BILKENT UNIVERSITY

Department of Electrical and Electronics Engineering

Bilkent 06800 Ankara Turkey

EEE 485/585 - Statistical Learning and Data Analytics

Interim Project Report

Classification of Epileptic Seizure From Electroencephalogram (EEG) Signals Based on Machine Learning Approaches

Ecem Şimşek – Emirhan Koç

21802384 - 22101553

Section: 1

Group: 7

Course Instructor: Süleyman Serdar Kozat

20.11.2022

TABLE OF CONTENTS

1. Introduction and Problem Description

Epilepsy can be defined as a well-known and prevalent chronic neurological disorder that can affect male and female individuals of any race, ethnicity, or age [1]. According to the number of epilepsy patients around the world, it was observed that it mostly affects infants and the elderly. Epilepsy is characterized by the occurrence of epileptic seizures in which the activities in different parts of the brain become abnormal and these abnormalities arise from sudden and extreme discharge of brain nerve cells [2]. These abnormalities can cause the patients to display unusual behavior, and/or experience unusual sensations and loss of consciousness [1]. Hence, it is important to detect seizures in a correct and timely manner in order to control the progression of the disorder, prevent the possibility of self-harm of the patients and enhance their quality of life. As a result, various medical monitoring and detection devices are used in clinical examinations to detect epileptic seizures and these tools include electroencephalogram (EEG), magnetoencephalography (MEG), and functional magnetic resonance imaging (fMRI) [2]. Among all of these devices, EEG is preferred mostly due to its mobile, safe, and cheap nature. In general, EEG is used to evaluate different types of neurological disorders including Alzheimer's disease, tumors, and various sleep disorders [3]. EEG works by the placement of electrodes that are in the form of "small metal discs with thin wires" onto the scalp of the patient [3]. These electrodes are able to detect the very small electrical charges that occur as a result of ongoing brain activities and transform these charges into electronic graphs by amplifying them at the same time [3]. Therefore, EEG as a testing method is frequently used in the detection of epileptic seizures that may occur in different parts of the brain. EEG can detect these seizures by its single or multiple electrodes which correspond to single or multiple channels placed on different parts of the scalp.

In the literature review, it was observed that the detection of epileptic seizures is an important study and research area, and a wide range of articles focus on the evaluation of epileptic seizures using different machine learning (ML) tools and algorithms [4], [5], [6]. Considering the importance of distinguishing between healthy and epileptic brain signals, most of the studies perform binary classification applications although the datasets that they utilize may have multiple classes for determining the type of epileptic seizures according to different metrics [6], [7], [8]. Hence, our term project is aimed to perform a binary classification task that aims to classify between brain signals with and without epileptic seizure as an extension of the state-of-the-art (SOTA) ML studies by utilizing two datasets that are frequently used by research articles in the fields of EEG and epilepsy. As a part of its problem description, our project will consist of three ML algorithms that are namely Decision Tree (DT), K-Nearest Neighbors (kNN), and Multilayer Perceptron (MLP). Our project modules will be designed as generically as possible in order to be applicable to any dataset that consists of feature (X values) and target (Y values) variables that are related to EEG measurements and their labels (healthy/epileptic) respectively. Moreover, one of the aims of our project is to analyze and report the accuracy of the classification results of participants that are epilepsy patients with seizure-free recordings as well as the vice versa results hence exploring the overall accuracy and classification power of our models [2]. This will lead our project to evaluate both the quality of the datasets and our method's performance [2]. From a general perspective, our generic data preprocessing and ML models will enhance and automatize the epileptic seizure detection process performed by checking the EEG recordings that are applied in clinical examinations.

2. Description of Dataset

For the first half of the project, a well-known, high volume, and publicly available dataset under the name of "**The Bonn EEG Time Series Dataset**" is utilized [9], [10]. This dataset is utilized in various SOTA studies that focus on epileptic seizure detection using EEG recordings [6], [11], [12]. This dataset is preferred due to its organized structure as well as the variety and abundance of its epileptic and healthy EEG recordings. In means of its properties, continuous multichannel EEG recordings were collected from a number of patients for this dataset; however, single-channel EEG signals of 23.6 seconds duration were used for investigation after visual inspection for artifacts resulting from muscle activity or eye movement [2]. The continuous EEG signals were sampled with a frequency of 173.61 Hz, and a bandpass filter with cut-off frequencies at 0.53 Hz and 40 Hz was used

to take the most useful frequencies of these signals [2]. After the pre-processing of the raw EEG signals, a dataset of size 500 x 4097 (number of patients x number of samples) was created for the study and for further studies. This dataset is mainly created for clinical and neurological investigation and research purposes [9]. Furthermore, it is used intensively in ML studies on epilepsy and EEG in recent studies [6], [13], [14], [15]. As briefly explained before, this dataset consists of EEG recordings of 500 participants. The dataset consists of 5 sets of conditions that each contain recordings from 100 participants: 2 sets are the EEG recordings obtained from the brain surfaces of the healthy participants with eyes-open and eyes-closed conditions. The other 2 sets are the intracranial EEG recordings (taken from inner regions of the brain) obtained from epilepsy patients placed in both neutral and seizure-stimulating experimental environments during their "seizure-free" condition. The last set also consists of intracranial EEG recordings obtained from epilepsy patients during their "seizure" condition [2]. By pre-processing and analyzing this dataset with the chosen tools and ML algorithms, our project's main aim is to perform a differentiation between EEG signals that contain and do not contain the "epileptic seizure" condition.

Importantly, a better-organized version of this dataset was obtained from Kaggle in the form of a "csv" file [16]. In this file, every 4097 data points were shuffled and divided into 23 groups in accordance with the 23.6 seconds duration for the EEG signals in the original dataset so that now these groups contain 178 data points (4097/23 = 178.13…≈ 178) lasting for 1 second [16]. As a result of this grouping, each data point got transformed to represent the EEG recording value at a different and unique point in time [16]. Moreover, the number of rows in the original dataset increased to 11,500 (23×500 = 11,500) corresponding to lines of information in the dataset. Each line (row) of information contains "178 data points for 1 second" [16] corresponding to different columns. A final column was created to label each row according to 5 categories that correspond to 5 labels. Among these labels, class 1 represents the patients that had an epileptic seizure whereas classes 2, 3, 4, and 5 represent the patients who did not have an epileptic seizure. Hence, the potential of this dataset being utilized for a binary classification problem was observed in accordance with a great number of SOTA studies performing binary classification instead of a 5-category classification. Consequently, we utilized this dataset in the first half of our project to perform binary classification in which class 0 represents the subjects that do not have an epileptic seizure whereas class 1 represents the subjects that have an epileptic seizure. In the figures below, more detailed explanations regarding the initial 5-classes, and the initial dataset that has 5 classes can be observed respectively.

```
Explanations for 5 Classes in the Original Dataset:

Class 1 - Epileptic seizure condition

Class 2 - Neutral experimental environment condition for the seizure-free subject during the recording of the EEG signals

Class 3 - Seizure-stimulating experimental environment condition for the seizure-free subject during the recording of the EEG signals

Class 4 - Eyes closed condition for the seizure-free subject during the recording of the EEG signals

Class 5 - Eyes open condition for the seizure-free subject during the recording of the EEG signals
```

Fig. 1: More detailed explanations for the 5 classes in the dataset

Unnamed	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
X21.V1.791	135	190	229	223	192	125	55	-9	-33	-38
X15.V1.924	386	382	356	331	320	315	307	272	244	232
X8.V1.1	-32	-39	-47	-37	-32	-36	-57	-73	-85	-94
X16.V1.60	-105	-101	-96	-92	-89	-95	-102	-100	-87	-79
X20.V1.54	-9	-65	-98	-102	-78	-48	-16	0	-21	-59
X14.V1.56	55	28	18	16	16	19	25	40	52	66
X3.V1.191	-55	-9	52	111	135	129	103	72	37	0
X11.V1.273	1	-2	-8	-11	-12	-17	-15	-16	-18	-17
X19.V1.874	-278	-246	-215	-191	-177	-167	-157	-139	-118	-92
X3.V1.491	8	15	13	3	-6	-8	-5	4	25	41
X3.V1.6	-5	15	28	28	9	-29	-41	-19	14	30
X21.V1.724	-167	-230	-280	-315	-338	-369	-405	-392	-298	-140
X7.V1.162	92	49	0	-32	-51	-65	-37	-19	-25	-29
X1.V1.211	15	12	0	-17	-28	-31	-39	-51	-44	-35
X1.V1.615	-24	-15	-5	-1	4	3	6	10	11	7
X22.V1.242	-135	-133	-125	-118	-111	-105	-102	-93	-94	-90
X1.V1.863	39	41	41	42	43	43	46	47	49	50
X9.V1.302	9	4	-5	-10	-22	-30	-33	-43	-41	-40
X7.V1.541	-21	-5	1	7	19	20	13	2	-1	-3

Fig. 2: A portion of the dataset representing the first 20 rows and their corresponding feature columns ranging from X1 to X10

y
4
1
5
5
5
5
4
2
1
4
5
1
3
4
2
3
2
3
4

Fig. 3: A portion of the dataset representing the labels of the first 20 rows

3. Feature Engineering and Data Preprocessing

In the project, Python was utilized as the main programming language due to its flexibility, simple coding structure, available data processing libraries, and user-friendly coding environments. In the feature engineering and data preprocessing parts, the Pandas and NumPy libraries were mainly used in order to manipulate the data. Once the aforementioned dataset was obtained as a “csv” file,

the classes were converted to a binary form as explained previously. This was performed by considering class 1 labels as the “epileptic seizure” condition and transforming the other classes ranging from 2 to 5 to 0 in order to consider them as the “seizure-free” condition or in other words the “healthy” condition. This transformation was realized right after the dataset was read as a Pandas DataFrame. The dataset was stored both as a whole and by splitting it into two DataFrames that represent the feature and target variables (labels) respectively for different feature engineering and data preprocessing applications.

Firstly, the rows of the dataset that correspond to different subjects were visualized by plotting all of their data points. Hence, data visualization was performed by the obtainment of these plots in order to perform feature engineering correctly. In the figure below, four plots obtained for two healthy and two epileptic seizure conditions can be seen.

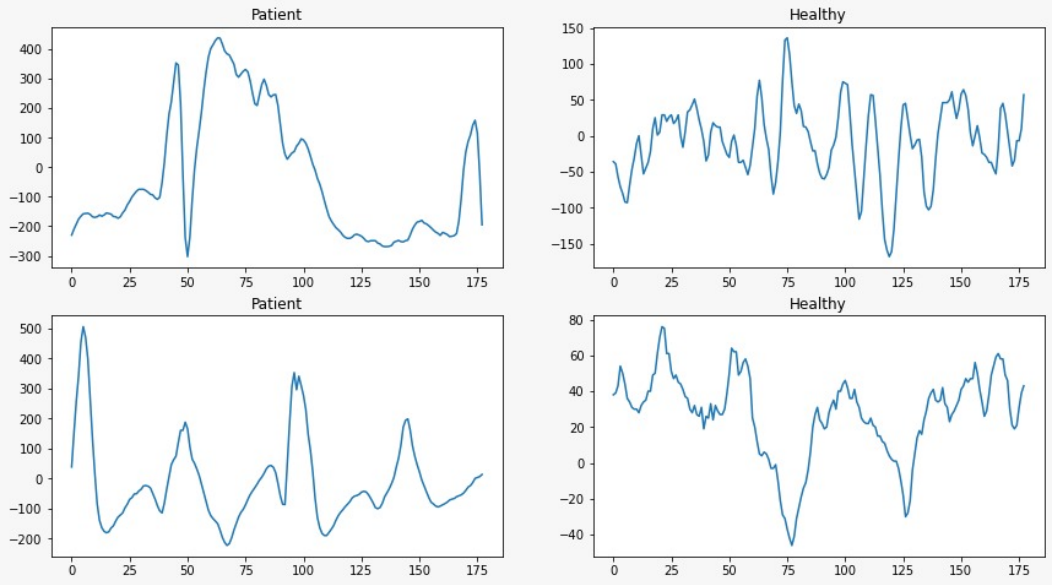

Fig. 4: Data visualization for two healthy and two epileptic seizure conditions

Through this visualization, it was observed that the amplitudes of the epileptic seizure condition are much greater than the healthy condition. In order to take advantage of these amplitude differences, 8 new nonlinear features were derived which correspond to the maximum of each row represented as a new feature (the “MAX” feature), the square of these maximum values (the “MAX2” feature), the cube of these maximum values (the “MAX3” feature), the minimum of each row represented as a new feature (the “MIN” feature), the square of these minimum values (the “MIN2” feature), the cube of these minimum values (the “MIN3” feature), the difference between the “MAX” and “MIN” columns represented as a new feature (the “MxMnDif” feature), and the square of these difference values (the “MxMnDif2” feature). Hence, nonlinear transformations (considering that “max”, “min”, and their squares and cubes are all nonlinear) were applied to the already-existing 178 features of the dataset while the utilized models are linear. The ML models were trained and tested using all of the 178 features versus using just the derived 8 features, and it was observed the performance of the new features was better although they are fewer in number. This verified that the performed feature engineering was successful and the 8 features were used in the rest of the training and test processes. Moreover, the big feature number of 178 was decreased which was causing the training processes of the models to take long amounts of time, and the most beneficial relationships among the data points were represented.

In the data preprocessing part, the features of the dataset were normalized considering that they were in raw condition, the whole dataset (now along with their labels) was shuffled, and the whole dataset was separated into train and test sets with the written generic Python functions that can work with any dataset. The z-score standardization was used as the normalization metric considering that min-max normalization can be problematic if the minimum and maximum values are equal to

each other. The z-score standardization was applied in a column-wise manner by finding the mean and standard deviation of each column, subtracting this mean from each element of the column, and dividing each element of the column by the found standard deviation. The feature engineering and data preprocessing parts were completed through these steps. Finally, the mathematical equations applied for performing z-score standardization can be observed in the following figure.

$$z = \frac{x - \mu}{\sigma}$$

$$\mu = \text{Mean}$$
$$\sigma = \text{Standard Deviation}$$

Fig. 5: The application of z-score standardization (normalization)

$$\bar{x} = \frac{\sum x}{N}$$

$\sum x =$ the sum of x

$N =$ number of data

Fig. 6: The formula of the mean (μ)

$$\text{SD} = \sqrt{\frac{\sum |x - \bar{x}|^2}{n}}$$

Fig. 7: The formula of the standard deviation (σ)

4. Implemented Machine Learning Approaches

In this project, we decided to implement three machine learning algorithms: **Decision Trees (DT), k-Nearest Neighbors (kNN), and Multilayer Perceptron (MLP)**. kNN is a long-standing, well-studied, and highly powerful algorithm in classification tasks. Moreover, it is a fast algorithm since it is not a gradient-based iterative algorithm and it can be implemented in a vectorized manner. DT is also another powerful algorithm that does not rely on iterative weight updates. It is also flexible in terms of tasks, easy to interpret and visualize, and does not require costly data preparation. MLPs are indeed highly nonlinear multivariate functions of unknown variables. These unknown variables are learned through the gradient descent on the loss which is the function of these unknown variables and data to process.

As part of the progress report, we have implemented the DT and kNN algorithms and presented our results. In the final report, MLP also will be implemented and its results will be provided.

4.1 Decision Tree (DT) Algorithm

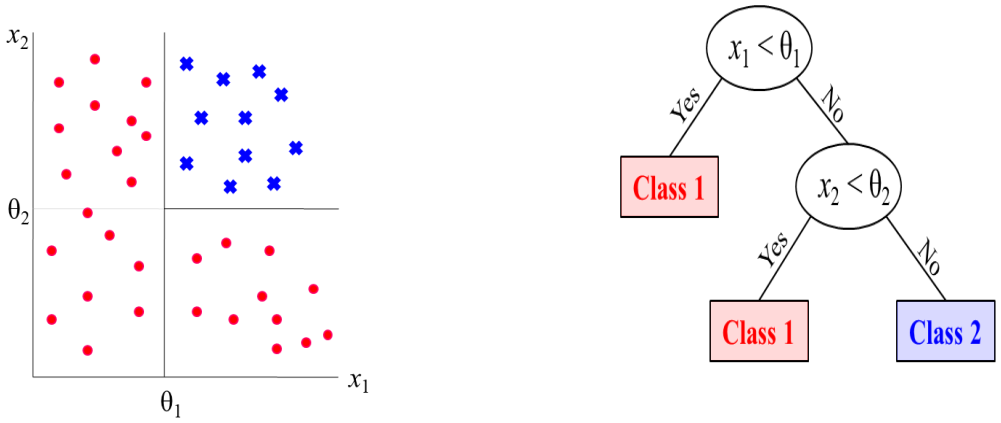

Fig.8: DT for binary classification [17]

DTs are built in the form of a tree structure that provides a model for classification and regression. DT corresponds to dividing the input space into different regions, each of which stands for a decision. In other words, in the DT algorithm, mainly, the goal is to find these decision regions. In DT, internal and leaf nodes correspond to the test function and final decision (a class for classification, and a value for a regression), respectively.

In the training stage, mainly, the goal is to build a tree with a possible minimum error. Starting from the root node, at each step, the "best split" is selected among all possible choices and iteratively continued until all leaf nodes are pure. It is worth indicating that "pure" in this context means that one class exists under the leaf node. In other words, entropy in a leaf node is 0.

In a classification tree, how to find the best split?

First, decide on the impurity function: **Entropy or Gini**

$I(m)_{entropy} = -\sum_{i=1}^{i=N_c}(P(C_i)_m * logP(C_i)_m)$,

$I(m)_{gini} = 1 - \sum_{i=1}^{i=N_c}(P(C_i)_m)^2$, where m and N_c are a feature and number of classes, respectively.

Second, calculate the total split impurity using one of the above impurity functions.

$I_{total}(S) = P_{left}I(left) + P_{right}I(right)$, where P_{left} and P_{right} correspond to the relative number of elements in the left and right branches.

Finally, select the best split which has the lowest total impurity and iteratively repeat this process until all leaf nodes are pure. By taking into consideration these, the training process of DT is an exhaustive process.

How to avoid overfitting?

Two options can be applied to refrain from overfitting: **Deterministic tree length and information gain.**

For deterministic tree length, a predefined maximum tree length is defined and splitting is stopped when the maximum length is reached. For information gain, if the information gain in a split

is less than a threshold value, then splitting is stopped and the majority class in the node is evaluated as the decision class. In our study, we used pruning-based information gain to avoid overfitting.

4.2 k-Nearest Neighbors (kNN) Algorithm

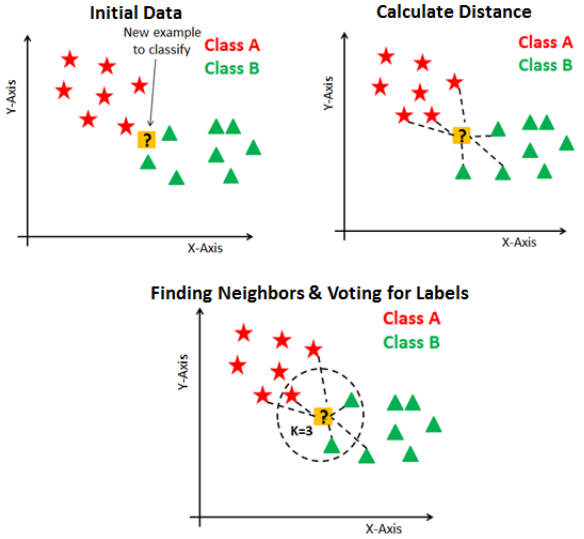

Fig. 9: k-Nearest Neighbor algorithm for binary classification [18]

kNN is a highly simple but judicious algorithm. In Fig. 9, the process of binary classification of a single test sample is illustrated. It can also be applied to multiclass problems without loss of generalization.

Pseudocode:

Input: $X \in \mathbb{R}^{N \times p}$ (Training Set), $Y \in \mathbb{R}^{N}$ (Training Labels), $x \in \mathbb{R}^{p}$ (Test sample), y (test sample labels)

Output: $y_{predict}$

$y_{predict}$=[]
for each test sample x
    distance_all=[]
    for n = 0 → N:
        distance = distance_metric(X_n*,* x*)*
        distance_all.append(distance)
    end
    indices= choose(sort(distance, "ascending"),k)
    y_best = max(frequency(Y[indices]))
    $y_{predict}$*.append(y_best)*
return $y_{predict}$

In the above pseudocode, an algorithm for predictions of a set of test samples is shown. After collecting the predictions for each test sample, predicted labels $y_{predict}$ are compared to true labels y and test accuracy is calculated.

4.3 Multilayer Perceptron (Feed-forward Neural Networks)

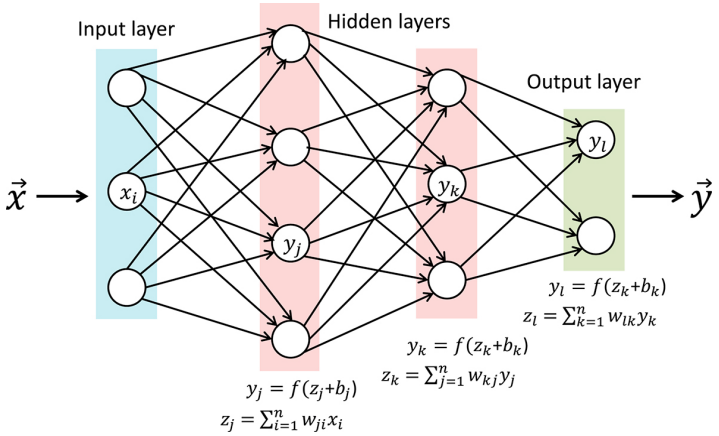

Fig. 10: Feed-forward Neural Network for Binary Classification [19]

The last algorithm that will be implemented for the final submission is MLP, also named as feed-forward neural networks. In MLP, a multidimensional input vector $x \in \mathbb{R}^p$ is fed to the network. The input vector is then mapped to a different multidimensional vector using a weighted sum of input features and a nonlinear activation function. This procedure is iteratively continued until the output layer. At the output layer, the loss of predicted value and true value is calculated. In binary classification tasks, binary cross entropy loss is used commonly. Moreover, the network weights are learned through backpropagation, which is a gradient-based learning algorithm, iteratively. In Fig.10, the forward propagation of input values from the input layer to the output layer is shown.

5. Simulation Results

In this report, we present the simulation results of the detection of epileptic seizures using DT and kNN algorithms. In each experiment, we split the data as training and test data by a ratio of 0.7/0.3. In order to select the best-performing hyperparameters, we use k-Fold cross-validation with grid-search of hyperparameters.

5.1 Decision Tree (DT) Algorithm

In DT, we used 5- Fold cross-validation to select the best hyperparameter combination and evaluated the best model on the test data. 5-fold cross-validation is applied on the training set, which is % 70 of the entire dataset, and evaluations are made on the separate test set, which is %30 of the entire dataset. In Fig.11, classification accuracies are shown with respect to **k** (pruning rate) and **impurity criteria**. In Fig. 13, a grid-search with 5-Fold cross-validation results are shown. Results show that a significant pruning rate brings about underfitting of the model. ***Also, the total time for 5 -fold is: 2252,52 seconds.***

k	entropy	gini
0.0	94.647343	94.357488
0.1	95.748792	95.739130
0.2	95.748792	80.009662
0.3	95.748792	80.009662
0.7	80.009662	80.009662
0.8	80.009662	80.009662
0.9	80.009662	80.009662

Fig.11: 5-Fold cross-validation accuracies on the training set

```
--------------- TOTAL ACCURACY ---------------
Accuracy is 95.48
********* CLASS BASED ACCURACY***************
Class 0
Accuracy is 97.06
Class 1
Accuracy is 89.18
```

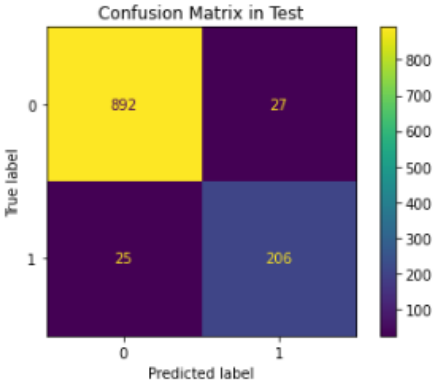

Fig.12: Total and class-based accuracies on the best model with prune rate=0.3 and “entropy” on the test set

```
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0  Impurity:  entropy :  94.64734299516907
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0  Impurity:  gini :  94.3574879227053
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.1  Impurity:  entropy :  95.7487922705314
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.1  Impurity:  gini :  95.73913043478261
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.2  Impurity:  entropy :  95.7487922705314
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.2  Impurity:  gini :  80.0096618357488
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.3  Impurity:  entropy :  95.7487922705314
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.3  Impurity:  gini :  80.0096618357488
Prune Rate: 0.7  Impurity:  entropy :  80.0096618357488
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.7  Impurity:  gini :  80.0096618357488
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.8  Impurity:  entropy :  80.0096618357488
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.8  Impurity:  gini :  80.0096618357488
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.9  Impurity:  entropy :  80.0096618357488
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
Prune Rate: 0.9  Impurity:  gini :  80.0096618357488
```

Fig.13: Grid-search with 5-Fold cross-validation on the training set

5.2 K-Nearest Neighbors (kNN) Algorithm

In kNN, we used 5- Fold cross-validation to select the best hyperparameter combination and evaluated the best model on the test data. 5-fold cross-validation is applied on the training set, which is % 70 of the entire dataset, and evaluations are made on the separate test set, which is %30 of the entire dataset. In Fig.14, classification accuracies are shown with respect to **k** (number of nearest neighbors) and **distance metric**. In Fig. 16, a grid-search with 5-Fold cross-validation results are shown. ***Also, the total time for 5 -fold is: 115.24 seconds.***

k	euclidian	manhattan
3	95.97	96.02
5	96.39	96.41
7	96.45	96.43
11	96.49	96.50
15	96.48	96.60

Fig. 14: 5-Fold cross-validation accuracies on the training set

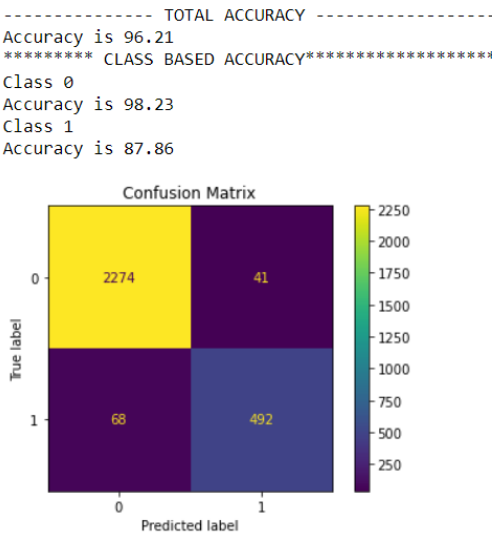

Fig.15: Total and class-based accuracies on the best model with k = 15 and “manhattan” on the test set

```
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
K-Nearest Neighbor: 3  Metric:  euclidian :  95.97
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
K-Nearest Neighbor: 3  Metric:  manhattan :  96.02
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
K-Nearest Neighbor: 5  Metric:  euclidian :  96.39
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
K-Nearest Neighbor: 5  Metric:  manhattan :  96.41
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
K-Nearest Neighbor: 7  Metric:  euclidian :  96.45
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
K-Nearest Neighbor: 7  Metric:  manhattan :  96.43
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
K-Nearest Neighbor: 11  Metric:  euclidian :  96.49
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
K-Nearest Neighbor: 11  Metric:  manhattan :  96.5
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
K-Nearest Neighbor: 15  Metric:  euclidian :  96.48
Fold  1 is completed.
Fold  2 is completed.
Fold  3 is completed.
Fold  4 is completed.
Fold  5 is completed.
K-Nearest Neighbor: 15  Metric:  manhattan :  96.6
Total time for  5 -fold is:  115.23999071121216 seconds.
```

Fig.16: Grid-search with 5-Fold cross-validation on the training set

6. Discussion on the Performance Results

For DT, we used 5-fold cross-validation with a grid-search of pruning rate and impurity criteria. In our implementation, we compared the information gained in a node with the pruning rate and splitting is continued according to their relative values. Results showed that the expectation of significant information gain in a node results in the early stopping of the tree building and underfitting of the model even in the training data. After exhaustively searching for the best classification performance, we tested the parameters of the best model under 5-fold. We observed similar performance in test and training sets and concluded that our model is not overfitting.

For kNN, we applied the scheme in the training and test stages. It is a relatively fast algorithm compared to DT. In kNN, the distance metric does not significantly affect performance. However, selecting the proper **k** (number of nearest neighbors) is crucial. In particular, small values of k show lower classification performance.

7. Future Work for the Final Report

For the future work of the project, a second dataset will be utilized, and it will be trained and tested with the implemented (and planned) ML algorithms. The second dataset has been found but not completely examined and optimized for the binary classification task yet. The second dataset can be

named the "**Epileptic EEG Dataset**" recorded and labeled at the epilepsy monitoring unit of the American University of Beirut Medical Center [10], [20]. The EEG signals were recorded from 6 epilepsy patients in a one-year period. The measurements were taken from multichannel (multiple numbers of electrodes placed on the scalp of the patients) EEG, and they were sampled at 500 Hz [10], [17]. The data was labeled according to the seizure-free condition (class 0), complex partial seizure condition (class 1), electrographic seizure condition (class 2), and video-detected seizure condition (class 3). Hence, it was observed that this dataset can be transformed to be used for a binary classification task as well. As one of the next steps, this dataset will be explored, visualized, preprocessed, and fed into the ML models for training and testing purposes. Moreover, a third ML algorithm will be designed, implemented, and validated in a similar manner to the already-implemented models. The third ML algorithm will be MLP, and it is planned to work with similar preprocessing functions and the validation method of K-fold cross-validation. All in all, in the remaining part of the project, a new dataset will be introduced to the project and the MLP algorithm will be implemented and optimized (such as by hyperparameter tuning), and its performance results will be obtained in means of its accuracy and training duration.

8. Observed and Expected Challenges

In the first half of the project, the observed challenges were regarding the chosen dataset, and the created and simulated ML models. The challenge regarding the dataset was related to its high-volume structure. Since there were 11,500 rows corresponding to the same number of subjects and a total of 178 features, the training duration for the dataset was relatively long at the beginning. This challenge was solved by the application of feature engineering and data preprocessing. Moreover, the challenges regarding the ML algorithms were related to writing them from scratch for the project. In general, it was observed that the ML tools and functions of the libraries such as Sklearn were fast due to their optimized and fast working principles that happen as background processes. Although implementing the kNN architecture was easier compared to implementing the DT architecture, it was seen that writing "for" loops made the Python codes very slow and inefficient. One of the reasons behind the long duration of the training processes was the cumbersome codes. In order to overcome this challenge, the ML algorithms were tried to be implemented by vectorizing the Python codes as much as possible as well as trying to recursive functions instead of loop structures.

For the rest of the project, the expected challenges are regarding the third ML algorithm which will be MLP, and the aforementioned second dataset that is planned to be used. The training duration of the MLP algorithm will be a challenge since high-volume datasets can introduce a great number of neural network layers that can slow down the working processes of the Python codes. Moreover, organizing and utilizing the "Epileptic EEG Dataset" of Beirut Medical Center will be a challenge since it is again a high-volume dataset and its accuracy results were not as good as the results of the current dataset although they were in an acceptable range. The expected challenges are planned to be solved by applying similar strategies which are applying data visualization, feature engineering, and data preprocessing for the high-volume dataset and implementing the third ML algorithm as efficiently as possible again by vectorizing the codes and refraining from long-ranged loop structures.

9. Gantt Chart for Workload

The Gantt chart showing the completed and planned workload can be observed in the figure below.

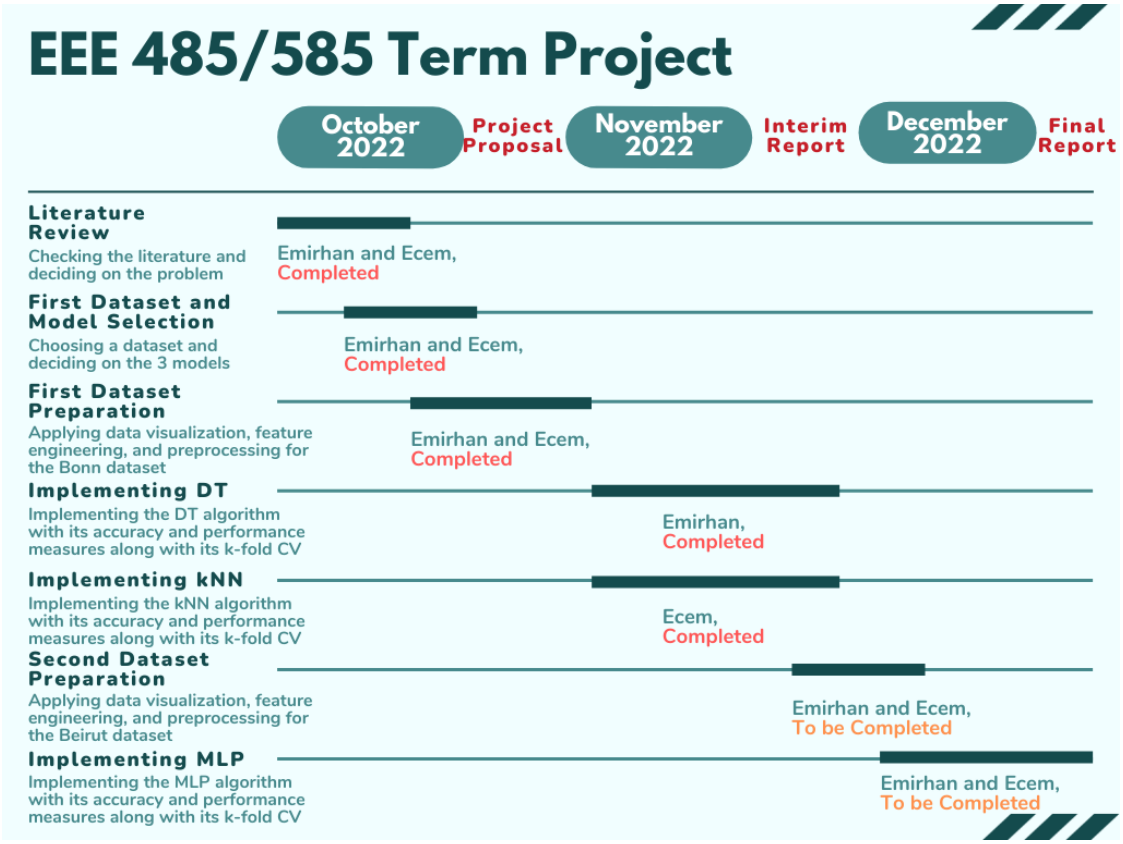

Fig. 17: The Gantt chart

10. Conclusion

In conclusion, two ML models were trained, tested, and validated by using "**The Bonn EEG Time Series Dataset**" with the purpose of performing a binary classification task for the detection of "epileptic seizure" and "healthy" (epileptic seizure-free) conditions in the first half of the project. For the utilized dataset, data visualization, feature engineering, and data preprocessing (normalization, shuffling, and train-test set splitting) were performed by implementing generic Python functions. The train and test sets were then fed into the two ML models which are DT and kNN algorithms. These models were implemented in Python from scratch and their performance results were checked by observing their overall prediction accuracies as well as the class accuracies for 0 and 1. Afterward, k-Fold cross-validation was applied to both models in order to perform a grid-search for hyperparameter tuning (for choosing the optimum pruning rates and impurity functions for DT and for choosing the optimum k and distance metrics for kNN). Both the k-Fold cross-validation functions and the grid-search architectures were written from scratch. The best hyperparameters for both ML models were obtained and under these parameters, it was observed that the accuracy results for the models were around 95-96%. Importantly, the class accuracies were close to each other which verified that the models were not overfitting to the dataset although they have very high accuracies. The confusion matrices were also plotted for both models as a means of an accuracy representation on a graphical structure. Moreover, the training durations of the models were very short (less than 5 minutes). All of these results showed that the feature engineering and data preprocessing procedures, and the model implementations were very successful. In the remaining parts of the project, another dataset is planned to be utilized along with the implementation of a new ML model which will be the MLP algorithm. Hence, two datasets will be preprocessed and applied to three different ML

algorithms by means of training, testing, and validating them with the aim of binary classification for the detection of epileptic seizures at the end of the project as proposed at the beginning of the semester.

11. References

[1] "Epilepsy," mayoclinic.org. https://www.mayoclinic.org/diseases-conditions/epilepsy/symptoms-causes/syc-20350093 (accessed Nov. 19, 2022).

[2] E. Ş, and E. Koç, "Classification of Epileptic Seizure From Electroencephalogram (EEG) Signals Based on Machine Learning Approaches - Term Project Proposal," Bilkent University, Ankara, Turkey, 2022. Accessed on: Nov. 19, 2022. [Online].

[3] "Electroencephalogram (EEG)," hopkinsmedicine.org. https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/electroencephalogram-eeg#:~:text=The%20electrodes%20detect%20tiny%20electrical,provider%20then%20interprets%20the%20reading. (accessed Nov. 19, 2022).

[4] I. Ahmad, X. Wang, M. Zhu, C. Wang, Y. Pi, J. A. Khan, S. Khan, O. W. Samuel, S. Chen, and G. Li, "EEG-based epileptic seizure detection via machine/Deep learning approaches: A systematic review," *Computational Intelligence and Neuroscience*, vol. 2022, pp. 1–20, 2022.

[5] M. Rashed-Al-Mahfuz, M. A. Moni, S. Uddin, S. A. Alyami, M. A. Summers, and V. Eapen, "A deep convolutional neural network method to detect seizures and characteristic frequencies using epileptic electroencephalogram (EEG) data," *IEEE Journal of Translational Engineering in Health and Medicine*, vol. 9, pp. 1–12, 2021.

[6] K. AlSharabi, S. Ibrahim, R. Djemal and A. Alsuwailem, "A DWT-entropy-ANN based architecture for epilepsy diagnosis using EEG signals," *2016 2nd International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, pp. 288-291, 2016.

[7] H. R. A. Ghayab, Y. Li, S. Abdulla, M. Diykh, and X. Wan, "Classification of epileptic EEG signals based on simple random sampling and sequential feature selection," *Brain Informatics*, vol. 3, no. 2, pp. 85-91, 2016.

[8] S. Kumar, R. R. Janghel, and S. P. Sahu "Classification of EEG Signals for Detection of Epileptic Seizure Using Restricted Boltzmann Machine Classifier," in Data Mining and Machine Learning Applications. R. Raja, K. K. Nagwanshi, S. Kumari and K. R. Laxmi, Ed. New Jersey: John Wiley & Sons, 2022, pp. 397–421. Accessed: Nov. 19, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/9781119792529.ch15

[9] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and Brain State," *Physical Review E*, vol. 64, no. 6, 2001.

[10] P. Handa, M. Mathur, and N. Goel, "Open and free EEG datasets for epilepsy diagnosis," *arXiv*, 2021.

[11] S. Ibrahim, R. Djemal, and A. Alsuwailem, "Electroencephalography (EEG) signal processing for epilepsy and autism spectrum disorder diagnosis," *Biocybernetics and Biomedical Engineering*, vol. 38, no. 1, pp. 16-26, 2018.

[12] Ö.Türk, and M. S. Özerdem, "Epilepsy Detection by Using Scalogram Based Convolutional Neural Network from EEG Signals," *Brain Sciences*, vol. 9, no. 5, pp. 1-16, 2019.

[13] H. Al-Hadeethi, S. Abdulla, M. Diykh, R. C. Deo, and J. H. Green, “Adaptive boost LS-SVM classification approach for time-series signal classification in epileptic seizure diagnosis applications,” *Expert Systems with Applications*, vol. 161, pp. 1-14, 2020.

[14] T. G. Altundoğan and M. Karaköse, “EEG Signal Classification with Deep Neural Networks using Visibility Graphs,” *2022 26th International Conference on Information Technology (IT)*, pp. 1-4, 2022.

[15] P. Mathur and V. K. Chakka, “Graph Signal Processing of EEG signals for Detection of Epilepsy,” *2020 7th International Conference on Signal Processing and Integrated Networks (SPIN)*, pp. 839-843, 2020.

[16] “Epileptic seizures dataset,” kaggle.com. https://www.kaggle.com/datasets/chaditya95/epileptic-seizures-dataset (accessed Oct. 21, 2022).

[17] *Bilkent.edu.tr*, 2022. http://www.cs.bilkent.edu.tr/~s.rahimzadeh/CS550.html (accessed Nov. 20, 2022).

[18] “KNN Classification Tutorial using Sklearn Python,” www.datacamp.com. https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn7 (accessed Nov. 4, 2022)

[19] H. Wang, R. Czerminski, and A. C. Jamieson, “Neural Networks and Deep Learning,” *The Machine Age of Customer Insight*, pp. 91–101, 2021.

[20] W. Nasreddine, “Epileptic EEG Dataset,” data.mendeley.com. https://data.mendeley.com/datasets/5pc2j46cbc/1 (accessed Nov. 14, 2022).

APPENDIX

decision_tree

November 20, 2022

[ ]:
```
import pandas as pd
from math import *
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from random import sample
import utils
from utils import shuffling
from utils import normalization
from utils import traintestsplit
from utils import feature_extract
import os
```

[ ]:
```
epilepsy_data = pd.read_csv("bonn_epilepsy.csv", sep =",")

epilepsy_data.drop("Unnamed",axis=1,inplace=True)
epilepsy_data.head()
epilepsy_data.y = epilepsy_data.y==1
epilepsy_data.y = epilepsy_data.y.astype(int)

epilepsy_data= epilepsy_data[epilepsy_data.isnull().any(axis=1)==False]

label = epilepsy_data["y"].astype("category").to_numpy()
label2 = epilepsy_data["y"]
epilepsy_data.drop("y",axis=1,inplace=True)
```

[ ]:
```
epilepsy_data= feature_extract(epilepsy_data)
epilepsy_data= epilepsy_data.iloc[:,-8:]
epilepsy_data
```

[ ]:
```
#normalize the data
normalized = normalization(epilepsy_data, label2)

#shuffle the data
shuffled = shuffling(normalized)
```

```
#split the data into train and test sets
X_train, X_test, y_train, y_test = traintestsplit(shuffled,0.1)
```

[ ]:
```
np_train_data= X_train
np_test_data= X_test

np_label_train=y_train
np_label_test=y_test
```

[ ]:
```
#Define some functions
def impurityCalculation(impurity_choice, labelsOfnode):

    labels, numLabels = np.unique(labelsOfnode, return_counts = True)
    total_labels= np.sum(numLabels)
    p_numLabels = numLabels/total_labels

    if impurity_choice == "entropy":
        entropy = 0
        total_entropy = np.sum([entropy+ (-np.log2(ii)*ii)  for ii in␣
 ↪p_numLabels])
        total_impurity = total_entropy


    elif impurity_choice == "gini":
        entropy = 0
        gini= (1/2)*(1-np.sum(np.square(p_numLabels)))
        total_impurity = gini

    else:
        raise Exception("Sorry", impurity_choice, "is not an impurity type. ")

    return total_impurity

def LeftRightSplit(attribute, label, value,impurity_choice):


    total_entropy = impurityCalculation(impurity_choice,label)

    R_split = label[attribute>value]
    L_split = label[attribute<=value]
    L_prop= len(L_split)/len(label)
    R_prop= 1- L_prop

    L_impurity = impurityCalculation(impurity_choice,L_split)
    R_impurity = impurityCalculation(impurity_choice,R_split)

    split_impurity = L_prop*L_impurity + R_prop*R_impurity
```

```
    informationGain= total_entropy-split_impurity

    return split_impurity, informationGain


def exhaustive_search(attributes,label,impurity_choice):

    splitCheckAll= np.Inf
    gainAll=0
    valueSplitAll=0

    featureIndexSplit=0

    for ft in range(0,attributes.shape[1]):

        splitCheck= np.Inf

        for val in attributes[:,ft]:
            isSplit,gainCheck = LeftRightSplit(attributes[:
 ↪,ft],label,val,impurity_choice)

            if isSplit < splitCheck:
                splitCheck=isSplit
                tempSplitCheck=splitCheck
                splitValue= val
                gainSplit= gainCheck


        if tempSplitCheck < splitCheckAll:
            splitCheckAll=tempSplitCheck
            featureIndexSplit=ft
            valueSplitAll= splitValue
            gainAll=gainSplit


    return splitCheckAll,gainAll,valueSplitAll,featureIndexSplit


def splitNode(attributes,label,impurity_choice):

    bestSplitImpurity, bestSplitInfoGain,bestValue,bestFeatureIndex =␣
 ↪exhaustive_search(attributes,label,impurity_choice)

    node_impurity = impurityCalculation(impurity_choice,label)

    left_node = attributes[attributes[:,bestFeatureIndex]<=bestValue]
    left_node_labels = label[attributes[:,bestFeatureIndex]<=bestValue]
```

```
    right_node = attributes[attributes[:,bestFeatureIndex]>bestValue]
    right_node_labels = label[attributes[:,bestFeatureIndex]>bestValue]


    return left_node,left_node_labels, right_node,right_node_labels,␣
↪node_impurity,bestSplitImpurity, bestSplitInfoGain,bestValue,bestFeatureIndex
```

[ ]:
```
class Node():
    def __init__(self,parent,depth):

        self.parent = parent
        self.depth=depth
        self.childen = []

        self.sampleNumber = None
        self.labelNumbers = []
        self.labelNames= []
        self.bestSplitValue= None
        self.bestSplitIndex= None
        self.splitImpurity = None
        self.isLeaf = 0
        self.leafLabelName= None


def Tree(attributes,label,node,prun,impurity_choice):

    node_labels, labelNumbers = np.unique(label,return_counts= True)
    node.labelNames= node_labels
    node.labelNumbers=labelNumbers
    node.sampleNumber= np.sum(labelNumbers)

    if len(node_labels)==1: # it is pure
        node.isLeaf=1
        node.leafLabelName=node_labels[0]
        return


    else :
        left_node_Feat,left_node_labels, right_node_Feat,right_node_labels,␣
↪node_impurity,bestSplitImpurity,\
        bestSplitInfoGain,bestValue,bestFeatureIndex =␣
↪splitNode(attributes,label,impurity_choice)
        #Preprunning
        if bestSplitInfoGain <= prun:
```

```
            labelname,labelNum= np.unique(label, return_counts=True)
            node.leafLabelName= labelname[np.argmax(labelNum)]
            node.isLeaf = 1
            return
        # You can also add another pruning methods

        node.bestSplitValue=bestValue

        node.bestSplitIndex= bestFeatureIndex
        node.splitImpurity=bestSplitImpurity


        node.L_child= Node(node,node.depth+1)
        node.R_child= Node(node,node.depth+1)
        Tree(left_node_Feat,left_node_labels,node.L_child, prun,impurity_choice)
        Tree(right_node_Feat,right_node_labels,node.R_child,prun,impurity_choice)




def TraverseTree(node,data):

    if node.isLeaf==1:
        prediction = node.leafLabelName

    else:
        if data[node.bestSplitIndex] <= node.bestSplitValue:
            prediction = TraverseTree(node.L_child,data)
        else:
            prediction = TraverseTree(node.R_child,data)
    return prediction
```

[ ]:
```
class DecisionTreeClassifier():

    def __init__(self,criterion,isPrunned="yes"):
        self.beginTree= None
        self.impurity_choice=criterion
        self.isPrunned=isPrunned


    def fit(self,data,label,prun):

        rootNode = Node(None,0)
        if self.isPrunned == "yes":
            Tree(data,label,rootNode,prun,self.impurity_choice)

        else:
```

```
            Tree(data,label,rootNode,0,self.impurity_choice)

        self.beginTree=rootNode

    def predict(self,data):

        if self.beginTree==None:
            print(" You need to create a tree !")

        else:
            prediction_list =[]
            for ii in range(0,data.shape[0]):
                prediction_list.append(TraverseTree(self.beginTree,data[ii,:]))

            prediction = np.asarray(prediction_list)
            return prediction
```

[ ]:
```
def test_accuracy(test_label,predicted_label):

  print("Accuracy is", round(np.mean(test_label==predicted_label)*100,2))
  return round(100*np.mean(test_label==predicted_label),2)

def class_accuracy(model,data,label):
   class_labels = np.unique(label,return_counts=False)
   class_acc_list = []
   for cl in class_labels:
        print("Class", cl)
        class_pred= model.predict(data[label==cl])
        class_acc= test_accuracy(label[label==cl],class_pred)
        #print("Test accuracy for class ", cl, "is : ", class_acc )
        class_acc_list.append(class_acc)


def confusion_matrix_plot(true_label,predictions):


    class_labels = np.unique(true_label,return_counts=False)

    cm= confusion_matrix(true_label,predictions)
    cp=ConfusionMatrixDisplay(cm,display_labels=class_labels)
    cp.plot()
    plt.title("Confusion Matrix in Test")
    plt.show()

  # Confusion Matrix Creation
    '''
    cm= confusion_matrix(label,preds)
```

```
    cp=ConfusionMatrixDisplay(cm,display_labels=class_labels)
    cp.plot()
    plt.title(mode + " Confusion Matrix")
    plt.show()
    '''



# Here I will implement graph plotting
```

[ ]:
```
def KFoldCrossValidation(data_train,label_train,model="decision_tree", Kfold=5,␣
 ↪impurity="entropy", prune_rate=0.1):
    sample_count= data_train.shape[0]
    fold_size = int(sample_count/Kfold)
    accuracy_all = list()

    for ff in range(Kfold):

        test_data=data_train[ff*fold_size:(ff+1)*fold_size,:]
        test_label= label_train[ff*fold_size:(ff+1)*fold_size]

        train_data= np.append(data_train[:ff*fold_size,:
 ↪],data_train[(ff+1)*fold_size:,:],axis=0)
        train_label= np.append(label_train[:
 ↪ff*fold_size],label_train[(ff+1)*fold_size:],axis=0)



        decision_tree= DecisionTreeClassifier(impurity,"yes")
        decision_tree.fit( train_data,train_label,prune_rate)

        preds = decision_tree.predict(test_data)
        acc= np.mean(preds==test_label)
        accuracy_all.append(acc)
        print("Fold ",ff+1,"is completed.")
    return sum(accuracy_all)/len(accuracy_all)*100
```

[ ]:
```
# K-fold cross validation with grid-search
grid_search = [[0,0.1,0.2,0.3,0.7,0.8,0.9],["entropy","gini"]]
accuracy_grid=np.zeros((len(grid_search[0]),len(grid_search[1])))
K_fold=5
import time

b=time.time()
for i,k in enumerate(grid_search[0]):
    for j,m in enumerate(grid_search[1]):
        acc=KFoldCrossValidation(X_train,y_train,Kfold=5,prune_rate=k,impurity=m)
        print("Prune Rate:", k, " Impurity: ", m, ": ",acc)
```

```
        accuracy_grid[i][j]=acc
e=time.time()
print("Total time for ",K_fold, "-fold is: ", e-b, "seconds.")
def organize_results(grid_search,accuracy_grid):

    df=pd.DataFrame(accuracy_grid)
    df.columns = grid_search[1]
    df.set_index(pd.Index(grid_search[0]),inplace=True)
    df.index.name="impurity"
    return df
```

[ ]:
```
# Print the model results

accuracy_table= organize_results(grid_search,accuracy_grid)
import dataframe_image as dfi
dfi.export(accuracy_table, "decision_girdsearch.png")
```

[ ]:
```
# Automatically select the best model

from numpy import unravel_index
best_index=unravel_index(accuracy_grid.argmax(),  accuracy_grid.shape)
decision_tree_best= DecisionTreeClassifier(grid_search[1][best_index[0]],"yes")
decision_tree_best.
 ↪fit(np_train_data,np_label_train,grid_search[0][best_index[1]])
y_pred= decision_tree_best.predict(np_test_data)

print("--------------- TOTAL ACCURACY -------------------")
acc= test_accuracy(np_label_test,y_pred)
print("********* CLASS BASED ACCURACY********************")
class_accuracy(decision_tree_best,np_test_data,np_label_test)
confusion_matrix_plot(np_label_test,y_pred)
```

[ ]:
```

```

kNN

November 20, 2022

[ ]:
```
import pandas as pd
from math import *
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from random import sample
import utils
from utils import shuffling
from utils import normalization
from utils import traintestsplit
from utils import feature_extract
import os
```

[ ]:
```
epilepsy_data = pd.read_csv("bonn_epilepsy.csv", sep =",")

epilepsy_data.drop("Unnamed",axis=1,inplace=True)
epilepsy_data.head()
epilepsy_data.y = epilepsy_data.y==1
epilepsy_data.y = epilepsy_data.y.astype(int)

epilepsy_data= epilepsy_data[epilepsy_data.isnull().any(axis=1)==False]

label = epilepsy_data["y"].astype("category").to_numpy()
label2 = epilepsy_data["y"]
epilepsy_data.drop("y",axis=1,inplace=True)
```

[ ]:
```
epilepsy_data= feature_extract(epilepsy_data)
epilepsy_data= epilepsy_data.iloc[:,-8:]
```

[ ]:
```
#normalize the data
normalized = normalization(epilepsy_data, label2)

#shuffle the data
shuffled = shuffling(normalized)

#split the data into train and test sets
```

```
X_train, X_test, y_train, y_test = traintestsplit(shuffled,0.25)
```

[ ]:
```
np_train_data= X_train
np_test_data= X_test

np_label_train=y_train
np_label_test=y_test
```

[ ]:
```
class kNearestNeighbor:

  def __init__(self,train_data, train_label, k=3, dmetric="euclidian"):

    self.train_data= train_data
    self.train_label= train_label
    self.k=k
    self.dmetric=dmetric


  def distance_metric(self,vector1,vector2):

    if self.dmetric=="manhattan":
      return np.abs(vector1-vector2).sum(axis=1)
    if self.dmetric=="euclidian":
      return np.square(vector1-vector2).sum(axis=1)

  def get_neighbors(self, test_data):

    distances= self.distance_metric(self.train_data,test_data)
    indices= np.argsort(distances)[:self.k]
    return indices

  def predict(self,test_data, test_label):
    y_predict= np.zeros(test_label.shape)

    for tt in range(0,test_data.shape[0]):

      indx= self.get_neighbors(test_data[tt])
      y_indices= self.train_label[indx]
      y_pred=np.bincount(y_indices).argmax()
      y_predict[tt]=y_pred

    return y_predict
```

[ ]:
```
def test_accuracy(test_label,predicted_label):
```

```
  print("Accuracy is", round(np.mean(test_label==predicted_label)*100,2))
  return round(100*np.mean(test_label==predicted_label),2)

def class_accuracy(model,data,label):
   class_labels = np.unique(label,return_counts=False)
   class_acc_list = []
   for cl in class_labels:
        print("Class", cl)
        class_pred= model.predict(data[label==cl],label[label==cl])
        class_acc= test_accuracy(label[label==cl],class_pred)
        #print("Test accuracy for class ", cl, "is : ", class_acc )
        class_acc_list.append(class_acc)

def confusion_matrix_plot(true_label,predictions):


    class_labels = np.unique(true_label,return_counts=False)

    cm= confusion_matrix(true_label,predictions)
    cp=ConfusionMatrixDisplay(cm,display_labels=class_labels)
    cp.plot()
    plt.title("Confusion Matrix")
    plt.show()
```

[ ]:
```
def KFoldCrossValidation(data_train,label_train,model="knn", Kfold=5, k=3,␣
↪metric="euclidian"):
    sample_count= data_train.shape[0]
    fold_size = int(sample_count/Kfold)
    accuracy_all = list()

    for ff in range(Kfold):

        test_data=data_train[ff*fold_size:(ff+1)*fold_size,:]
        test_label= label_train[ff*fold_size:(ff+1)*fold_size]

        train_data= np.append(data_train[:ff*fold_size,:
↪],data_train[(ff+1)*fold_size:,:],axis=0)
        train_label= np.append(label_train[:
↪ff*fold_size],label_train[(ff+1)*fold_size:],axis=0)
        model= kNearestNeighbor(train_data, train_label,k,metric)
        preds = model.predict(test_data,test_label)
        acc= np.mean(preds==test_label)
        accuracy_all.append(acc)
        print("Fold ",ff+1,"is completed.")
    return sum(accuracy_all)/len(accuracy_all)*100
```

[ ]:
```
# K-fold cross validation with grid-search
grid_search = [[3,5,7,11,15],["euclidian","manhattan"]]
accuracy_grid=np.zeros((len(grid_search[0]),len(grid_search[1])))
K_fold=5
import time
b= time.time()
for i,k in enumerate(grid_search[0]):
    for j,m in enumerate(grid_search[1]):
        acc=KFoldCrossValidation(X_train,y_train,Kfold=K_fold,k=k,metric=m)
        print("K-Nearest Neighbor:", k, " Metric: ", m, ": ",round(acc,2))
        accuracy_grid[i][j]=round(acc,2)
e= time.time()
print("Total time for ",K_fold, "-fold is: ", e-b, "seconds.")
def organize_results(grid_search,accuracy_grid):

    df=pd.DataFrame(accuracy_grid)
    df.columns = grid_search[1]
    df.set_index(pd.Index(grid_search[0]),inplace=True)
    df.index.name="k"
    return df
```

[ ]:
```
# Print the model results

accuracy_table= organize_results(grid_search,accuracy_grid)
import dataframe_image as dfi
dfi.export(accuracy_table, "knn_girdsearch.png")
```

[ ]:
```
# Automatically select the best model

from numpy import unravel_index
best_index=unravel_index(accuracy_grid.argmax(),  accuracy_grid.shape)
knn_best=  kNearestNeighbor(np_train_data,np_label_train,␣
 ↪grid_search[0][best_index[0]],grid_search[1][best_index[1]])
y_pred=knn_best.predict(np_test_data,np_label_test)
print("--------------- TOTAL ACCURACY -------------------")
acc= test_accuracy(np_label_test,y_pred)
print("********* CLASS BASED ACCURACY*******************")
class_accuracy(knn_best,np_test_data,np_label_test)
confusion_matrix_plot(np_label_test,y_pred)
```

[ ]:
```

```

utils

November 20, 2022

1 Helper Functions

[ ]:
```
import pandas as pd
from math import *
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from random import sample
import os
```

[ ]:
```
def shuffling(data): #works
    shuf_data = data.sample(frac = 1).reset_index()
    shuf_data.pop('index')
    return shuf_data
```

[ ]:
```
def standardization(data, label): #works, very very slow
    norm_data = data.copy()
    row_num = norm_data.shape[0]
    for column in norm_data.columns:
        col_obj = norm_data[column]
        mu_m = col_obj.sum()/row_num
        std_dev = col_obj.std()
        col_obj = (col_obj - mu_m)/std_dev
        norm_data[column] = col_obj
    norm_data["y"] = label
    return data
```

[ ]:
```
def z_norm(data, label): #works, very slow
    data_z_norm = data.copy()
    for column in data_z_norm.columns: #or don't put .columns, doesn't matter␣
 ↪but it becomes slower
        data_z_norm[column] = (data_z_norm[column] - data_z_norm[column].mean())/
 ↪data_z_norm[column].std()
    data_z_norm["y"] = label
    return data_z_norm
```

[ ]:
```
def normalization(data, label): #works, very fast
    means = data.mean()
    std_devs = data.std()
    nor_data = (data - means)/std_devs
    nor_data["y"] = label
    return nor_data
```

[ ]:
```
def traintestsplit(data,ratio): #works
    row_num = data.shape[0] #row number
    label = data["y"].to_numpy()
    data.pop("y")
    data_n = data.to_numpy()
    if row_num%10 == 0 or row_num%100 == 0 or row_num%1000 == 0:
        test_num = int(row_num*ratio)
        X_test = data_n[0:test_num,:]
        y_test = label[0:test_num]
        X_train = data_n[test_num:,:]
        y_train = label[test_num:]
        return X_train, X_test, y_train, y_test
    else:
        test_num = int(floor(row_num*ratio))
        X_test = data_n[0:test_num,:]
        y_test = label[0:test_num]
        X_train = data_n[test_num:,:]
        y_train = label[test_num:]
        return X_train, X_test, y_train, y_test
```

[ ]:
```
def feature_extract(data):
    max_data = data.max(axis=1)
    max_data_s= max_data**2
    max_data_c= max_data**3

    min_data = data.min(axis=1)
    min_data_s= min_data**2
    min_data_c= max_data**3

    max_min_dif= max_data-min_data
    max_min_dif_s= max_min_dif**2

    data["MAX"] = max_data
    data["MIN"] = min_data
    data["MAX2"]= max_data_s
    data["MIN2"]= min_data_s
    data["MAX3"]= max_data_c
    data["MIN3"]= min_data_c
    data["MxMnDif"] =  max_min_dif
    data["MxMnDif2"] =  max_min_dif_s
```

```
    return data
```