, N the size of the dataset.
Initialize $F_0 \leftarrow \arg\min_{\gamma} \sum_{i=1}^{N} \mathcal{L}(y, \gamma)$.

for $m = 1, \ldots, M$ **do**
Calculate *pseudo-redsiduals* $r_{m,i} = -\frac{\partial \mathcal{L}(\boldsymbol{x_i}, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}$
Form the dataset $\{x_i, r_{m,i}\}$ and fit a simple tree h_m to $\{x_i, r_{m,i}\}$
Calculate $\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} \mathcal{L}(x_i, F_{m-1}(x_i) + \gamma h_m(x_i))$
Update the model: $F_m \leftarrow F_{m-1} + \gamma_m h_m$
Let finally $F \leftarrow F_M$

In particular, the loss function to be minimized in XG has the form

$$\mathcal{L}(y_i, F(x_i)) = \sum_{i=1}^{N} l(x_i, F(x_i)) + \sum_{k=1}^{K} \Omega(f_k),$$

where l is a convex, differentiable function. Furthermore, F consists of K simple trees f_k, where T is the number of leaves and $\boldsymbol{w}$ is a vector of weights in f_k. Then

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \|w\|^2, \quad \gamma, \lambda \in \mathbb{R}^+$$

It is easy to see that Ω penalizes model complexity, and hence we can think of it as a regularization term.

3 Method

The software associated with this paper can be found at [MHK19b].

Dataset

The original dataset [RG+01] is a very commonly used dataset featuring epileptic seizure detection. It was composed by recording electroencephalographic (EEG) signals of 500 individuals for 23.6 seconds. Each individual has 4097 data points.

The dataset from the UCI Machine Learning Repository [WF17] is a preprocessed and restructured version of the epileptic seizure dataset. The 4097 data points per individual has been shuffled and divided into 23 sections,

each corresponding to a time span of 1 second. Each section contains 178 data points. These 178 data points serve as the explanatory variables or the predictors. Thus we have $23 \cdot 500 = 11500$ pieces of information, and each piece of information contains 178 data points, resulting in a 11500×178 design matrix.

The response variable can fall into five classes. Class 1 represents the subjects that had an epileptic seizure. The remaining classes did not have an epileptic seizure, and the classes differ in how the brain signal recordings were administered. For example, the subjects in class 4 had their eyes open during the recording, whereas the subjects in class 5 had their eyes closed. In this paper we are interested in classifying whether an epileptic seizure occured or not, and thus use binary classification on class 1 against the rest.

Of the 11500 observations, 2300 represent epileptic seizural activity, meaning that 20% of the dataset belongs to class 1 and 80% belongs to class 0. Thus the dataset is biased in favor of non-seizural activity.

The data will be scaled using scikit-learn's `StandardScaler`, which transforms the features to standard normal distribution by subtracting the mean and scaling to unit variance. This is done by first splitting the data into two sets by using scikit-learn's `train_test_split`, then fitting the scaler to the training set, and using it to transform both sets. This prevents information from the training set to influence the test set, also known as data leakage.

3.1 Model Validation and Parameter Tuning

As mentioned, we will use scikit-learn's `train_test_split` (TTS) with 75% of the dataset allocated to training. We set aside a validation set, the remaining 25% after TTS.

To tune the hyper parameters for KNN and RF, we have used scikit-learn's method `GridSearchCV`. `GridSearchCV` takes an array of possible values for each hyper parameter to be tuned for the model, and exhaustively estimates the accuracy of the model for each combination of hyper parameters. The set of hyper parameters associated with the best performance can then be identified. We point out that the validation set is never seen by `GridSearchCV`. This reduces the risk of overfitting.

Once the set of hyper parameters for the model is selected, we estimate the model accuracy and ROC-AUC score N times by splitting and scaling for each iteration. We use these estimates to create a 95% confidence interval for the scores.

3.2 K-Nearest Neighbors

There are three parameters to be tuned in KNN, the number of neighbors, weighting and the choice of norm. The number of neighbors corresponds to the parameter `n\_neighbors`. The choice of weighting, which is either uniform or distance weighting, has parameter name `scaling` scikit-learn's KNN classifyer. The parameter name for the choice of norm is `p`, and a choice of p corresponds to the norm $\|\cdot\|_p$, i.e. the p-norm, where $p \in \mathbb{N}^+$. We will use `GridSearchCV` to tune these, and the performance will be measured by accuracy score. The set of parameters passed to `GridSearchCV` will be: n_neighbors $= \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, scaling $= \{\text{distance}, \text{uniform}\}$, $p = \{1, 2, 3\}$.

3.3 Random Forest

The analysis for random forests was performed using functionalities from the scikit-learn library.

The `RandomForestClassifier` has 19 parameters that can be tuned, but to avoid overly complicated models we choose three parameters to tune, namely number of estimators (or trees), max depth and criterion. Max depth decides the maximum length of a path from the root to a leaf of the trees. The criterion parameter is which function to use to measure the quality of a split.

Due to computational costs, a select number of values will be chosen for testing. Looking at forests with few shallow trees, we find the results to be unimpressive. Therefore we choose to look at forests with 80, 90, 100 and 110 trees, as random forests are generally more robust with more trees. For max depth, we will test 15 and 20. The Gini index and entropy are the only supported criteria for scikit-learn's random forest classifier, and thus we will look at both of these for testing.

3.4 XGBoost

For tuning of parameters in the `XGBClassifier`, one could use `GridSearchCV`. As mentioned, this function does an exhaustive search over the specified parameter values for a model. But the classifier from XG easily overfits, and in this manner one must be careful to enjoy a pretty result too much. Therefore it might be clever to manually loop through these parameters and stop the running when the change in accuracy is below a tolerance of choice. The choice of parameters will be based on plotting the accuracy

and ROC-AUC score as a function of number of trees. After fine tuning the number of trees and maximum depth, we will also tune the learning rate.

4 Results

Parameter Tuning

The chosen hyper parameters are listed in table 1. As mentioned, the parameters for KNN and RF were found by `GridSearchCV`, based on the accuracy score. Whenever several models resulted in indistinguishable scores, we opted for the simplest model.

To choose suitable parameters for XG we have done as described in section 3.4. The parameters we have tuned are `n_estimators` and `max_depth`, which are the number of trees in the forest and the maximum depth of each tree. As written in table 1 the respective choices were 100 and 5.

Figure 2 shows the difference in the accuracy score by change of color and from this we might observe that there is little action going on after number of trees reaches 100 or the maximum depth reaches 5. This observation may also be confirmed by looking at fig. 3a, where the slope of the tangent decreases rapidly when the number of trees reaches 100. In fig. 3b it seems that a good choice of learning rate would be somewhere between 0.05 and 0.1 and we have therefore opted for 0.08.

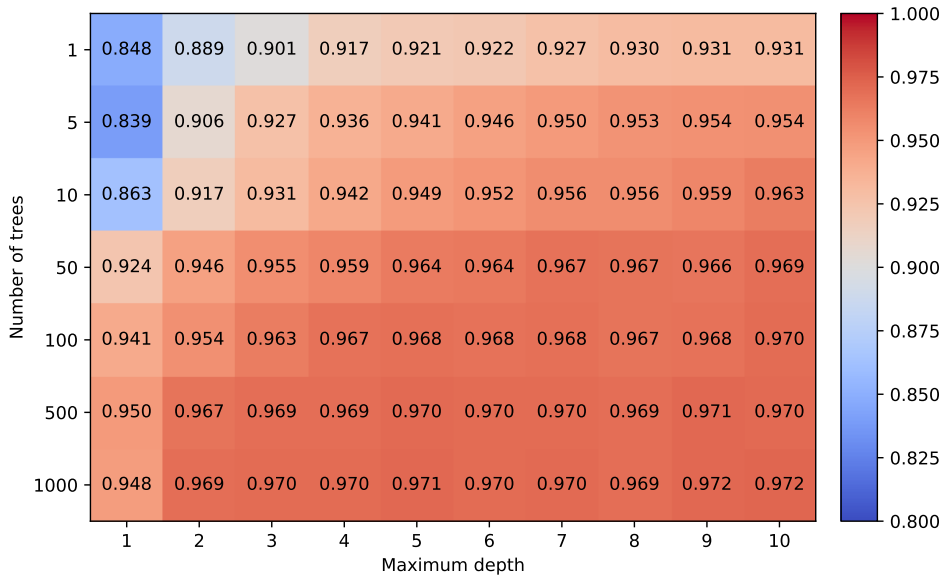
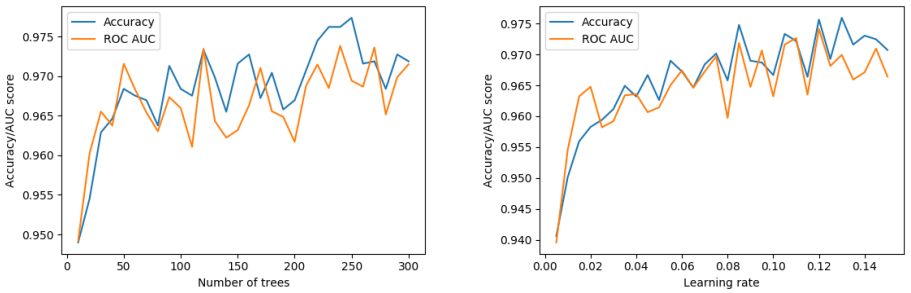


Figure 2: Heatmap showing accuracy score of XGBoost's classifier over different numbers of trees and different numbers of maximum depths. All other parameters of the classifier are set to default.



(a) Fine tuning of the number of trees. **(b)** Fine tuning of the learning rate.

Figure 3: Fine tuning of the number of trees (estimators) and learning rate in the XG classifier.

	RF	XG	KNN
No. estimators	80	100	-
Max depth	20	5	-
Criterion	ent	-	-
No. Neighbors	-	-	2
Scaling	-	-	dist
Norm	-	-	L_2

Table 1: The set of hyper parameters chosen for each of the models. Ent is the entropy criterion. Dist is the inverse distance scaling.

Accuracy and ROC-AUC scores

Using the models explained in table 1, we get the accuracy and ROC-AUC scores listed in table 2. The results are reported as 95% confidence intervals. The confidence intervals are relatively narrow for all the models, with KNN being the widest, suggesting that the scores for KNN are the least certain. With a confidence level of 95%, RF achieved an accuracy score of 0.9710 ± 0.0062 and ROC-AUC score of 0.9673 ± 0.0084. XG achieved accuracy and ROC-AUC scores of 0.9683 ± 0.0015 and 0.9681 ± 0.0055, respectively. KNN achieved slightly lower values across all metrics, with an accuracy of 0.9467 ± 0.0102 and ROC-AUC score of 0.9659 ± 0.0102. However, KNN has shown to classify significantly fewer false positives than the other methods, see table 3.

	Accuracy	ROC-AUC
KNN	0.9467 ± 0.0102	0.9659 ± 0.0102
RF	0.9710 ± 0.0062	0.9673 ± 0.0084
XG	0.9683 ± 0.0015	0.9681 ± 0.0055

Table 2: 95% confidence intervals for the accuracy and ROC-AUC scores for the chosen KNN, RF and XG models.

Figure 4 shows the ROC curve for our KNN, RF and XG models. We see that the RF and XG curves are nearly identical, to the point where basing the choice of the best model on the ROC curves alone would not makes sense. For KNN, the ROC curve is only defined pointwise. This is due to the fact that there is only a finite number of thresholds and the prediction probabilities are clustered.

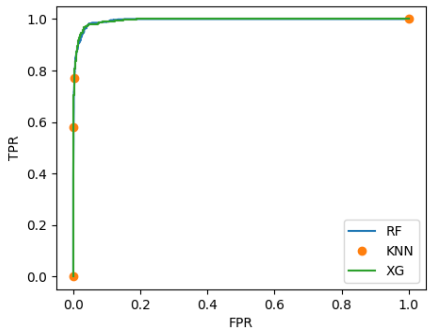


Figure 4: ROC curves for the chosen KNN, RF and XG models on test set which is 25% of the original data.

Confusion Matrices

Confusion matrices for the three models can be found in table 3, table 4 and table 5. It is worth noting that from a medical perspective, the consequence of falsely predicting a negative might be worse than falsely predicting a positive. Letting the decision threshold for predicting classes vary might remedy this.

Table 3 shows that the KNN model with 2 neighbors using the L_2 norm and distance scaling classifies negatives nearly perfectly, much better than RF and XG. However, KNN has the highest number of false positives.

Based on the confusion matrices, we have calculated the sensitivity and the specificity for each of the models and compiled them in table 6. We see that KNN has the highest sensitivity, whereas RF has the highest specificity.

		Predicted	
		Seizure	Nonseizure
Actual	Seizure	428	3
	Nonseizure	129	2315

Table 3: Confusion matrix for KNN with 2 neighbors, using the L_2 norm and weighting by inverse distance. The test set was 25% of the original data and contained 557 seizural observations and 2318 nonseizural observations.

		Predicted	
		Seizure	Nonseizure
Actual	Seizure	497	24
	Nonseizure	60	2294

Table 4: Confusion matrix for RF with 80 trees of max depth 20, using the entropy. The test set was 25% of the original data and contained 557 seizural observations and 2318 nonseizural observations.

		Predicted	
		Seizure	Nonseizure
Actual	Seizure	486	21
	Nonseizure	71	2297

Table 5: Confusion matrix for XG with 100 trees of max depth 5. The test set was 25% of the original data and contained 557 seizural observations and 2318 nonseizural observations.

	Sensitivity	Specificity
KNN	0.9930	0.9472
RF	0.9539	0.9745
XG	0.9586	0.9700

Table 6: The sensitivity and specifity for our chosen KNN, RF and XG models.

5 Discussion

In [Can19], the authors compared XG to other machine learning techniques on a large sample of UCI datasets. They found that with default parameters, XG does not perform better than other machine learning models. They raise the question of whether some of XG's success can be attributed to parameter tuning, since this is easily done within the XGboost framework, in contrast to for example RF techniques. It is therefore important to be aware of the risk of overfitting, and when feasible, limit the amount of parameter tuning when comparing the results to the results of other models. Random forests are much more able to handle overfitting, due to the randomness achieved from bagging the decision trees.

From our results, there is no convincing reason to prefer XG to RF on this dataset, as the models perform practically identically. Continuing the parameter tuning in XG could probably increase accuracy scores further, but this would likely increase the risk of overfitting. The ROC-AUC scores for the two models indicate that both are able to separate the classes well.

As mentioned, KNN had slightly lower accuracy and ROC-AUC scores than XG and RF. The confidence intervals were also wider, meaning that there is a higher uncertainty. However, though it has a higher number of misclassifications than the two other models, it has significantly fewer false negatives. Therefore it might be a safer model for medical prediction. It also has the benefit of being highly explainable and less prone to overfit.

6 Conclusion

In this paper, we have applied the machine learning classifiers K-nearest neighbors, random forest and extreme gradient boosting to the binary classification of EEG data. We took an empirical approach in parameter tuning, and compared the resulting models. We found that the models achieved accuracy scores in the range of $(0.9365, 0.9772)$. The best sensitivity, 0.9930, was obtained by KNN. RF achieved the best specificity, 0.9745. These results suggest that all the models may all be feasible choices for supporting physicians in diagnosing epilepsy.

In this paper we have taken a simple approach to the classification problem. Even though the results have been satisfactory, they could possibly be improved upon by applying methods of feature selection or more specialized data preprocessing, such as wavelet transformation.

In addition, in safety critical applications like medical prediction, it is important that the models are explainable, i.e. that the predictions can be explained and understood. It is also preferable that the model expresses a level of confidence in the prediction. Other machine learning techniques might be more suited to accomplish these goals, for example probabilistic neural networks, as suggested in [Seb19].

References

[Can19] G. M.-M. Candice Bentéjac Anna Csörgő. "A Comparative Analysis of XGBoost". In: 2019. URL: https://arxiv.org/abs/1911.01914.

[CG16] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 785–794. URL: http://doi.acm.org/10.1145/2939672.2939785.

[Hjo19] M. Hjorth-Jensen. *Data Analysis and Machine Learning: From Decision Trees to Forests and all that*. Nov. 2019. URL: https://compphysics.github.io/MachineLearning/doc/pub/DecisionTrees/html/DecisionTrees.html.

[HTF09] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*. 2nd ed. Springer, 2009.

[Mar09] S. Marsland. *Machine Learning - An Algorithmic Perspective*. 2009.

[MHK19a] T. K. McLean, M. H. Havgar, and Å. D. Kvitvang. *Classification and Regression - From Regression to Neural Networks*. University in Oslo, Nov. 2019. URL: https://github.com/aasmunkv/PROJECT2_FYS-STK4155.

[MHK19b] T. K. McLean, M. H. Havgar, and Å. D. Kvitvang. *Classification Epileptic Seizures with Machine Learning Algorithms*. University in Oslo, Dec. 2019. URL: https://github.com/aasmunkv/PROJECT3_FYS-STK41555.

[Ped+11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[RG+01] A. R.G. et al. *Epileptic Seizure Recognition*. Orginal source of dataset. 2001. URL: http://epileptologie-bonn.de/cms/front_content.php?idcat=193&lang=3&changelang=3.

[Seb19] Y. G. Sebastian Farquhar Michael Osborne. "Radial Bayesian Neural Networks: Beyond Discrete Support In Large-Scale Bayesian Deep Learning". In: 2019. URL: https://arxiv.org/abs/1907.00865.

[Sub07] A. Subasi. "EEG signal classification using wavelet feature extractionand a mixture of expert model". In: Department of Electrical and Electronics Engineering, Kahramanmaras Sutcu Imam University, Avsar Yerleskesi, 46050-9 Kahramanmarasß, Turkey, 2007. URL: https://doi.org/10.1016/j.eswa.2006.02.005.

[WF17] Q. Wy and E. Fokoue. *Epileptic Seizure Recognition (UCI)*. Pre-processed and reshaped version of a very commonly used dataset featuring epileptic seizure detection. May 2017. URL: https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition.