Topics in Machine Learning

EEG Brain Signal Dataset for Emotion Prediction

PROJECT REPORT

I. INTRODUCTION

This dataset [1] is used for emotion prediction. For the experiment, 15 Chinese film clips were chosen, of positive, neutral and negative emotions. These clips were conditioned to criterion, such as easily understandable content, short duration to prevent fatigue in subjects, and depiction of a single desired target emotion.

Although a research [2] revealed other components or features related to human emotions in different frequency bands, this dataset only focuses on the Differential Entropy (DE) features. Each session in the experiment contains a 10-second hint prior to the beginning of the clip, followed by the 4-minute long film clip, and a 20-second rest and feedback time.

II. DATASET SPECIFICATION

The dataset used in this project contains a total of 310 attributes, each attribute is the reading of a particular sensor to detect human response to some stimuli. Based on these attributes, a particular instance in the dataset can be categorized into three emotions - positive, negative and neutral. There exist a total of 84,420 training samples, and 58,128 testing samples.

III. FEATURE EXTRACTION

The experimental analysis in this project summarizes the performance of the model over the original set of attributes, as well as extracted features. Moreover, a grid-search was also applied to examine if feature-scaling and/or normalization would improve the performance.

For feature extraction, different methods were used, namely principal component analysis (PCA), linear discriminant analysis (LDA), and quadratic discriminant analysis (QDA). Among the three mentioned, features extracted by PCA yielded the best overall performance, irrespective of the model used for further classification. Therefore, features have been extracted using PCA.

Model	***Train Score***	***Test Score***
Original	0.9720	0.6945
PCA	1.0000	**0.8106**
LDA	0.9762	0.7180
QDA	1.0000	0.7187

TABLE: **Performance of feature extraction methods**

IV. FEATURE SCALING

After feature extraction, with the help of a grid-search over performances of a given model over scaled and unscaled features, it was indicated that feature-scaling can be useful, depending on the classification model used. The table below shows the comparison of performances (for the ELM model) for unscaled features, as well as scaled features using StdScaler and MinMaxScaler from *sklearn*. Similar comparison has been done for each other model, and the best possible combination of scaling-methods and extraction-methods have been used for classification.

Scaling	*Train Score*	*Test Score*
No scaling	0.9702	0.6945
Standard Scaler	0.9463	**0.7184**
Min-Max Scaler	0.9450	0.4703

TABLE: **Performance of scaling methods on ELM**

V. CLASSIFICATION MODELS

A variety of classification models were used in experimenting for the best results. The performances of these are shown in the table below. Analysis was done for these models using scaled as well as unscaled, and original features or extracted features and recorded the best performance for the given model specification.

- **Extreme Learning Machine (ELM):**
  The fixed ELM model used in this project contains 3 layers, an input layer, a hidden layer, and an output layer. The number of input nodes (features) for this model is 180 which are obtained by applying PCA on the original 310 features. The hidden layer of ELM contains 1000 hidden neurons, which finally transform to the output, i.e. 3 classes. The architecture of the ELM-model is shown below.

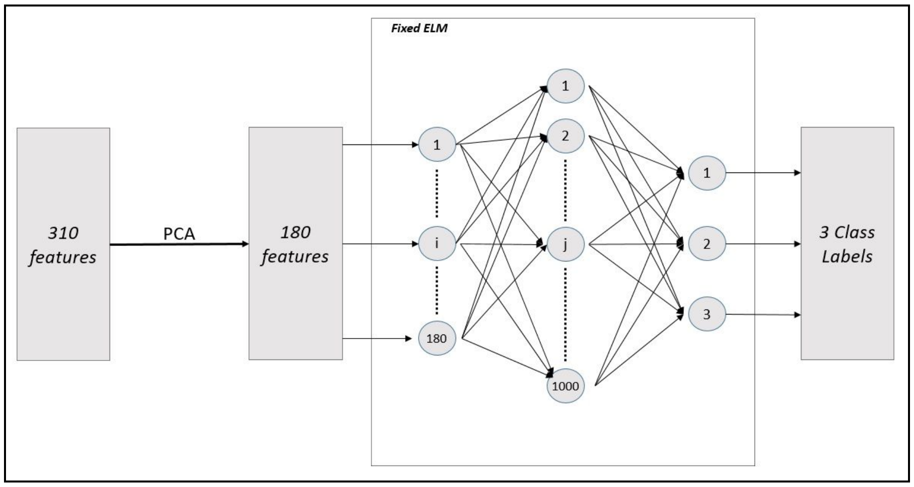
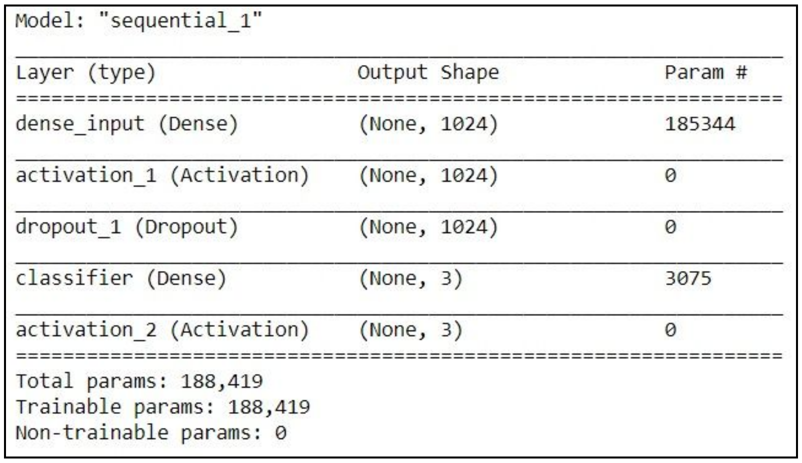


Figure: **ELM Architecture**

- **Random Forests (RF):**
  The Random Forest model used here is called from the *sklearn* library in *python*. There are two hyperparameters for random-forests - number of trees in the forest, and number of features selected for each tree in the forest. The number of trees in the RF model used in this project is 200, whereas all of the 180 features selected after PCA are used by each tree. Moreover, these 180 selected features are further scaled using the StdScaler method in *sklearn*, which are then used for classification.

- **Support Vector Machine (SVM):**
  The support vector machine model used is also from the *sklearn* library in *python* and the only hyperparameter tuning done for this model is on the value of C (penalty factor), which gives the best results at *C=0.2*. No feature scaling is done for the SVM classifier used. The overall method of SVM takes as input the set of 180 PCA-selected features, and transform them into the output class-labels, i.e. positive, negative or neutral.

- **Neural Network:**
  A simple neural network is used, for which the class-labels are encoded in *one-hot notation*. This neural network contains an input layer of 180 nodes (PCA selected features), a hidden *dense* layer of 1024 nodes, and finally the output layer with 3 nodes for classification. The network architecture is shown below.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_input (Dense)          (None, 1024)              185344
_________________________________________________________________
activation_1 (Activation)    (None, 1024)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
classifier (Dense)           (None, 3)                 3075
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0
=================================================================
Total params: 188,419
Trainable params: 188,419
Non-trainable params: 0
```

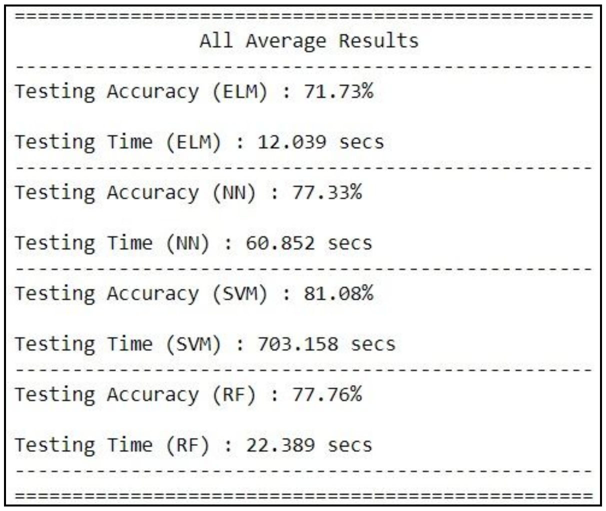
Figure: **Neural Network Architecture**

Classification Model	*Feature Extraction*	*Feature Scaling*	*Train Score*	*Test Score*
ELM	PCA (features = 180)	No-scaling	0.9437	0.7235
Random Forest	PCA (features = 180)	Std Scaler	1.0000	0.7852
SVM	PCA (features = 180)	No-scaling	1.0000	**0.8109**
Neural Network	PCA (features = 180)	No-scaling	1.0000	0.7786

TABLE: **Different models used (with accuracy scores)**

VI. RESULTS

The experiment performed used four different classification models for comparison. These include - Fixed ELM, Neural Network, SVM, and Random Forests. Analysis on these models, given the feature extraction and feature-scaling methods for each, yield average testing accuracies as mentioned below. For these models, the average time per iteration is also depicted by the graph below.



```
==================================================
               All Average Results
--------------------------------------------------
Testing Accuracy (ELM) : 71.73%

Testing Time (ELM) : 12.039 secs
--------------------------------------------------
Testing Accuracy (NN) : 77.33%

Testing Time (NN) : 60.852 secs
--------------------------------------------------
Testing Accuracy (SVM) : 81.08%

Testing Time (SVM) : 703.158 secs
--------------------------------------------------
Testing Accuracy (RF) : 77.76%

Testing Time (RF) : 22.389 secs
--------------------------------------------------
==================================================
```

Figure: **Average Results from all models used**

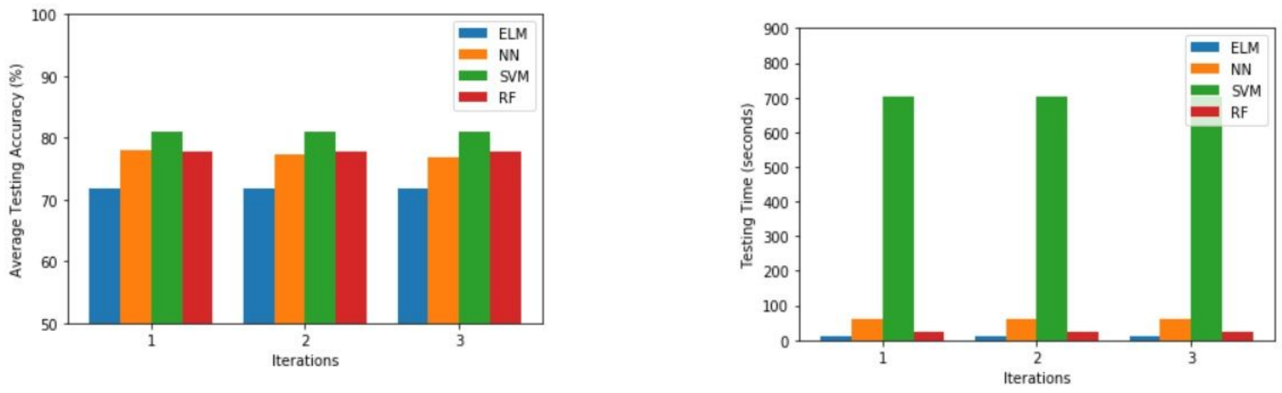


Figure: **Testing accuracy and testing time comparison for different models**

VII. CONCLUSION

The best testing accuracy was achieved by using a Support Vector Machine (SVM) model, with feature extraction using Principal Component Analysis (PCA) and without feature-scaling, i.e. an average testing accuracy of 81.08%, and an average testing time of 703 seconds (11 mins 43 seconds) per iteration.

Although, the time taken by the SVM-model at iteration is exceptionally high as compared to the other models (approximately 30 seconds per iteration), it yields the best overall performance based on classification accuracy on the test-set.