Eye-State Detection Project Report

Thodoris P. Papathanasiou *2011030058*, Nikoletta A. Tontou *2011030082*

Abstract—In this paper we investigate how to predict the eye state of a subject via measuring of the brainwaves. We use a corpus from [1]. We tested different machine learning algorithms as well as three neural network architectures on their accuracy predicting correctly the eye state of the subject. We present the results of the best classification algorithms which achieved an average error of 2.3%. We also achieved accuracy of up to 79% with the use of neural networks.

Index Terms—Classification, Eye-state detection, Machine Learning, Neural Networks, Pattern Recognition, Supervised Learning.

1 Introduction

The classification of the eye state by way of electroencephalography can benefit a plethora of applications. For example, it can be used to control input to devices and computers by handicapped persons or military personnel. Other novel approaches include controlling video games or tracking emotions. Thus an accurate and fast way of determining the eye state is of great importance to the subject.

In previous works adequate classification methods have been developed. [1] used the K-* algorithm to achieve a classification accuracy of 97.3%. [2] introduced artificial neural networks to the problem, though achieving poor results compared to [1]. Finally, several earlier works investigated the difference between eye states but did not suggest a robust and tested way of classification.

In this work, firstly we tried to reproduce the results of [1] using their dataset. After extensive experiments, using the Classification Learner App in MATLAB, and testing over 20 algorithms we were able to achieve a classification error less than 2.4% using the Fine KNN and Fine Gaussian SVM classifiers. Next, we created an Artificial neural network classifier using the Tensorflow library in Python. We tested 3 different network architectures and achieved a classification error of 21% using the Cascade Forward Neural Network.

The rest of the paper is structured as follows. In Section 2 we describe the dataset, giving more details as to how the measurements were received and what their values represent. In Section 3 we give the details of our implementations and our algorithms. The results of our experiments are then presented in Section 4. Finally in Section 5 we draw conclusions about the subject and suggest future improvements.

2 Materials and Methods

For the training and testing of our algorithms we used the data provided by [1]. For more information on how the experiments were conducted and the data acquired refer to [1]'s work.

The data-set we used consists of 14980 instances with 15 attributes each. The first 14 represent the electrode values and the 15th the eye state. The instances are also stored in chronological order as to retain their temporal dependencies. Around 55% of the the data corresponds to the open eye state while the rest 45% to the eye closed. As a result the corpus is considered balanced.

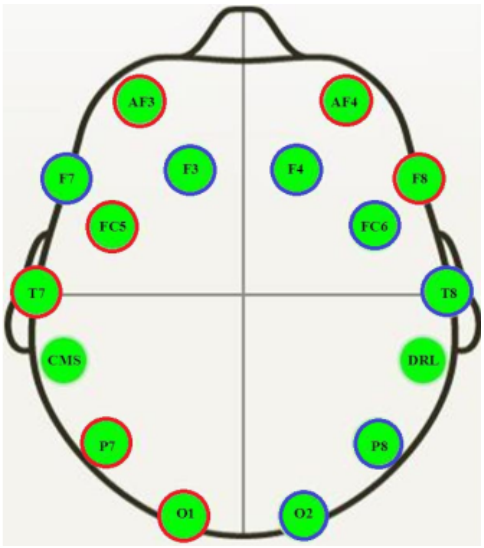

Fig. 1. Electrodes' placement and division into two groups: left (red color) & right (blue color)

The electrodes were placed as seen in Fig 1 and their value ranges are as seen in Fig 2. Based on these values they can be split into two groups, a left and a right. In the left group the minimum values decrease when we transition to the eye open state, while in the right group the maximum values increase in the same transition. With these observations in mind we computed different features for each group in the next section and tried to enhance our classification techniques.

3 Machine Learning Algorithms

3.1 General Testing

For the first part of our classification experiments we used the Matlab Classification Learner App (CLA) [3]. To start the experiments we decided on 10 fold cross validation for the entire 14980 instances on all CLA classifiers. There

Eye	closed			open		
State	min	mean	max	min	mean	max
AF3	4198	4305	4445	1030	4297	4504
F7	3905	4005	4138	3924	4013	7804
F3	4212	4265	4367	4197	4263	5762
FC5	4058	4121	4214	2453	4123	4250
T7	4309	4341	4435	2089	4341	4463
P7	4574	4618	4708	2768	4620	4756
O1	4026	4073	4167	3581	4071	4178
O2	4567	4616	4695	4567	4615	7264
P8	4147	4202	4287	4152	4200	4586
T8	4174	4233	4323	4152	4229	6674
FC6	4130	4204	4319	4100	4200	5170
F4	4225	4281	4368	4201	4277	7002
F8	4510	4610	4811	86	4601	4833
AF4	4246	4367	4552	1366	4356	4573

Fig. 2. Ranges And Means Of The Sensor Values For The Eye States

was no further parameter optimization on the classifiers.

After the first experiment round we took note of the classifiers that surpassed the 90% accuracy threshold and set to optimize them. The 3 best classifiers were: Fine Gaussian SVM, Fine KNN and Cubic SVM. Of these 3 Fine KNN was by far the fastest, so we focused optimization on it and then also tested in Fine Gaussian SVM.

3.2 KNN - SVM

Of all the traditional classifiers available through the CLA only two families showed good performance, K Nearest Neighbour and Support Vector Machine classifiers. From these two families the best results we achieved by Fine KNN and SVM with Fine Gaussian kernel. To further improve our results we conducted several tests altering the input data based on its observable differences.

To begin, we computed some more features from our data. These were the variance, the skewness and the kurtosis (v-s-k). Then we run the same tests and observed the results. To continue, we separated the electrodes into the groups described in Section 2 and we computed each groups variance, skewness and kurtosis. After this procedure we tested our classifiers on most possible combinations of features (e.g. electrodes, v-s-k). Finally, in an attempt to reduce the input needed to classify the eye state, we used only the electrodes with the biggest variance in their values and tested the classification on them and the v-s-k features.

The results of these experiments showed that with the addition of the extra features KNN benefited significantly while SVMs did not provide results capable of justify the extra computational complexity. Also the reduction of the features, while lowering the accuracy, could still produce acceptable results.

3.3 Neural Networks

Artificial neural networks resemble simplified versions of the biological nervous systems. They consist of a large number of highly interconnected computing elements called neurons and after they are adequately trained they can be used to solve untrained problems.

In our implementation we used tensorflow to implement and research 3 different neural network architectures: Feed Forward Neural Networks (FF NN), Long Short Term Memory Recurrent Neural Networks (LSTM RNN) and Cascade Forward Neural Networks (CF NN).

In all our tests we divided the dataset as follows. 64 samples were reserved for the testing set. Of the rest 80% was used for training and 20% for validation. We also used one hot encoding for our state labels meaning that the eye state was represented as a 1x2 vector ([0, 0]), taking values [0, 1] for the open state and [1, 0] for the closed state.

3.3.1 Feed Forward Neural Network (FF NN)

Common FF NNs consist of an input layer, 1 or more hidden layers and an output layer. Our implementation consists of an input layer of size 14, 2 hidden layers of size 256 each and an output layer of size 2. The output layer represents the predicted class of the input, in our case the eye state. The hidden layers consist of a matrix multiplication and addition between their input, the relative weights and biases, and a Rectified Linear Unit activation function. On the second hidden layer we also implement a dropout of factor 0.8 as to reduce training overfitting. Finally, the output layer resembles the hidden layers with the difference that its output is of size 2 and its activation function is a softmax_cross_entropy.

We used the Adam optimizer provided by tensorflow setting the learning rate to 0.001. The input data was split into batches of size 64. The FF NN architecture was trained on 200 epochs, reshuffling the data on each, for 10 sessions and its results were averaged to present the final accuracy. An overfit control method was implemented were if the FF NN validation loss did not decrease for 5 consecutive epochs the training was stopped. The results we very poor, nearly random, as we will see in the next section.

3.3.2 Long Short Term Memory Recurrent Neural Network (LSTM RNN)

RNNs make use of an internal memory (state) to model the network input and predict the output. Their main applications involve predicting time series output, given an uninterrupted data segment.

To test with our dataset we first disabled reshuffling of the input to keep its temporal dependencies. Continuing, we altered a given tensorflow lstm network, making it compatible with our data dimensions. Our LSTM RNN consisted of an input layer, a LSTM unit and an output layer. As with the previous architecture, we used the tensorflow Adam optimizer with a learning rate of 0.001 and the input data was split into batches of size 64. The training was conducted on 10 sessions of 100 epochs each. The same overfit control was applied. The results of the LSTM RNN were also very bad reaching up to 55% accuracy.

3.3.3 Cascade Forward Neural Network (CF NN)

The final network we implemented was the Cascade Forward Neural Network. In CF NNs the architecture resembles that of a FF NN with the difference that the input is cascaded directly to the output layer. Our CF NN was based on the previous implementation of the FF NN with

TABLE 1
Initial Classification Learner Results

Classifier	Accuracy
Logistic Regression	64.1%
Linear Discriminant	64.1%
Coarse Gaussian SVM	71.1%
Complex Tree	77.8%
Quad Discriminant	78.7%
Cubic SVM	94.5%
Fine KNN	97.3%
Fine Gaussian SVM	97.6%

the addition of a connection between the input layer and the output layer via appropriate weights and biases.

The same parameters were set and the same number of tests were run as with the FF NN. The results were much more encouraging and with further parameter optimization reached nearly up to 80% classification accuracy. We will explore the experimental results further in the next section.

4 Experiments and Results

4.1 KNN - SVM

All of our experiments used 10 fold cross-validation. The runtimes of the algorithms varied from a few minutes up to hours. The relevant hardware used to run the experiments was: 3rd generation Intel i5 processor, 8GB of RAM.

As it can be seen in Table 1. the initial results showed that only 3 classifiers surpassed 90% accuracy. The 2 best of them were Fine KNN with 97.3% accuracy and Fine Gaussian SVM with 97.6 accuracy. After our optimizations both of the previous 2 classifiers achieved a maximum classification accuracy of 97.7% shown in Table 2.

4.2 Neural Networks

For all the network architectures the same experiment was carried out. The parameters were set and then we performed a round of 10 training-testing sessions. The testing accuracy of each session was noted and an average value was computed at the end. On Table 3 we present the network configurations with the best results on the above experiment.

FF NN and LSTM RNN did quite poorly, achieving an average accuracy of 55% no matter the configuration. On the other hand CF NN did manage to achieve a classification accuracy of 72% with the standard configuration and even reached up to 79.4% by increasing the overfit control to 100 epochs before an early stop.

5 Conclusion and Future Work

In this work we implemented some some already established and some new methods for the classification of the eye state via electroencephalography. We saw that the traditional classifiers fare much better than artificial neural networks. Furthermore, in contrast with past works we observe that SVMs are a very good alternative to KNN or K-Star when no data processing is available. Finally we note

TABLE 2
Optimized Classification Learner Results

Fine Gaussian SVM results	
Features	Accuracy
Right data	82.5%
Left data	89.5%
All data (with PCA)	91.2%
Left data (with v-s-k)	92.5%
Right data (with v-s-k)	92.5%
All data	97.6%
All data (with v-s-k)	97.7%
Fine KNN results	
Features	Accuracy
Right data (with v-s-k of right data)	64.4%
Right data (with v)	67.6%
Right data (with s)	69.1%
Right data (with k)	71%
Right data (with v-s-k)	71.3%
Left data (with k)	75%
Right data (with v-s-k of left data)	75.6%
Left data (with s)	75.7%
Left data (with v-s-k of left data)	75.7%
Left data (with v)	76.6%
Left data (with v-s-k of right data)	76.7%
Left data (with v-s-k)	77.9%
Right data	79.4%
4 most varied electrodes (with v-s-k)	85.9%
Left data	88.6%
All data (with PCA)	89.8%
9 most varied electrodes (with v-s-k)	96%
10 most varied electrodes (with v-s-k)	96.9%
All data (with v-s-k of right data)	97.3%
All data (with k)	97.4%
All data (with v)	97.5%
All data (with s)	97.5%
All data	97.3%
All data (with v-s-k)	97.7%
All data (with v-s-k of left data)	97.7%

TABLE 3
Neural Networks results

FF NN results on 200 epochs and 0.001 learning rate.	
Network Configuration	Average Test Accuracy
NoES=5	0.577
NoES=5 & log(input)	0.524
NoES=8	0.555
LSTM RNN results on 100 epochs and 0.001 learning rate.	
Network Configuration	Average Test Accuracy
NoES=5	0.577
CF RNN results on 200 epochs and 0.001 learning rate.	
Network Configuration	Average Test Accuracy
NoES=5	0.719
NoES=100	0.794

NoES = Number of Early Stops i.e. overfit control

that even though the neural networks did not manage to perform good enough, small changes to their architecture had significant improvement to their results.

For future work, we think that a further investigation of feature reduction could possibly reduce the complexity of the traditional classifiers making them a viable option for applications in microelectronics, wearables etc. Finally we believe that with further research a viable neural network classifier can be implemented to this problem.

REFERENCES

[1] O. Rosler and D. Suendermann, *A First Step towards Eye State Prediction Using EEG*, Hewlett-Packard, Boblingen, Germany . Baden-Wuerttemberg Cooperative State University (DHBW), Stuttgart, Germany.

[2] B. Chambayil, R. Singla, and R. Jha, *EEG Eye Blink Classification Using Neural Network* in Proc. of the World Congress on Engineering, London, UK, 2010

[3] Classification Learner App reference: https://www.mathworks.com/products/statistics/classification-learner.html