Machine Learning Algorithms
Project Report

Ayush Minocha (104410979)

March 11, 2015

1 Implementing Machine Learning Algorithms

The main part of the project is to implement two machine learning algorithms for time series data. The first one is k-Nearest Neighbours and the second one is a Neural Network. In the first part of the report I will go over the details about the datasets, the features that I have used and then the details about each of the algorithm that is implemented.

1.1 Datasets

We received two datasets, each of which has Systolic and Diastolic blood pressures for 39 patients, over 26 days. So each patient is represented by 26 pairs of blood pressures. For both the datasets, each of the patient is associated with one of the two classes, positive or negative. The codes that I have implemented can work in general for 'N' number of subjects, each subject represented by 'M' signals, and each signal containing 'P' points.

The biggest challenge was to extract meaningful features from these time series, because it is not possible to use the entire signal for classification. It may be used when the signal is short, but in the real world problems, the signals are extremely large, and it is computationally not feasible to use the entire signal for classification. So we extract a few meaningful features from the time series, which capture the essence of the data well.

1.2 Feature Extraction

This is the most important part in analysis of any data. As mentioned above it is difficult to use the entire time series signal and so it is important to extract some features that will represent the entire series well. The first approach I used was to extract statistical features from each signal. A list of features I tried is as follows:

- Mean
- Variance
- Minimum
- Maximum
- Root Mean Square
- Skewness
- Kurtosis

From my observations I concluded that the mean and root mean square values were nearly the same and hence gave equally good results for the old dataset. After trying various combinations of these features, I used the ones the worked best for each algorithm. I will provide the features that I used for each algorithm and the corresponding values obtained in the respective sections.

Along with these statistical features, I also tried using some features that would give a trend over the entire signal. So I used some coefficients of the Fourier transform and Wavelet transform of the data. But this did not give very good results for our dataset, and hence I did not pursue it further. On the same lines I also tried to use auto-correlation of the signal as one of the features, but statistical features tend to give better results in our case.

1.3 Neural Network

I started with implementing a single layer perceptron, with 3 inputs (2 input and 1 bias) and 1 output. I tested this with logical operations like AND, OR and EX-OR. As expected AND and OR worked properly as they were linearly separable, but EX-OR did not give good results as it is not linearly separable. Another important thing to note was that since there were only 3 parameters to learn, the three weights from the inputs to the outputs, they converge in a small number of iterations like 1000, but as the number of parameters (weights) to learn increase, it takes more number of iterations to converge to an optimum.

Since the datasets we have are not linearly separable, I implemented a general Multi Layer Perceptron, with 'N' number of layers and each layer having 'M' number of neurons. This general implementation gave me a flexibility to test different possible combinations of layers and neurons for the Neural Network. In general if we have large number of layers and large number of neurons in each layer then we can get a complex classifier that can classify very complicated data's but we should have enough amount of training data to support this, because large number of layers and large number of neurons mean very large number of parameters to learn, and thus less data will give an overfitted model. Since our dataset size is very small (i.e. only 40 subjects), we used only a 3 layers model, where the first layer is the input layer, middle layer is the hidden neurons layer and the outermost layer is the output layer.

An important thing that I noticed here was that, since we have 3 layers, we will have a decent number of parameters to learn, depending on the number of neurons in each layer, and so this requires a large number of iterations to train the neural network. If the weights of the neural network are not converged (i.e. the network has not been trained properly), then all the output values will be close to 0.5, that is all the activations are close to 0 and hence the sigmoid gives a value close to 0.5 . But when the network has been fully trained the output values converge close to 1 (0.995) and 0 (0.001). Using this observation I tried different number of iterations for the network. Since I had written the code in MATLAB, it took a long time to run for more then 10000 iterations. After writing the code in C (MEX), I was able to do 500,000 iterations in a few seconds, and got the network fully trained.

After trying a variety of features and their combinations as mentioned in the previous sub section, I found that the neural network gives the best results for a combination of two features namely Minimum and Maximum, for both the datasets. This gives a total of 4 features for each subject, min and max for both systolic and diastolic. Thus the neural network has an architecture of 4 input nodes, 4 hidden layer neurons, and 1 output neuron. I trained this neural network for 700,000 iterations. The results are given in the Table 1 for the old dataset and Table 2 for the new dataset. The ROC curves for the neural networks are given in the Figure 2 and the Figure 4. The ROC curve for the Unique Algorithm for the old data is given in Figure 3 and the ROC curve for the Unique Algorithm for the new data is given in Figure 5.

Model	Features	Accuracy	Precision	Recall	Sensitivity	Specificity	F-Measure
KNN, k = 5, L1 norm	Mean	0.6667	0.6667	0.6316	0.6316	0.7000	0.6486
KNN, k = 5, L2 norm	Mean	0.6410	0.6316	0.6316	0.6316	0.6500	0.6316
KNN, k = 1, DTW	Time Series	0.6410	0.6190	0.6842	0.6842	0.6000	0.6499
NN, [4 4 1]	Min, Max	0.7436	0.7647	0.6842	0.6842	0.8000	0.7222
NN + KNN + NN, k = 5, L2 norm	Min, Max, Mean, RMS	0.7949	0.7895	0.7895	0.7895	0.8000	0.7895

Table 1: Results for different models using different feature sets for the old dataset. KNN: K-Nearest Neighbour, NN: Neural Network, RMS: Root Mean Square, [4 4 1] : 4 input neurons, 4 hidden neurons and 1 output neuron, DTW: Dynamic Time Warping

Model	Features	Accuracy	Precision	Recall	Sensitivity	Specificity	F-Measure
KNN, k = 1, L1 norm	Mean, Variance	0.7179	0.7143	0.7500	0.7500	0.6842	0.7317
KNN, k = 5, L2 norm	Mean, Variance	0.6923	0.6818	0.7500	0.7500	0.6316	0.7142
KNN, k = 1, DTW	Time Series	0.4359	0.4444	0.4000	0.4000	0.4737	0.4211
NN, [4 4 1]	Min, Max	0.5385	0.5625	0.4500	0.4500	0.6316	0.5000
NN + KNN + NN, k = 1, L1 norm	Min, Max, Mean, Var	0.7949	0.7727	0.8500	0.8500	0.7368	0.8095

Table 2: Results for different models using different feature sets for the new dataset. KNN: K-Nearest Neighbour, NN: Neural Network, RMS: Root Mean Square, [4 4 1] : 4 input neurons, 4 hidden neurons and 1 output neuron, DTW: Dynamic Time Warping

1.4 k-Nearest Neighbours

For the k-Nearest Neighbour (KNN) , I implemented the kd-Tree algorithm to reduce the complexity of searching the k nearest neighbours. The key points in the k nearest neighbour algorithm are the distance metric used to find the nearest neighbours, and the parameter k which decides the number of neighbours to consider. On varying these two parameters I got different results. For the distance metric I used L1, L2 and L3 norms. Like neural networks, I tried different combinations of input features and found that for the old dataset, KNN gives best results for mean, at $k = 5$ and L1 norm, whereas for the new dataset the results are best for the feature combination of Mean and Variance, at $k = 1$ and L1 norm. The outputs values for these and other combinations is given in the Table 1 and Table 2 for comparison.

In KNN once I have the classes for the k nearest neighbours, I use the mode of these values to get the output class of the test point. I also return the regression values for each test data point, using a tricubic kernel, with λ equal to the sum of the maximum distance and one tenth of the maximum distance. These regression values are used in my unique algorithm described in the next section.

On doing some literature survey I found out that for time series analysis Dynamic Time Warping (DTW) can be used as a distance measure. DTW is an algorithm to measure the similarity or difference between two time series signal. The sequences are warped in the time dimension to measure the similarity. I did not implement this concept, but just tested it using a code on DTW from the internet. When using DTW, I used the entire time series as the input, and calculated the distance between the time series using the DTW algorithm. It gave good results for $k = 1$. The results are given in the Table 1 for the old dataset and Table 2 for the new dataset. The tables on the next page represent the confusion matrices for different models for data. Rows are actual classes, and columns are the predicted classes.

12	7
6	14

Table 3: Confusion Matrix for KNN, k = 5, L1 norm, Old data

13	6
4	16

Table 4: Confusion Matrix for NN, [4 4 1], Old data

15	4
4	16

Table 5: Confusion Matrix for Unique Algorithm, Old data

15	5
6	13

Table 6: Confusion Matrix for KNN, k = 1, L1 norm, New Dataset

9	11
7	12

Table 7: Confusion Matrix for NN, [4 4 1], New Dataset

17	3
5	14

Table 8: Confusion Matrix for Unique Algorithm, New Dataset

2 Unique Algorithm

For the unique algorithm part, I tried a lot of different combinations of features and models. I tried to combine the two algorithms, i.e. KNN and MLP, because they both have some strengths. My idea behind approaching the unique algorithm was to somehow try to get the best out of both, so basically something like use Neural Networks first and then use KNN on its outputs, hoping that KNN would correct the mistakes done by the neural network. Thinking on these lines I came up with a little complex architecture that gives better results, than these two algorithms used individually, on both the datasets.

The architecture is given graphically in the Figure 1. I start by training a neural network with 3 layers, with 'N' input neurons, 'N' hidden neurons and 1 output neuron. 'N' depends on the dimension of the input vector. I get best results for N=4, where the features are minimum and maximum values of the systolic and diastolic pressure for each subject. After training the neural network, I get the activations of the output neurons for the training data, and find the absolute error for each point. This acts as a new feature for my KNN algorithm. The KNN algorithm uses its normal features like mean and variance or mean and root mean square, along with the absolute error as a new feature obtained from the neural network. After building the kd-Tree, I find the outputs for the training data in KNN. The value of 'k' and the distance metric depends on the dataset being used. I have used the parameters, which gave best individual results for these datasets. Then I used the output of the previous neural network, and the output of the KNN algorithm as inputs for my new neural network, which forms the last step of the algorithm. So the last step has an input dimension of 2, where the 2 features are the outputs of the previous two models. The output of this neural network, is the final output.

For good results, I used the features and parameters that were giving the best results for the individual algorithms. So for the old dataset I used a neural network, with the features as minimum and maximum values for the two signals for each subject, with 700,000 iterations (so that the neural network is completely trained), followed by a KNN algorithm with the features as mean and root mean square for the two signals. For the new dataset, I used a neural network with same features as previous one, with 700,000 iterations, and followed by a KNN algorithm with features as mean and variance, and $k = 1$ and L1 norm.

The intuition behind this algorithm is that all the data that is misclassified by the neural network, will be in one plane and all the correctly classified data will be in another plane in the feature space. We will only have two major planes, because the neural network is completely trained, and all the weights have converged, so all the activations will be saturated at either 0 or 1 and hence the errors will also be either close to 0 or close to 1. Now the assumption here is that the neural network is a good complex classifier and can classify mostly all the points except a few outliers. Therefore most of the points in the misclassified plane will be outliers. Now if the new point is in the this plane, then it is most likely an outlier, and hence it is better to use other outliers to classify this point. So in the KNN algorithm, the k nearest neighbours for this point, will be mostly all outliers, and hence we are using all the outliers to predict an outlier, and hence the result of the KNN should improve. And the last step is applied in hopes to take the best results from both the models, i.e. the KNN algorithm and the neural network.

The ROC curves for the unique algorithm are given in the Figure 3. Also I have plotted these ROC curves along with their only neural network outputs, and it can be seen that the unique algorithm gives an improvement over the individual neural network as the ROC curve

for the unique algorithms are better in case of both the datasets.

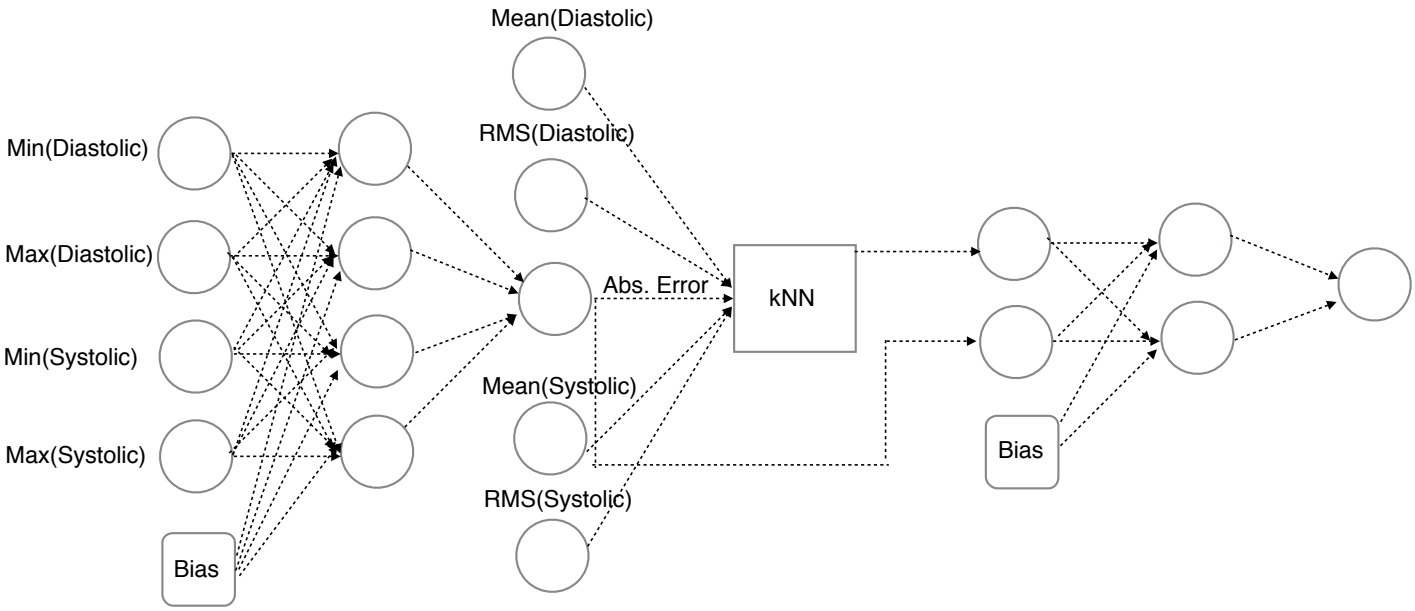

Figure 1: Architecture for unique algorithm

3 New Dataset

The new dataset consists of two similar signals for each subject. The signals correspond to the systolic and diastolic pressures over 26 days. The data is similar to the old dataset, but the classes are different, i.e. they represent different classes, and hence the model for the old dataset can not be used for the new dataset and a new model has to be trained for the new dataset. Also the trend in data is different, and hence same features might not work, for the new dataset.

On trying different combinations of features, I found that the new dataset gives best results in KNN for the features mean and variance of the two signals per subject, with $k = 1$ and L1 norm. Whereas neural net gives best results for the features minimum and maximum of the two signals per subject, and 4 input neurons, 4 hidden neurons and 1 output neuron. The value for these results are given in the Table 2. The features and parameters for the best results in the unique algorithm are as mentioned in the previous section.

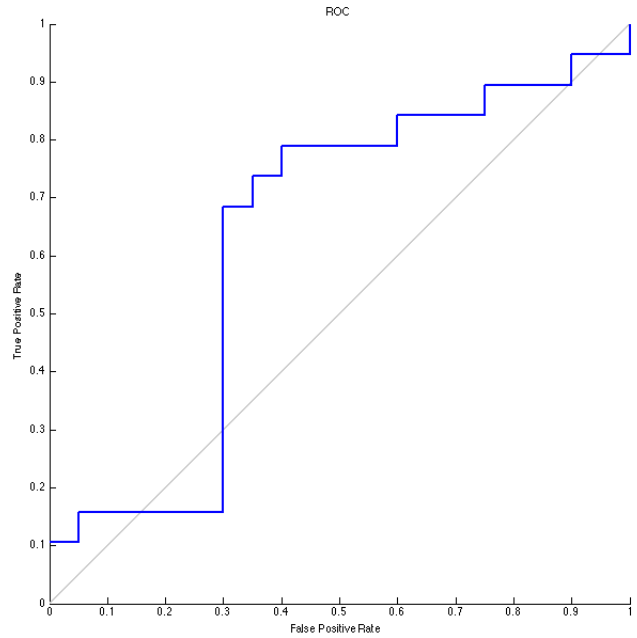

Figure 2: ROC for the Neural Network for old dataset

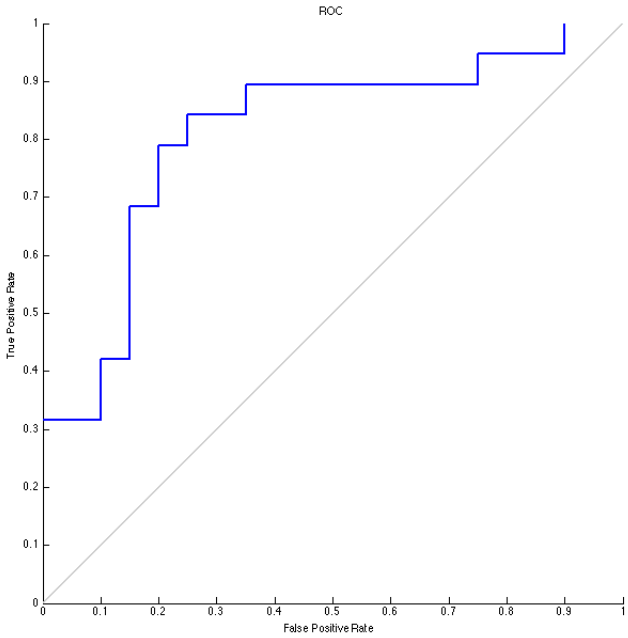

Figure 3: ROC for the Unique Algorithm for old dataset

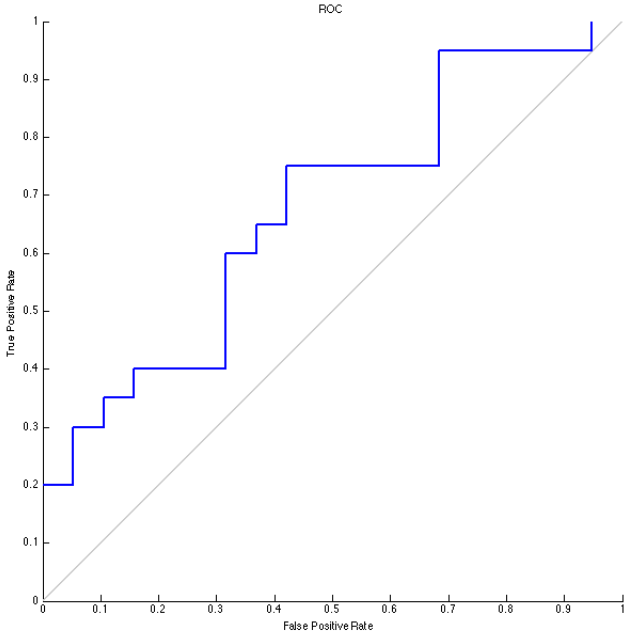

Figure 4: ROC for the Neural Network for new dataset

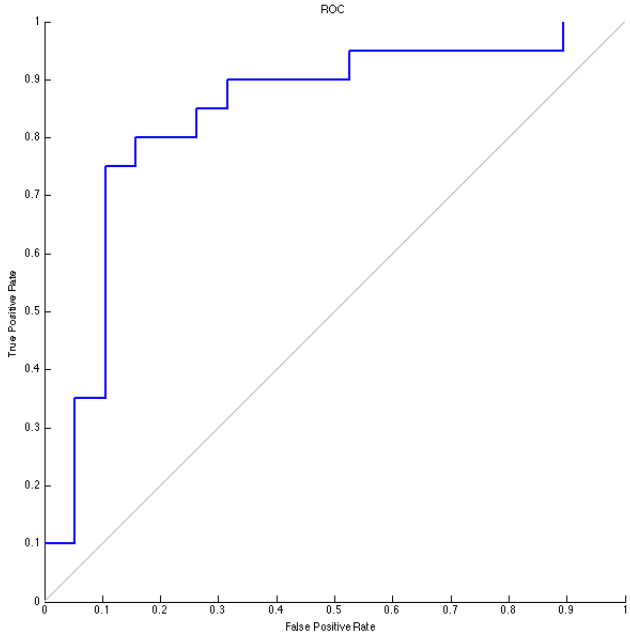

Figure 5: ROC for the Unique Algorithm for new dataset

4 Future Work

In this section I want to talk about some possible techniques that can be used for the analysis of time series data. I came across these techniques while doing some literature survey for the project.

The technique that I feel would give good results is a Convolutional Neural Network. This is a modification of the normal neural network. It has multiple layers, with the initial layers being the convolutional layers and the final few layers being the fully connected multi layer perceptrons. The idea is that the convolutional layers extract the features from the data and feed it to the MLP at the end, which performs the classification. So each hidden layer of the convolutional layer has some receptive field, and thus is not connected to all the input points, but the weight for it are shared. So each hidden neuron performs a convolution operation on the input, and thus learns some feature from it. As we go deeper in the convolution layers, they learn high level features, whereas the initial few layers learn the low level features. These features are then fed to the MLP which are the last few layers, and we get a final classification. This is a very powerful technique. I have not implemented this due to the time constraints, and the complexity of the algorithm. But it could be a good approach to analyse time series data and hence worth mentioning.