CISC5800 Final Project Report (2019 Spring)

Ming Chen Xuming Wang

Prof. Daniel D. Leeds

1. Introduction

One of the many great things about Machine Learning research is that due to its intrinsic general nature, its spectrum of possible applications is comprehensive. And one of the possible directions in which we can push forward the Machine Learning Detection in Medicine. In Medicare domain, with the help of new methods, doctors could efficiently diagnose the kind of disease of a patients.

In this project, we are going to build a model that can help doctors find the condition of patients' heart and ultimately save human lives. We want to create a system that can identify cardiac Single Proton Emission Computed Tomography (SPECT) images, in which each patient would be classified into two categories: normal and abnormal. We'll have to explore Feed-Forward Neural Networks. In the model building, we use Numpy to build it from scratch and we also build our second classifier with the help of Pytorch. Both classifiers are reached to XXX %. Finally, we test several optimization solutions to pick the best way to solve that problem.

2. Dataset

2.1 Dataset Description

SPECTF is a useful data set for testing ML algorithms; it has 267 instances that are described by 45 attributes. The database of 267 SPECT image sets (patients) was processed to extract features that summarize the original SPECT images. As a result, 44 continuous feature pattern was created for each patient. The CLIP3 algorithm made rules that were 77.0% accurate (as compared with cardiologists' diagnoses).

2.2 Data Pre-processing

First, we need to separate our training data into a training data and test data. Fortunately the data set has already been divided to training data ("SPECTF.train" 80 instances) and testing data ("SPECTF.test" 187 instances).

Obviously the train data is relatively small, so my are considering using GANs and prepended neural net.

This dataset contains no missing value and we have same number of class 1 and class 0 which indicates that the data is totally balanced.

2.3 Feature Reduction

As mentioned above the quality of the data is quite good. But we still want to try some dimension reduction method in our project. This time we apply some correlation analysis and PCA. First of all we do correlation analysis between 44 features in train data

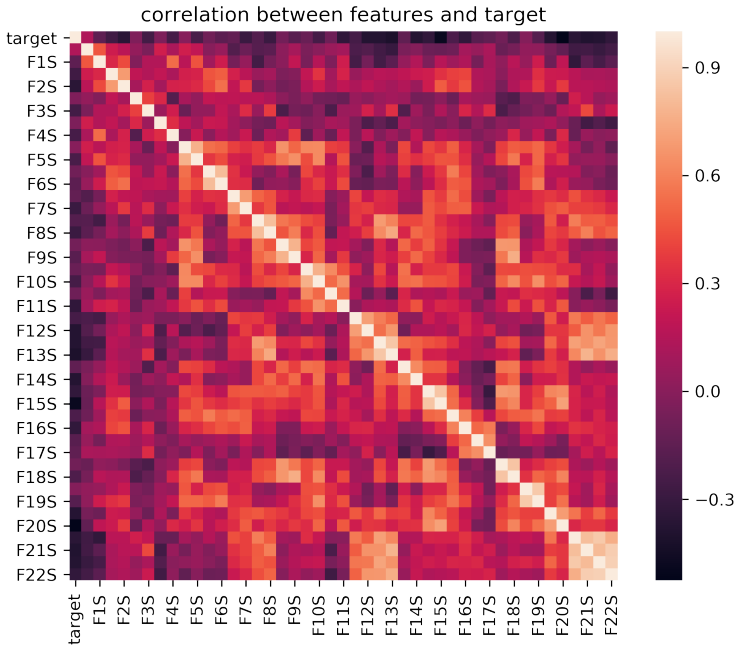


Figure 1: Correlation between original features

Then we use PCA to generate new components

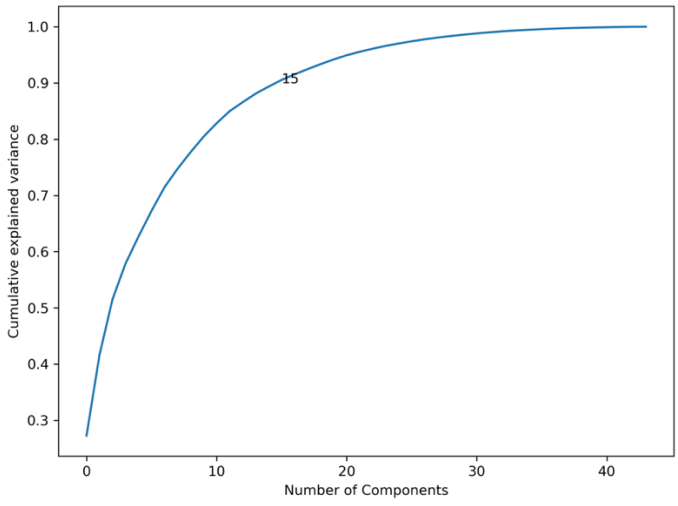


Figure 2: PCA

According to the fig above we can see that 15 components can give us over 90 percent of information. After the PCA we reduce correlation almost to 0 among different components.

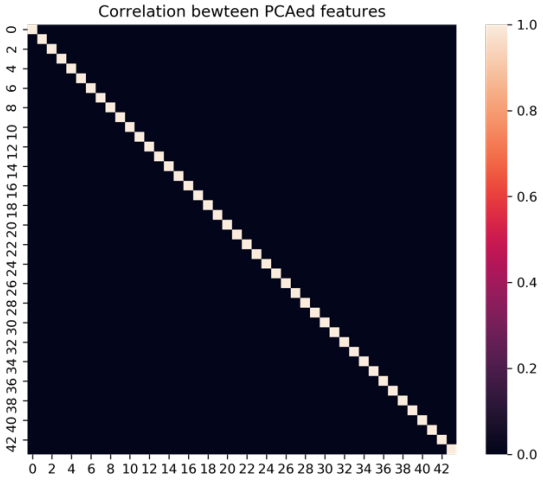


Figure 3: Correlation between features after PCA

3. Learning Method

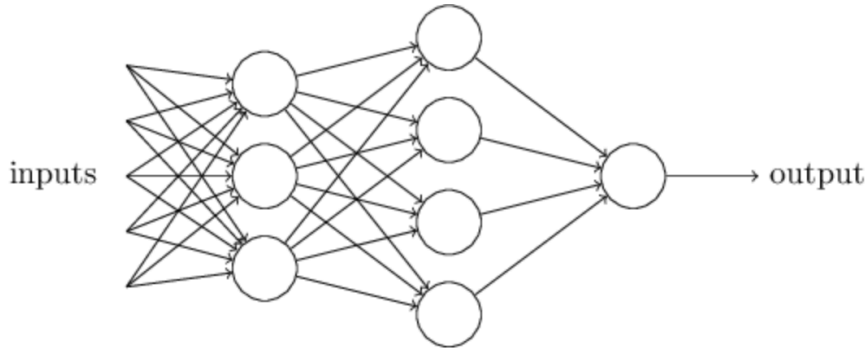
Now it time to build our model. We would use the neural networks we built to solve the classification problems. We are going to write a neural model from scratch and use training data to train the model.

Besides, we have prepared a neural network built by Pytorch for comparison. Both models have the same architecture and learning rate. Although our model is simpler, the results show that the two models both achieve xx% accuracy on the test set.

Another problem is that our handwritten model takes double amount time compared to the Pytorch Neural Net. So the speed optimization would also be a interesting part to explore.

3.1 Neural Network from scratch

First we go through some basic concept and formula. In this project we only write the traditional multi-layer perceptron. The following is a typical neural network structure.



The basic mechanism behind the neural network is feedforward and back propagation. Each neuron, we still pick sigmoid function as our activation function. Therefore, our feed forward function is shown as follow:

$$r_k^m = \text{sigmoid}(\Sigma_j w_{k,j}^m r_j^{m-1} + b_k^m)$$

The machine learning part of this model is to update the weight of every pair of neuron. The updating rules called back propagation, rules as follow:

For top layer:

$$\Delta w_{k,j}^m = \varepsilon \delta_k^m r_j^{m-1}$$
$$\delta_k^m = (1 - r_k^m)(y - r_k^m) r_k^m$$

For non-top layer:

$$\Delta w_{k,j}^m = \varepsilon (1 - r_k^m)\left(\Sigma_n w_{n,k}^{m+1} \delta_n^{m+1}\right) r_k^m r_j^{m-1}$$
$$\delta_k^m = (1 - r_k^m)\left(\Sigma_n w_{n,k}^m \delta_n^{m+1}\right) r_k^m r_j^{m-1}$$

We set our layers and nodes just as the fig above. In the first layer we have 3 nodes and we have 4 nodes in the second layer. When we started to design our code structure we just found that there are so many loop in the model and every time we do iteration we need a lot variable to store data. So we decide to make each node a class. The class concept was inspired by biological collections of features (attributes) and abilities (methods).

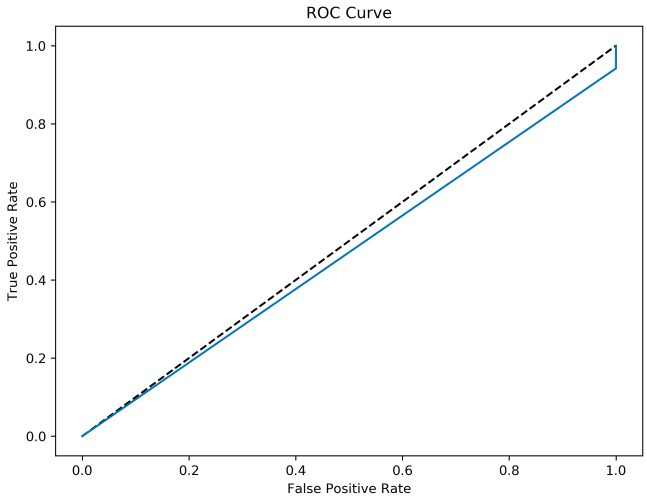
We strictly follow the formula above and below is the result of our model:

```
Runing time :  0:00:00.773471
Acc: 0.8663101604278075
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        15
           1       0.92      0.94      0.93       172

   micro avg       0.87      0.87      0.87       187
   macro avg       0.46      0.47      0.46       187
weighted avg       0.84      0.87      0.85       187

[[  0  15]
 [ 10 162]]
AUC: 0.47093023255813954
```



3.2 Neural Network with Pytorch

We use Pytorch package as our second classifier. With Pytorch we can easily adjust our parameters such as weights, evaluate function and number of nodes and layers.

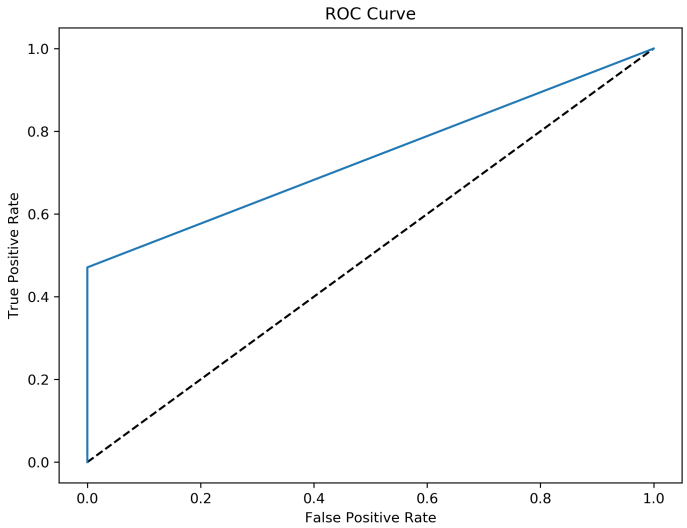
4. Optimization

Machine Learning is an iteration process to renew the parameters of the model based on a convex loss function. The goal is to find the parameter values that minimize the cost function. The process starts by guessing some initial parameter values. Then we iteratively change the parameter values in such a way to reduce the cost function. Hopefully, the process ends with a minimum.

4.1 Accuracy Optimization

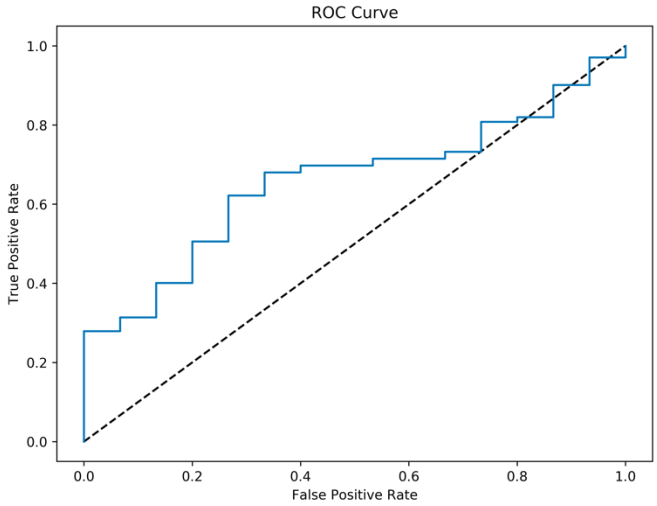
There is a lot space to improve our neural network. For example, our model constitutes of 3 layers, and each layer owns fixed number of nodes. We assume that the number of nodes would influence the result.

We want to find the best number of nodes in each layer. So we did several for loops in our hand written model. And we got the below roc curve:



We finally set 22 nodes in first layer and 5 nodes in second layer. As we can see that the adjustment of nodes number brings us a significant improvement.

Dropout is a regularization technique. During training, dropout randomly sets units in the hidden layer h to zero with probability p drop (dropping different groups each minibatch) and then multiplies h by a constant γ. For Pytorch we can test this parameter from 0.1 to 0.9 and our prediction achieve best when dropout equals to 0.5, below fig is the result.



4.2 Converge Speed optimization

The changing steps are based on the partial derivative of the parameters themselves and make sure that the lost function downhill to a minimum. By gradient descent, loss function

can get coverage in an optimum. But in real life, optimization problems are incredibly complicated and continue in using the traditional update rule cannot permit a reasonable time completed.

Thus, to implement the model better, we introduce others update rules with significant computation efficiency for parameters learning. Adam Optimization1 uses a more sophisticated update rule with two additional steps:

First, Adam uses a trick called momentum by keeping track of m, a rolling average of the gradients:

$$\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} J_{\text{minibatch}}(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \mathbf{m}$$

Where β1 is a hyperparameter between 0 and 1 (often set to 0.9).

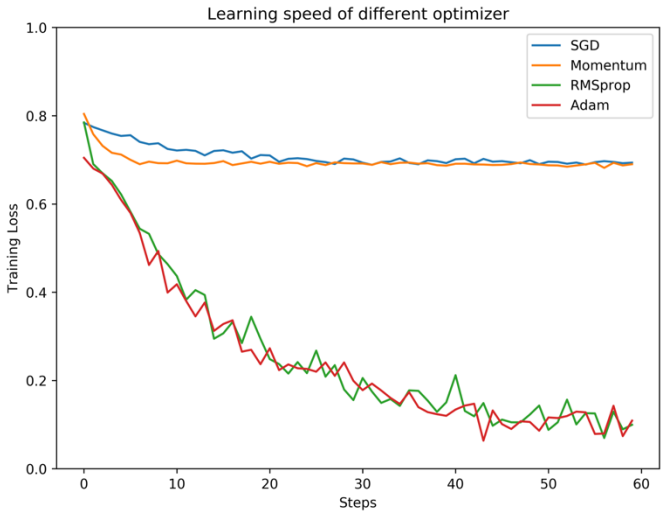
Adam also uses adaptive learning rates by keeping track of v, a rolling average of the magnitudes of the gradients:

$$\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \nabla_{\boldsymbol{\theta}} J_{\text{minibatch}}(\boldsymbol{\theta})$$
$$\mathbf{v} \leftarrow \beta_2 \mathbf{v} + (1 - \beta_2)(\nabla_{\boldsymbol{\theta}} J_{\text{minibatch}}(\boldsymbol{\theta}) \odot \nabla_{\boldsymbol{\theta}} J_{\text{minibatch}}(\boldsymbol{\theta}))$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \odot \mathbf{m} / \sqrt{\mathbf{v}}$$

Where ⊙ and / denote elementwise multiplication and division (so z⊙z is elementwise squaring)

and β2 is a hyperparameter between 0 and 1 (often set to 0.99).

We are curious about how we can adjust our gradient changing rules to get a better converge speed. Below is our result for this training dataset.



5.Conclusion

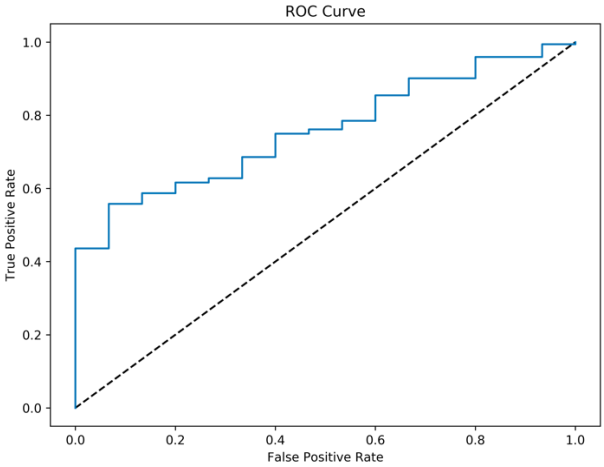
Combining all the parameter we choose, we fit them to the Neural Network built by Pytorch , and the final result is much better than the model we built at first.

```
0:00:05.138208
Acc: 0.49732620320855614
              precision    recall  f1-score   support

           0       0.13      0.93      0.23        15
           1       0.99      0.46      0.63       172

   micro avg       0.50      0.50      0.50       187
   macro avg       0.56      0.70      0.43       187
weighted avg       0.92      0.50      0.60       187

[[14  1]
 [93 79]]
AUC: 0.6963178294573643
```



Although we use two same model to do the prediction but as we test there are some obvious difference. With the help of Pytorch package we get chance to explore more parameters' influence to the performance such as drop rate.

It is common knowledge that the more data an ML algorithm has access to, the more effective it can be. Even when the data is of lower quality, algorithms can actually perform better, as long as useful data can be extracted by the model from the original data set.

Unfortunately, we only got 80 instances in our train data while we focus more on parameter choosing, we would do more analysis about the feature selecting and solve the problem from small dataset.

6.Citation:

1. https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

   Stochastic Gradient Descent with momentum

2. http://web.stanford.edu/class/cs224n/assignments/a3.pdf

Dropout and Adam

3. http://cs231n.stanford.edu/reports/2017/pdfs/300.pdf
   The Effectiveness of Data Augmentation in Image Classification using Deep Learning