UCLA

UNIVERSITY OF CALIFORNIA, LOS ANGELES

Health Prediction Course Project

CLASS: CS260 MACHINE LEARNING
PROFESSOR: SARRAFZADEH, MAJD
TA: ALSHURAFA, NABIL

DONE BY **CHEYUN XIA**

UID **504422348**

Abstract - This is a course project designed to study Machine Learning technologies. The organization of this report goes as following: in part 1 I'll introduce the project and the objectives; in part 2 I'll review some related academic concepts; in part 3 I'll implement the system in steps and discuss the results using MATLAB; in part 4 I'll specify my unique contribution to this project and draw the conclusion for the project.

1 Introduction & Objectives

The project meant to implement two machine learning algorithms using time series data. The first one in kNN, and the second one is Neural Network. The dataset includes some patients. For each patient, there are two features, Diastolic and Systolic, and each pair features have been collected for 26 different timestamps. The objectives of this project are to fully understand the underpinning of these two algorithms and extract useful features from the time series data sets.

2 Literature review

k Nearest Neighbor – kNN method is quite straight forward: for input data, kNN looks up its k nearest neighbor training data and classify this input in the same class as the majority in its k nearest neighbor.

Neural Network – NN system is actually a weighted sum and activation feedback system. A single layer perceptron (pcn) is able to correctly classify linear separable dataset while a multi-layer perceptron (mlp) with hidden layer could be trained to classify nonlinear dataset.

Discrete Cosine Transform – DCT expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. It is powerful to be applied to series data when the data have consecutions in some sense.

3 Implementation & Results

3.1 Basic algorithms

knn – The kNN method is implemented in knn.m file. Since the project only involves 2 classes, I just counted class0 and class1 in k nearest neighbor.

pcn – The single layer perceptron is implemented in pcn.m file. Bias node is added at the first position of each input.

mlp –The multi-layer perceptron method is implemented in mlp.m file. There is one hidden layer in the model. Beta is included to better adjust the changing rate of weights. Momentum is also included for possibility of escape from local minimum to global minimum.

The parameters of the function interface are described in the readme.txt file with code. All of the three basic algorithm models have been tested in the first block of code.

3.2 Feature selection

By roughly plotting the data (Fig 1), one may find they're not easily linear separable without transformation. Inspired by this thought, I decided to use mlp instead of pcn.

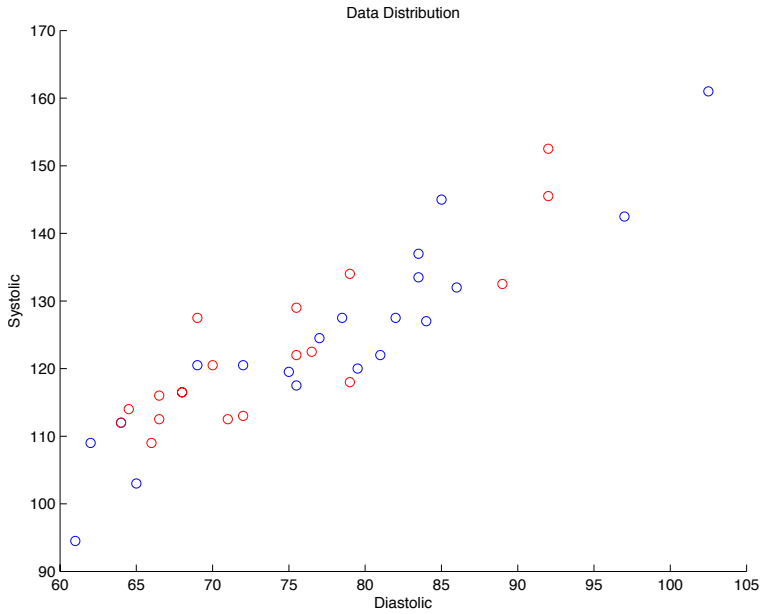

Figure 1. Data Distribution

3.2.1 Feature 1: mean and median

One natural thought is to directly use some statistic features like mean, median for each feature over time series. I tried this feature as inputs (other features are named inputs2, inputs3…). After using leave-one-out validation, the system outputs below:

Round	1	2	3	4	5
feature: mean					
Accuracy_knn	0.6154				
Accuracy_mlp	0.3590	0.3077	0.3333	0.3077	0.3590
feature: median					
Accuracy_knn	0.5128				
Accuracy_mlp	0.3077	0.3333	0.3333	0.3590	0.3333

Table 1. Feature1 result

It works fine for knn especially when using mean. Obviously this is not a good feature for mlp. One can tell several things from the table: the accuracy is no better than chance; mlp did worse than knn for this feature;
This result is not surprising because if we take mean or median directly, we lose the time series information. Instead we only use one absolute number as the feature. Inspired by this thought, we come to feature 2.

3.2.2 Feature 2: normalized input

Since the variation of data during time series could possibly tell information, we could try normalized input, which is (input-mean)/standard variance. This transforms the inputs to be mean=0, variance=1 dataset. Now we construct our inputs2 as a 52 dimensions vector for each person. The first 26 entries are normalized Diastolic and the second 26 entries are normalized Systolic data for 26 different timestamps. So now our inputs2 is a 39*52 matrix representing 39 patients and each one with 52 normalized features. After using leave-one-out validation, the system outputs below:

Round	1	2	3	4	5
feature: normalized input					
Accuracy_knn	0.3846				
Accuracy_mlp	0.4103	0.3846	0.4103	0.3846	0.4103

Table 2. Feature2 result

Obviously the result is better for mlp but much worse for knn. This can be expected since when we normalized the input data, the input ranges from -1 to 1, which fits better for our mlp model. But it's smaller for knn method. Notice that we discard the absolute information of data, only taking the variation of each one to count. And most importantly, only normalization is not enough to tell the whole variation of data in time series. This inspired me for the feature 3.

3.2.3 Feature 3: DCT of input

Using normalized input as feature will lose absolute information of data and is not enough to represent the variation and correlation in time series. And we found that there should be some correlation and consecution in time series for Diastolic and Systolic. That reminds me of Discrete Cosine Transform. DCT is famous to process series data; especially there is correlation in that series. In image and video domain, DCT is well applied since the pixels have temporal redundancy and special redundancy. Here we could treat the 26 pairs of Diastolic and Systolic has some sense of 'temporal redundancy'. So apply DCT to Diastolic and Systolic and normalized it we get our inputs3, which is a 39*52 matrix representing 39 patients and each one

with 52 normalized discrete cosine transformed features. After using leave-one-out validation, the system outputs below:

Round	1	2	3	4	5
feature: normalized input					
Accuracy_knn	0.3333				
Accuracy_mlp	0.3077	0.3333	0.3333	0.3077	0.2821
Accuracy_mlp'	0.6923	0.6667	0.6667	0.6923	0.7179

Table 3. Feature3 result

At first sight, the performance is not so good enough as expected. For knn, it is even the worst performance so far. For mlp, the accuracy stays around 0.20-0.40, which is also the lowest ever! However, since it is also stable, if we take complement of the result, the accuracy comes to 0.6-0.80, which is actually great. And we could still make optimization based on the result, which is talked in next chapter.

3.3 Input combination & cross validation

Based on discussion in 3.2, we decided to use feature 3 for mlp and feature 1 for knn. However we are not going to apply feature3 to system directly. By literature, Diastolic and Systolic should have some relationships; hence maybe there combination (maybe difference) is more informative than them. Inspired by this idea, I tried different combination of DN inputs (discrete cosine transformed normalized inputs). For here, I tried linear combination of [(2-p)*DN(Diastolic), p*DN(Systolic)] as final inputs, where p is a parameter we could change to see the performance. I tried p = 0.1:0.1:2 and use leave-one-out cross validation to see the performance, some of the best results is below:

Round	1	2	3	4	5
test different input combination					
Accuracy_knn	0.5385	0.5385	0.5128	0.5385	0.4872
	p=0.1	p=0.2	p=0.3	p=0.5	p=0.8
Accuracy_mlp'	0.7949	0.7949	0.8718	0.8205	0.8205
	p=1.4	p=1.6	p=1.7	p=1.8	p=2

Table 4. different input combination result

From the table, p makes difference so we could choose best p accordingly.

3.4 Calculate statistic parameters and plot ROC

Based on the results above, for we use feature1=mean for knn while use feature3 and p=1.7 to calculate accuracy, precision, etc. and plot ROC curves. In practice, choose the best p to calculate those parameters with running the data with other dataset.

Performance calculation					
accuracy	**precision**	**recall**	**sensitivity**	**specificity**	**F measure**
knn, feature1 = mean					
0.6154	0.6111	0.5789	0.5789	0.6500	0.5946
knn, feature3 = dctn, p=0.1					
0.5385	0.5455	0.3158	0.3158	0.7500	0.4000
mlp, feature3 = dctn and p=1.7					
0.8718	0.7917	1.0000	1.0000	0.7500	0.8837

Table 5. performance table

And the ROC curve of mlp is plotted as fig 2.

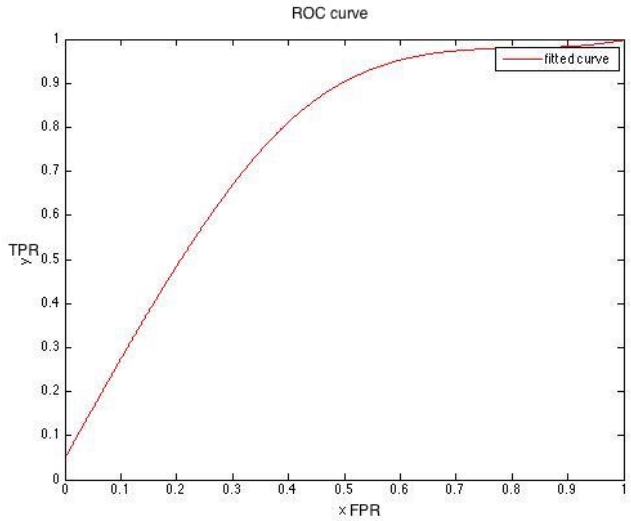

Figure 2. ROC curve of mlp

3.5 Performance with the final test dataset

Try the final test dataset with the system. Modify formatSpec0 and formatSpec1 to proper directory and modify other parameters accordingly. And feed the data to the system. And that yields the following performance(Table 6, fig 2):

Note that for knn, although feature1=mean works fine for previous data, it is worse for the final testing set. However, feature 3 with p=0.1 still yields good result.

Compared with the original dataset, our system performance at a similar level. This is due to the input comparison mechanism. We could select the best p due to the performance of each combination. Our system is stable in some sense.

Performance calculation					
accuracy	**precision**	**recall**	**sensitivity**	**specificity**	**F measure**
knn, feature1 = mean					
0.4103	0.4400	0.5500	0.5500	0.2632	0.4889
knn, p=0.1					
0.5385	0.5833	0.3500	0.3500	0.7368	0.4375
mlp, p=0.8					
0.8462	1.0000	0.7000	0.7000	0.7000	0.8235

Table 6. performance table for final test dataset

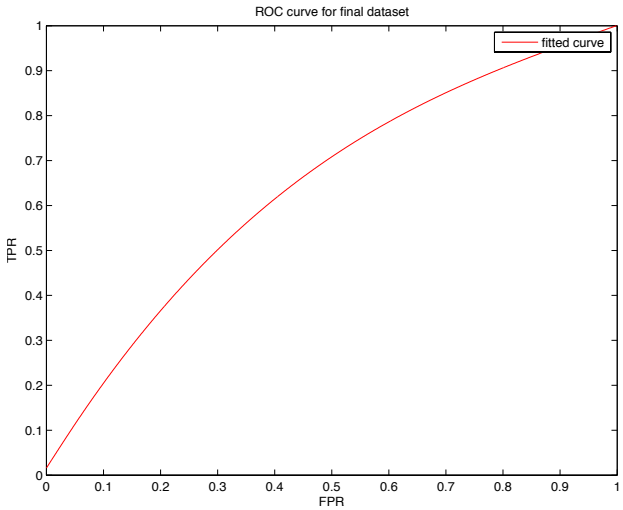

Figure 2. ROC curve of mlp for final dataset

4 Contribution & Conclusion

My unique contribution to the project two multifold: firstly, I tested some statistic features like mean, median, and normalized inputs and compared their performances. Based on this observation I put forward a new feature using Discrete Cosine Transformed and Normalized input as feature; secondly, based on the observation that Diastolic and Systolic have inner connection, I put forward a input combination mechanism to select best performed combination. I only implemented a simply linear combination in this project but as we can see, this implementation has already guaranteed great robustness for our system. One may extend to implement other complex combination of input at his will.

From this project we see knn's performance depends a lot on the dataset. We also see the importance of feature selection. What's more, if the data is hard to separate in time domain sometimes, and then we could turn to frequency domain to separate them. An improvement of this project could be the mixed feature of both time domain and frequency domain.