Epileptic Seizure Detection using EEG

Yanqi Hong 5884683 y.hong-8@student.tudelft.nl

Zhixuan Ge 5927633 z.ge-4@student.tudelft.nl

EE4C12 Machine Learning for Electrical Engineering Project

Group 16

October 26, 2023

1 Summary

In this project, a classification of seizure/non-seizure status (Task 2) and a clustering of seizure patients (Task 3) are conducted. General standardization and feature selection methods are introduced first. Some special data-preprocessing methods based on the properties of the data are discussed and tested. Then, we construct multiple structures of classification and tune them with random search for Task 2. After comparing their performance, we choose the Deep Neuron Network (DNN) model with optimized hyperparameters as the final model. In Task 3, we consider the data as a few time series and use Dynamic Time Warping (DTW) to measure the similarities between them for clustering. Different combinations of feature selection and cluster number are tested to find the hyperparameter set with the highest silhouette score.

2 Task 1: Data pre-processing

2.1 Preliminary analysis

The electroencephalogram (EEG) data used in this project has 55456 samples from 15 different patients and 362 columns. Each sample is extracted data from a processed 2-second-long EEG segment. The first column *Patient* is the patient ID and the second column *annotation* is the segment's label. 1 stands for seizure segment, -1 stands for non-seizure segment, and 0 stands for the transition state between seizure and non-seizure, which means segments with 0-labels can have patterns of both seizure and non-seizure. The other 360 columns are 18 statistical patterns in the time domain and the frequency domain from 20 sets of channels.

Fortunately, there are no missing values in the EEG data, so we do not have to handle it. However, we find anomalies in the outlier analysis. We can see some very strange statistics in some features, like *norm_power_HF|P4-O2*. More of them are shown in Table 1.To find out what happened, we plot that feature as a time series and separate it according to patience ID. As we can see in Figure 1. In this case, there are 3 patients with 2 records, so we plot them as two independent time series. It indicates that feature *norm_power_HF|P4-O2* of Patient 5804 looks unnormalized, which conflicts with its variable name. Also, we are not sure whether there is anything wrong with the data of Patient 6175, because his/her 2 records have a great difference in magnitude. They can be recorded by different devices or just someone made a mistake in data processing. In general, we can just exclude these situations are just noises, but we are not sure whether they are reflections of exact patterns or mistakes made in raw data processing. To solve this kind of potential problem, an alternative standardization method is used, besides standardizing the features directly. In this method, We assume that errors like that are common in the data so the data have to be standardized separately according to the time series to which a sample belongs. Specific instructions will be introduced in 3.1.

Besides what we have analyzed above, we can also some peaks which are much greater than the average values of the sequences. But if we look at their neighborhoods closely, we can see the process of the changes. It informs that they are exact patterns of seizure rather than noise or other things that should not be here. Hence, we just leave them here as normal data.

2.2 Pre-processing

Due to the dissimilarity of Task 2 and Task 3, their data pre-processing steps are different. We will explain them separately in 3.1 and 4.1 with more details. This section only introduces some basic concepts of methods used both in Task 2 and Task 3.

	norm_power_HF|F4-C4	norm_power_HF|C4-P4	norm_power_HF|P4-O2
count	55456	55456	55456
mean	0.944334	0.109339	0.109481
std	11.95869	0.518148	0.524319
min	0.000006	0.000012	0.00006
25%	0.003619	0.004051	0.004649
50%	0.014222	0.011859	0.01209
75%	0.102257	0.053111	0.042975
max	566.7368	40.51387	54.89866

Table 1: Statistics of a few features with anomalies

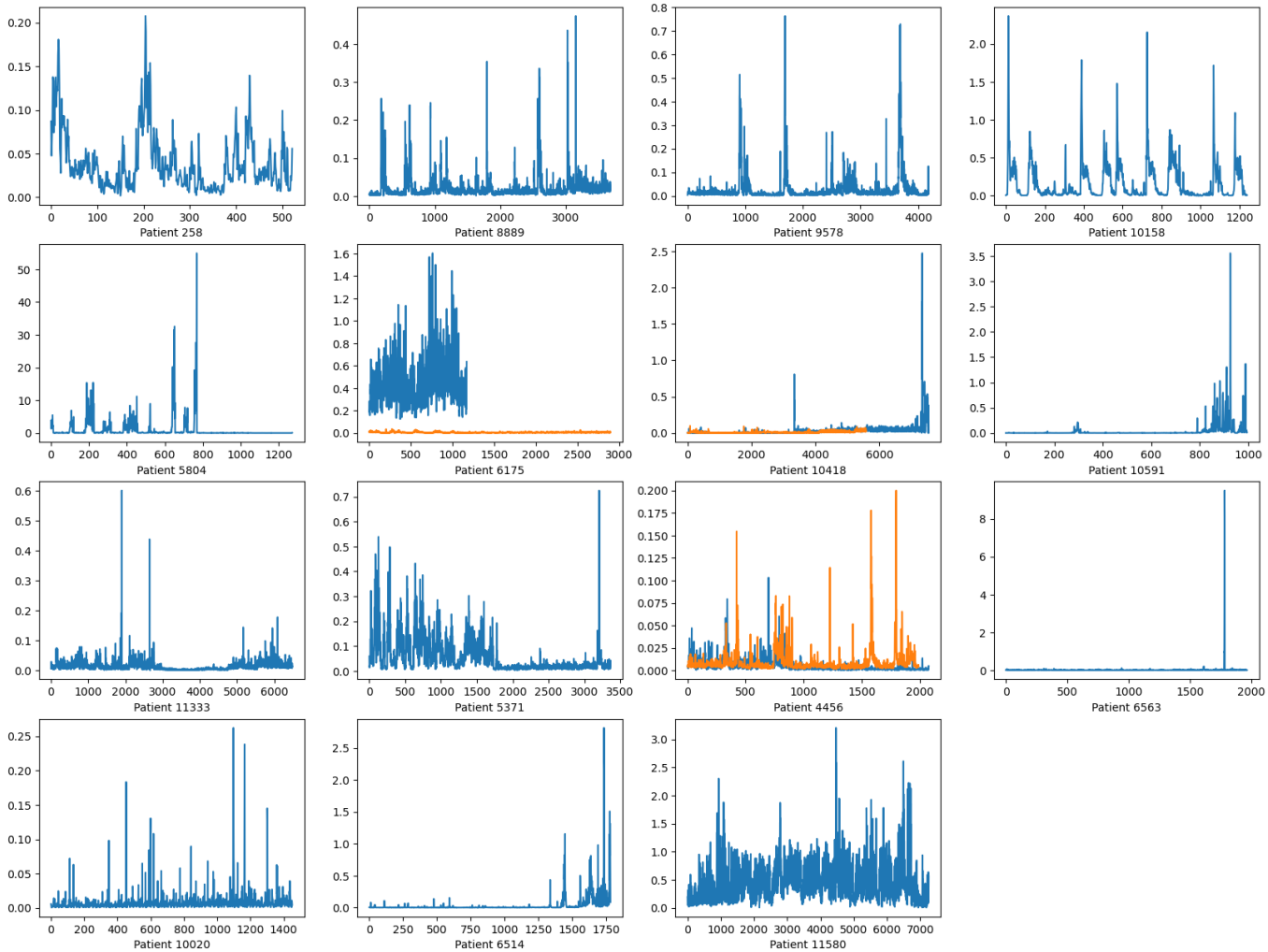

Figure 1: Feature *norm_power_HF|P4-O2* of 15 patients (2 time series are plotted for patients with 2 records)

2.3 Standardization

Generally, standardization and feature selection, are applied in both tasks. Standardization is based on the Z-score, which follows

$$x_{scaled} = \frac{x - \mu}{\sigma} \tag{1}$$

where x is the data before transformation, μ is the mean of the data, σ is the standard deviation of the data, and x_{scaled} is the scaled data. Z-score standardization removes the mean and scales the data to unit variance.

2.4 Feature selection

Employing all 360 features for training the classification model may result in prolonged computational time, as well as issues with the curse of dimensionality and overfitting. The 360 features are not independent of each other and must have redundant information. Appropriate dimensionality reduction steps can extract key information from the raw data, improving performance and decreasing time consumption.

Since we do not have a background in EEG, we just apply feature engineering processes directly and do not care about the name of the features/channels. Two kind of feature engineering methods are used, including Extra Tree Classifier (ETC) and Principal Component Analysis (PCA).

The first method we choose to use is ETC. The ETC works by constructing a large number of decision trees and selecting the features that are the most frequently used as splitting criteria[1]. Utilizing the feature importance list generated by ETC, we can arrange it in descending order to identify the most crucial features. It is clear to distinguish which feature is important in the result (shown in Figure 2) from ETC. Through a series of tests varying the number of features retained in the models, we determined that maintaining 120 features strikes a balance between model performance and computational efficiency.

The second method used is PCA. PCA projects the data to a lower dimensional space by singular value decomposition. Compared with the simple data selection based on importance in ECT, PCA applied a linear transformation to the data[2].

3 Task 2: Seizure/Non-seizure classification

In this task, a model that can classify each new data segment as seizure or non-seizure segments need to be developed. The 55456×362 dataframe are considered as 55456 independent samples in Task 2. Patient ID is ignored, *annotation* is considered as the output of the model with possible values in set $\{-1, 0, 1\}$,i.e., the label of the classification task. The other 360 columns will be the input of the model.

3.1 Data pre-processing

In Task 2, to avoid data leakage of the test set, we split the dataset into the train set and the test set first before standardization. The train set occupies 75% of the dataset and the test set occupies 25%.

After the train-set split process, we apply the Z-score standardization to the inputs. The scaler is fitted on the train set only to avoid test set leakage. However, we use two methods to scale the data because of the situation we described in 2.1. One is scaling the train set input array as a whole, as what is common in the normal classifications, and another is to train 18 scalers for each time series to handle the specificity of the data set. The alternative method of scaling time series separately is denied because the accuracy will be much lower (65%) in trials, compared with 80% of normal standardization. It is clear that normalizing the same feature segmentally will influence weight computation in training, but the problem we analyzed before does exist, so the best way is to check them manually. The features with this problem can be discarded in ECT and we can get a model with a relatively good accuracy, so we decide to take the first plan and consider the problem features as noise with limited impact.

After standardization, feature selection methods based on ETC and PCA are both applied. We conduct a few preliminary tests about two selection methods on basic machine learning models with different structures, finding that models trained on ETC data always perform better than on PCA with the same number of features selected. We check the explained variance ratios of features and find that the ratios of components are relatively close. We conclude that the linear transformation of PCA just mixes them up rather than filters the real principle components. On the other hand, ETC just picks the features related to the labels closely and discards the insignificant ones. ETC performs well because the mixture is not in it. The importance of features is shown in Figure 2.

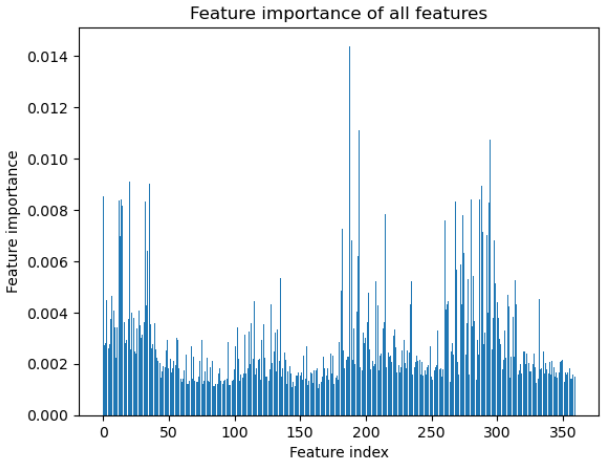

Figure 2: The feature importance of all features based on ETC in train set

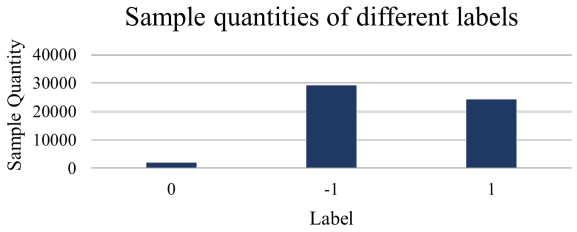

Figure 3: The different sample quantities among 0-labels, 1-labels and -1-labels

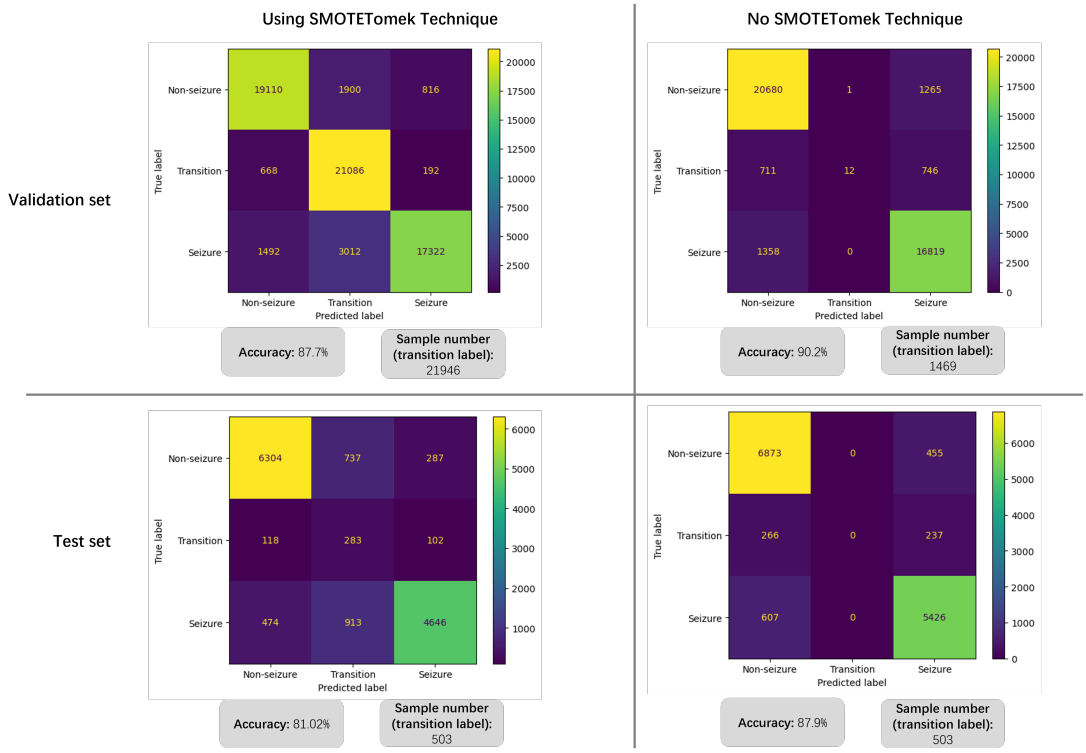

Figure 4: Comparison the confusion matrices of models with and without SMOTETomek in validation and test set

Unbalanced sample problem

Another troublemaker is unbalanced samples. Regarding the labels within the samples, the number of 0-labels (transition period between seizure and non-seizure) is considerably lower compared to the 1-labels (segments indicating a seizure) and -1-labels (segments indicating a non-seizure state). Figure 3 indicates the unbalanced sample problem in the data set. This problem makes our model hardly recognize 0-labels.

We try two methods to address this issue. The first is changing weights. However, it is not a general method because some structures do not accept weights as parameters. It leads to our the second solution: Over-sampling. We introduce SMOTETomek from package imbalanced-learn to realize over-sampling. This method also combines the under-sampling technique to clean some noise samples after over-sampling[3].

Figure 4 indicates the comparison of same machine learning model [1] with and without SMOTETomek technique in the same validation set and test set. According to the comparison result, it is clear that the model with SMOTETomek can distinguish the transition period (minority class) better, while the accuracy in both validation set and test set is worse than the model without SMOTETomek technique. The reason is that the SMOTETomek technique generates approximately 20000 samples with transition-labels into the training set. This amount is ten times the original number of samples for the transition period data. Besides, all generated data is based on the original samples, so that the generated data is similar to each other which may lead to lower performance in the model and over-fitting problem. This situation also happens to other methods including weight adjustments and simple over-sampling.

Given that the primary objective is to classify seizure and non-seizure moments (focusing on the majority class), we have decided to refrain from using SMOTETomek in the final data pre-processing progress to enhance accuracy within the majority class. It is unwise to sacrifice accuracy for seizure detection in exchange for recognizing transition status.

3.2 Model selection

Since the primary objective in this section is to construct a machine learning model for classifying various seizure states, we have chosen to develop the algorithm using four methods: logistic regression, support vector machine classifier, random forest classifier, and fully connected neural network. The first 3 models are realized by the package scikit-learn, and the neural network is constructed by Pytorch on GPU. Besides, all selected methods use cross entropy loss as loss function. Once we have identified the optimal hyperparameters through random search, we will proceed to assess the performance of each model using these refined parameters. This evaluation will guide us in determining the most effective algorithm for our task, based on accuracy and computational resources both. We use accuracy the judge the performance of a model, and the confusion matrix is also plotted to visualize the result.

Figure 5 is the pipeline to find the best model with optimized hyperparameters. A 5-fold cross validation is always applied in training.

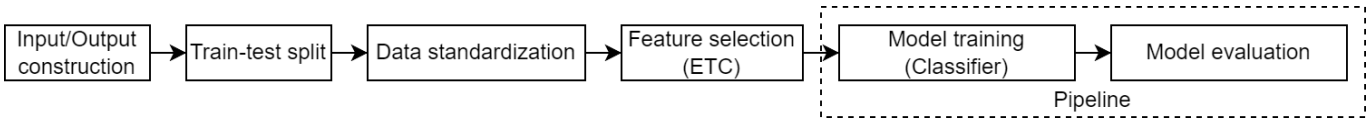

Figure 5: Task 2 Pipeline

3.2.1 Logistic regression

Logistic regression (LR) is a statistical learning method used to address classification problems. Its purpose is to predict the probability of a sample belonging to a particular class, yielding an output between 0 and 1 based on a logistic function as the boundary of different class. We apply random search to find out the best combination of the hyperparameters. The hyperparameter space we choose is listed in Table 2.

hyperparameter	**Explanation**	**Scope**
C	Regularization strength	$C = \frac{1}{10^x}; x \in [-4, 1]$
Penalty	Define the norm of the penalty	l_1, l_2, elastic net, No penalty
l_1 ratio	Elastic-Net mixing parameter	$l_1 \in [0, 1]$

Table 2: Hyperparameter space of Logistic regression

In an LR model, hyperparameters focus on the regularization. C is the reciprocal of the regularization factor. *Penalty* decides the method of regularization. l_1 ratio corresponds to a combination of L1 and L2 penalties if a elasticnet penalty is applied. C is the reciprocal of the regularization factor. The formula of C might look strange because it is the inverse of regularization strength. We want the strength to have the same possibility from 0.01 to 0.1 and from 0.1 to 1, so an exponential distribution is used. More details can be found on [4].

After 500 times random search, we can get the best hyperparameter combination for the logistics regression[2]. The result can be found in Table 6. Figure 6 shows the confusion matrix of this LR model.

[1] DNN model with input layer: 120; hidden layer: (512,512); output layer: 3; activation function: LeakyReLU; epochs: 10; regulation strength: 0.005; learning rate: 0.001

[2] maximum iteration for this logistics regression is 500 and the solver is saga

3.2.2 Support vector machine classifier

The main goal of the Support Vector Machine (SVM) classifier is to identify a hyperplane within the feature space that maximizes the margin between distinct classes. Predictions are then made based on this hyperplane, allowing the SVM to effectively classify data points[5]. Random search are employed on the search of the best combination of the hyperparameters. The hyperparameter space we choose is listed in Table 3.

hyperparameter	**Explanation**	**Scope**
C	Regularization strength	$C = \frac{1}{10^x}; x \in [-4, 1]$
kernel	The kernel type to be used in the algorithm	linear, poly, rbf, sigmoid
γ	Kernel coefficient for rbf, poly and sigmoid	$\gamma \in [0, 1]$

Table 3: Hyperparameter space of Support vector machine classifier

The parameter C is the same as that in LR. In SVM, the parameter *kernel* is used to specify the type of decision boundary. It determines how the similarity or distance between two data points is measured in the feature space. γ is a parameter in SVM that influences the range of the kernel, thereby determining whether the model might overfit or underfit.

After 500 times random search, we can get the best hyperparameter combination for the SVM[3]. The result can be found in Table 7. Figure 6 shows the confusion matrix of this SVM model.

3.2.3 Random forest classifier

The Random Forest Classifier (RFC) constructs multiple decision trees during training and combines their predictions to make the classification task[1]. A random search is employed for the best combination of the hyperparameters. The hyperparameter space we choose is listed in Table 4.

hyperparameter	**Explanation**	**Scope**
n_estimators	The number of trees in the forest	[10,100]
max_depth	The maximum depth of the tree	[5,25]
min_samples_split	The minimum number of samples required to split an internal node	[50,250]
min_samples_leaf	The minimum number of samples required to be at a leaf node	[50,250]

Table 4: Hyperparameter space of Support vector machine classifier

These hyperparameters decide the basic structure of the forest, and more details can be found on [6]. The ranges of hyperparameters are based on the size of the train set and adjust through some trial and error. We also optimize by cutting some depths or the number of trees to decrease training time.

After 50 times random search, we get the best hyperparameter for the RFC. The result can be found in Table 7. Figure 6 shows the confusion matrix of this RFC model.

3.2.4 Deep neural network

The deep neural network (DNN) is used to be the model of classifying the EEG signal. The DNN is composed of multiple hidden layers between the input and output layers. For training DNN, we select PyTorch[7] as the framework to boost our training process, because it can utilize the performance of GPU based on CUDA toolkit.

We construct the basic form of DNN: Full Connect Artificial Neural Network (FC-ANN). It stacks a few hidden layers between the input layer and the output layer. Due to the decided feature number and the output class, the input layer and the output layer are 120 and 3 separately. The hyperparameters we used for random search are listed in Table 5.

[3] maximum iteration for this Support Vector Machine is 500

hyperparameter	Explanation	Scope
hidden layer width	Control the scope of the width of DNN	[8,720]
hidden layer depth	Control the scope of the depth of DNN	[1,3]
activation function	A list of the activation function	ReLU, Tanh, LeakyReLU, Sigmoid
optimizer	The type of the optimizer of DNN	SGD, Adam
learning rate	The step size of updating the model in the beginning	[0.0001,0.5]
weight decay	Regularization strength	$weight\ decay = 10^x,\ x \in [-4, 0]$
epochs	Number of iterations	{5,10,15,20}

Table 5: Hyperparameter space of Deep neural network

For FC-ANN and other DNN, the number of layers and neurons in a neural network impacts its ability to capture complex patterns. In order to achieve better performance on classifying the selected data (120 features) into 3 classes and the computational cost, we set the boundary of the width and the depth which is shown in Table 5. The *activation function* is also a hyperparametes to decide how nonlinearity is added, like the smooth Sigmoid or the sharp ReLU. According to the parameter in the *optimizer*, SGD uses a fixed learning rate, while Adam adjusts the learning rates dynamically, often leading to faster convergence in deep learning tasks.

After 500 times random search, we get the best hyperparameter for the DNN [4]. The result can be found in Table 9. Figure 6 shows the confusion matrix of this DNN model.

Logistic Regression Result	
Best parameters	*penalty*: elastic net $l_1 ratio = 0.7455$ $C = 0.5089$
Best score in validation	0.827322
Accuracy in test set	0.822129

Table 6: The result of LR model

Support Vector Machine Result	
Best parameters	*kernel*: sigmoid $\gamma = 0.0292$ $C = 1.4928$
Best score in validation	0.654382
Accuracy in test set	0.700159

Table 7: The result of SVM model

Random Forest Classifier Result	
Best parameters	$n\_estimators = 90$ $min\_samples\_split = 66$ $min\_samples\_leaf = 50$ $max\_depth = 21$
Best score in validation	0.864421
Accuracy in test set	0.863459

Table 8: The result of RFC model

Deep neural network Result	
Best parameters	*hidden layer*: [142] $weight\ decay = 0.00011467$ *activation function*: Tanh *optimizer*: Adam $learning\ rate = 0.007989$ $epochs = 20$
Best score in validation	0.917725
Accuracy in test set	0.884449

Table 9: The result of DNN model

3.3 Model comparison

We can find that most models with the best hyperparameter work well on the test set, except the SVM model. SVM performs badly not only on the test set but also on the validation set, so it is hard to judge directly whether it is over-fitting or anything else. We try to increase the number of interactions but the performance doesn't increase, so under-fitting is excluded. In short, we believe the SVM model is influenced by the noise of the data, and extra steps are required in the data pre-processing.

Another interesting phenomenon is that the LR and the RFC models fail to differentiate the 0-label completely (shown in Figure 6). The number of 0-labels is relatively small compared with the rest, so features related to them play an insignificant role in feature selection. Hence, it is harder to find the relationship between the input and the 0-labels after feature selection. Fortunately, the focus of Task 2 is to differentiate other labels, so we can ignore it. However, the SVM is so sensitive that it still finds some trace of 0-label, but its sensitivity also makes it hard to

[4] In this DNN, batch size = 1024

tolerate noise, as we see in the last paragraph. On the other hand, our DNN is complex enough to extract some details about 0-labels, so it also does a good job of recognizing 0-labels.

The structure of our DNN model is also inspiring, or we should call it WDD rather than DNN because it just has one hidden layer with 142 neurons, which is relatively wide. Anyway, it illustrates to us that a wide neuron network with 1 layer can solve the problem well enough, but we still believe that the performance can be improved if more research is done.

The DNN model not only has the highest performance but also can be trained faster than other models because the GPU is applied. As a result, it is selected as the final model with an accuracy of 88.4% in the test set.

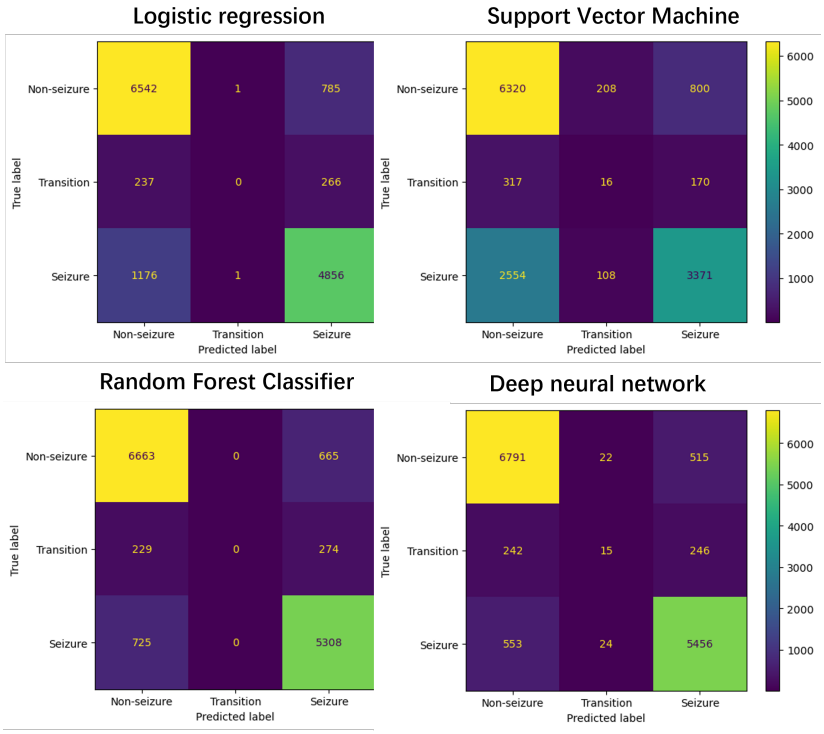

Figure 6: Confusion matrices of different models

4 Task 3: Patient categorization

4.1 Data pre-processing

In Task 3, however, the data will be considered completely different from Task 2. It is an unsupervised learning problem aiming at clustering different patients. The 55456 samples are not considered independent anymore. On the contrary, data of the same Patient ID will be taken as a time series, sorted by their indices. Here comes our first problem: for Patient 4456, 6175, 10418, there are two segments of continuous series in the dataframe. We handle the two segments as different time series, which means we will extract 18 time series from 15 patients. This step introduces a potential risk, in which two segments from the same patient are clustered into different subgroups. We will analyze it later.

Besides, the train-test split does not exist anymore because we use the silhouette score[8] to evaluate the result rather than the scores we used in Task 2. The 360 features are considered as different channels of the time series. However, one extra channel, *annotation*, which is used as labels in Task 2, is taken into account in Task 3. It is viewed as a time series with discrete values. Standardization and feature selection are basically the same as in Task 2, except for the removal of the train-test split. Choices about standardization and over-sampling are the same as in Task 2, but ETC and PCA are both used here to see if they can function as the data structure changes. Figure 7 is the pipeline of Task 3 to decide the feature selection method and the number of clusters.

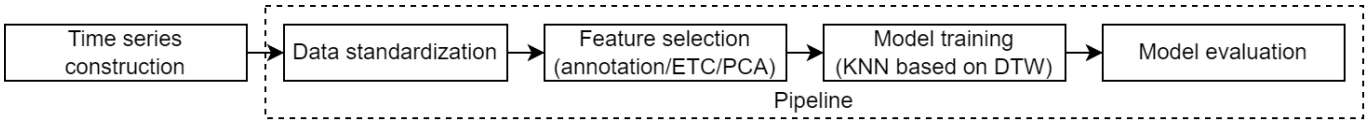

Figure 7: Task 3 Pipeline

4.2 Model introduction

In Task 3, we use K-Nearest Neighbors algorithm (KNN) for time series[9] to cluster patients into subgroups. In 4.1, we have transformed the original data into 18 time series with different lengths and multiple channels. One of the crucial points is how to define the "distance" between 2 time series, especially the inconsistent lengths. The normal Euclidean distance is obviously unavailable and makes no sense.

We import the Python package tslearn for time series analysis and apply the metric Dynamic Time Warping (DTW) to measure the distance among time series. It aligns 2 series and finds the optimal path to calculate the similarity between them. It is formulated as below:

$$DTW(X,Y) = \min \sqrt{\sum_{(i,j)\in\pi} d(x_i - y_j)^2} \tag{2}$$

where $x = (x_0, \ldots, x_{n-1})$ and $y = (y_0, \ldots, y_{m-1})$ are 2 series with the length of n and m respectively and share the same number of dimensions, $\pi = [\pi_0, \ldots, \pi_K]$ is the path which starts from $\pi_0 = (0,0)$ and stops at $\pi_K = (n-1, m-1)$. For all $k > 0$, $\pi_k = (i_k, j_k)$ is related to $\pi_{k-1} = (i_{k-1}, j_{k-1})$ as follows:

$$i_{k-1} \leq i_k \leq i_{k-1} + 1$$

$$j_{k-1} \leq j_k \leq j_{k-1} + 1$$

To conclude, DTW finds the best alignment path that minimizes dissimilarity between two sequences, then calculates the Euclidean distance between aligned time series is minimal[10].

4.3 Evaluation metrics

In Task 3, the silhouette coefficient is chosen as the evaluation metric. It is given as

$$s = \frac{b - a}{\max(a, b)} \tag{3}$$

where a is the mean distance between a sample and all other points in the same class and b is the mean distance between a sample and all other points in the next nearest cluster. A high Silhouette Coefficient indicates that the data point is well-matched to its own cluster and poorly matched to neighboring clusters. Also, we will check the curves of the time series to analyze if the clustering is reasonable. It is not a quantitative metric but it improves the interpretability of the model.

4.4 Model tuning

We use the feature selection method and the number of clusters as the hyperparameters of the model. Three kinds of feature selections are applied. First, we just use the *annotation* only. Second, an ECT model is trained on the 360 features and their importance is sorted. We choose the previous *n_features* of the features with the highest importance and *n_features* can be a number in the set {1,18,36,72,120,180,360}. Besides, the feature selection will degrade to using all 360 original features directly if $n\_features = 360$, because no transform is applied and there is only feature discarding in ECT. Third, the PCA transformation with the same parameters is also applied, and it shares *n_features* with ECT.

The result can be seen in Table 10. We can see models always get their best performance at $n\_features = 1$. It can be the result of too many features bringing much conflicting interference. Hence, if we just extract the most crucial features and discard the rest, we can get the highest performance. If we choose ECT with $n\_features = 1$ and $n\_clusters = 5$, we can get the best model. The clustering result is in Figure 8. This model manages to classify the 2 records of Patient 6175 into the same group but fails in Patient 4456 and 10418. There are 2 possible reasons

causing this. It can either the model is not general enough or the 2 records of the same patient can be different in patterns exactly, which means their seizure status has characteristics of multiple clusters. In conclusion, we can see the patients can be classified into 5 subgroups, but there are 2 patients having characteristics of both group 0 and 2. Hence, we can set up a new group to classify them, so we have 6 groups in total.

selection methods	*n_clusters*	*n_features*						
		1	18	36	72	120	180	360
annotation	3	0.460314						
	4	0.286254						
	5	0.271606						
ETC	3	0.570234	0.004588	-0.17914	-0.02579	-0.16445	-0.12922	-0.08612
	4	0.580941	0.014176	-0.15159	-0.09956	-0.14166	-0.11401	-0.08168
	5	0.58955	0.022047	-0.11489	-0.0746	-0.10497	-0.0828	-0.06213
PCA	3	0.458774	-0.1226	0.067956	-0.07302	-0.08669	-0.08475	-0.08612
	4	0.479859	-0.07818	-0.08096	-0.0751	-0.09176	-0.08037	-0.08168
	5	0.206326	-0.06596	-0.06582	-0.07139	-0.09535	-0.06153	-0.06213

Table 10: Silhouette scores of hyperparameter combinations

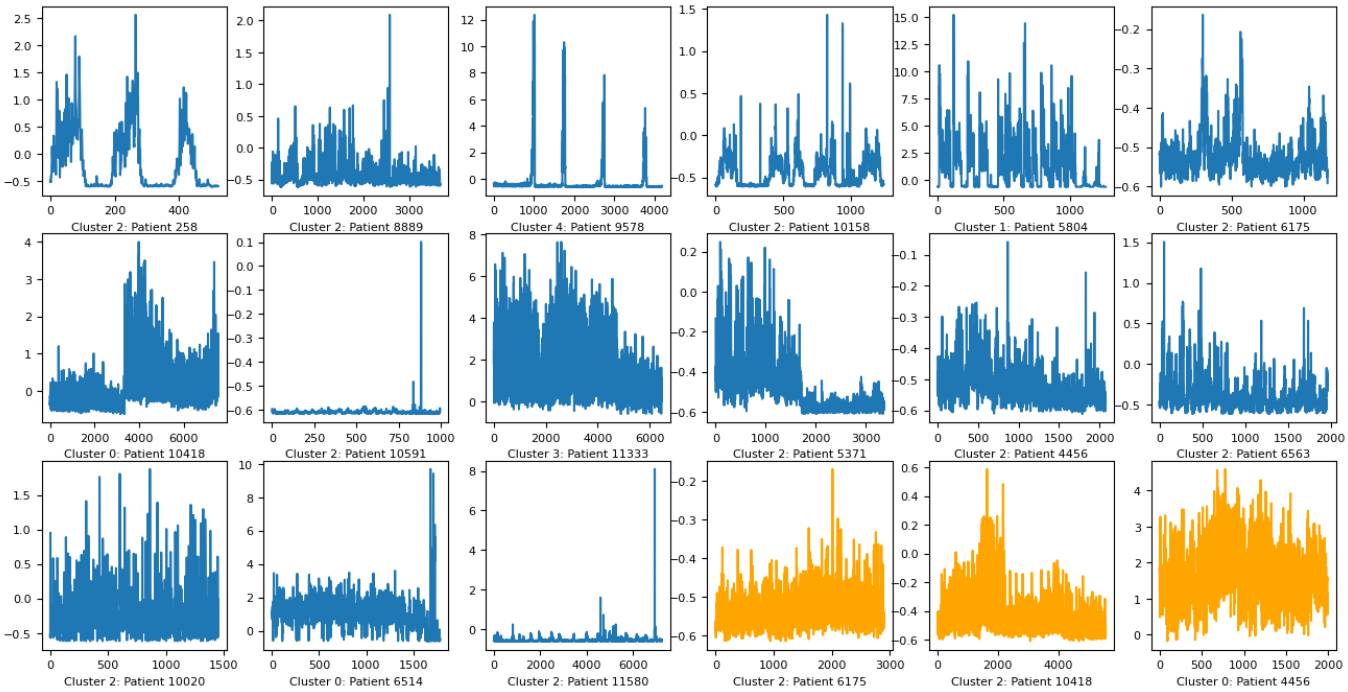

Figure 8: Clustering result for each time series

5 Conclusion

We get a DNN model with high performance for Task 2 in the end, but it is still a way from perfect. A more elaborate data analysis of each channel can lead to a better standardization method, which improves the upper bound of the model performance. Also, we give up recognizing 0-labels for it is not our object, but methods that balance the accuracy and resolution of 0-labels can be found through more complex research. Our model for Task 3 is still fundamental though *silhouette* = 0.59 indicates a reasonably good clustering. More things can be done to improve the interpretability.

References

[1] *scikit-learn forest User Guide*. URL: https://scikit-learn.org/stable/modules/ensemble.html#forest (visited on 10/25/2023).

[2] *scikit-learn PCA User Guide*. URL: https://scikit-learn.org/stable/modules/decomposition.html#pca (visited on 10/25/2023).

[3] *Imbalanced-learn Combination of over- and under-sampling User Guide*. URL: https://scikit-learn.org/stable/modules/decomposition.html#pca (visited on 10/25/2023).

[4] *scikit-learn Logistic Regression API*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression (visited on 10/25/2023).

[5] *scikit-learn SVC User Guide*. URL: https://scikit-learn.org/stable/modules/svm.html#svc (visited on 10/25/2023).

[6] *scikit-learn Random Forest Classifier API*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn-ensemble-randomforestclassifier (visited on 10/25/2023).

[7] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[8] *scikit-learn Silhouette Coefficient User Guide*. URL: https://scikit-learn.org/stable/modules/clustering.html#silhouette-coefficient (visited on 10/25/2023).

[9] *Methods for variable-length time series*. URL: https://tslearn.readthedocs.io/en/stable/variablelength.html (visited on 10/25/2023).

[10] *tslearn DTW API*. URL: https://tslearn.readthedocs.io/en/stable/gen_modules/metrics/tslearn.metrics.dtw.html#tslearn.metrics.dtw (visited on 10/25/2023).