Build up Perceptron in Diabete data

Melinda Dong
University of Adelaide
North Terrance, SA 5000
nanyu.melinda@gmail.com

Abstract

In this study, perceptron model was implemented from scratch to predict weather sample individuals have diabetes or not by 8 physical health indicators. By adjusting the parameters, hyperparameters and using different loss functions. The best model in this case, SoftMax perceptron, give the best training accuracy as 0.786 and the best test accuracy as 0.765.

1. Introduction

Here are some more clear information of the data source and the brief history and basic conception of perceptron.

1.1. Data source

The original data are from https://www.csie.ntu.edu.tw/cjlin/libsvmtools/datasets/binary.html.

There are 8 independent variables to predict weather a sample has diabetes or not. [1]Those 8 medical predictors are:

- Number of times pregnant
- Plasma glucose concentration 2 hours in an oral glucose tolerance test
- Diastolic blood pressure (mm Hg)
- Triceps skin fold thickness (mm)
- 2-Hour serum insulin (mu U/ml)
- Body mass index $(weightinkg/(heightinm)^2)$
- Diabetes pedigree function
- Age (years)

The outcome are 2 classes, 268 have diabetes, 500 samples don't have diabetes.

1.2. Background of perceptron

Perceptron is a machine learning algorithm; it has a long history (since 1960s), and it is fundamental of neural network. It is a supervised machine learning model, and it is a simple binary classifier.

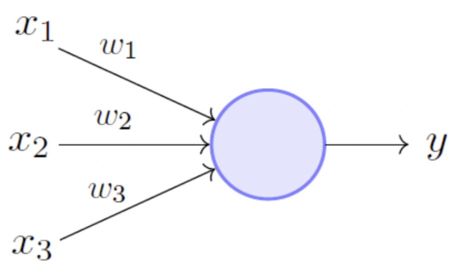

Figure 1. Perceptron Model (Minsky-Papert in 1969) [2]

2. Method description

The first model is called the primitive model. It shows the very first idea of how perceptron works. The Second model is called SoftMax model, it used SoftMax method to predict the probability of one sample have diabetes or not. And used cross entropy to calculate the loss. The data set as the first 500 data as training and the rest 268 data as test. The training - testing ratio is around 65:35. For the primitive model, the results is set as -1 and 1, -1 means have diabetes, and 1 means don't have diabetes. As for the SoftMax model, the result is set as 0 and 1, 0 means have diabetes, and 1 means don't have diabetes for coding convenience.

2.1. Primitive model

The logic used in primitive model is: first initialize the weights and bias as 0. Iterate each data line by line. For each data, check whether the predicted result is the same as real result, if they are the same, continue iteration, if not, update the weight and bias.

Our predicted results, $\widehat{y}$, is predicted by this formula:

$$\widehat{y} =< w, x_i > +b$$

I check whether $y_i * \widehat{y}$ less or equal to 0 to tell if the prediction is right or wrong. If $y_i * \widehat{y} < 0$, means the sign of real one and predicted one are different, which indicates a wrong prediction.When the prediction is wrong, I update the weight by adding $y_i * x_i$ and update the bias as adding y_i.

2.1.1 perceptron loss function

The primitive model is using the loss function as follows:

$$L(y, X, W) = max(0, -y < w, x >)$$

And equivalent to gradient descent with batch size 1. The model check one sample at a time, if the prediction is correct, which means $< w, x >$ greater than 0, hence, $- < w, x >$ less than 0 and the maximum between 0 and $- < w, x >$ is 0, which means 0 loss. If the model predicts wrong, the maximum between 0 and $- < w, x >$ is $- < w, x >$, which means loss is $- < w, x >$.

2.2. SoftMax model

The second model used SoftMax function to do classification. And used cross entropy as loss function to update parameters. And this model gives a better performance.

2.2.1 SoftMax function

The weight was initialized as normal distributed random number with mean 0 and standard deviation 0.01 and the bia was initialized as 0. The model still using the basic formula $X.W + b$ to get the output, for every class (in this case, 0 and 1), through this formula will get an output, and then use SoftMax function to convert these outputs into a probability by this formula:

$$\widehat{y_i} = \frac{exp(o_i)}{\sum_k \exp(o_k)}$$

where
o_i is the output of the correct class
o_k is all the possible outputs.

This reason why we can see it as a probability is after this transformation, the new output has 2 qualities, 1 is non-negative, another is the summation is 1, which satisfy the nature of probability. [3]

2.2.2 Cross entropy loss function

Cross entropy is a function that used to measure the differences between different probabilities. [4] The general cross entropy formula looks like this:

$$H_{(p,q)} = \sum_i -p_i \cdot log(q_i)$$

In this case, since the there is only one class is correct, so the correct class has the probability of 1 and the rest has the probability of 0, then the loss function can be simplified as:

$$L_{(\mathbf{y},\hat{\mathbf{y}})} = -\sum_i y_i \cdot log(\widehat{y_i}) = -log(\widehat{y_y})$$

where
$\mathbf{y}$ is the real correct class(vector).
$\hat{\mathbf{y}}$ is the predicted probability of the classes(vector).
$\widehat{y_y}$ is the predicted probability of the correct class.

2.2.3 Gradient descent

Gradient descent is an optimization algorithm which is commonly used to train machine learning models and neural networks. [5]

After getting the loss function, I use learning rate as 1 to update the weight using formula:

$$W_t = W_{(t-1)} - \eta \cdot \frac{\partial L}{\partial W_{(t-1)}}$$

where
η is learning rate
L is loss function

At the baseline SoftMax model, 200 epochs were run to update the model and try to get close to the optimised solution.

3. Method implementation

Here is the git hub link: https://github.com/MelindaDong/Perceptron_Melinda.

Part 1 is primitive model.

Part 2 is SoftMax model.

4. Experiments and analysis

Several experiments were done, which includes using different weights and bias initialization; using different loss function; running different epochs; and using different learning rate and compare the result with Scikit-learn build-in perceptron model.

4.1. Weights and bias initialization

Different ways of initializing parameters were implemented on the primitive model. At first, w and b were initialized as 0, before training, these 0-parameters setting gives a 0 accuracy on both training data and testing data. After one round training, it gives a training accuracy as 0.658 and test accuracy as 0.642.

Then the w and b were both initialized as random number, normal distributed, with mean as 0 and standard deviation as 0.01. Before training, these random parameters setting gives training accuracy as 0.368 and test accuracy as 0.354. After one round training, it gives a training accuracy as 0.682 and test accuracy as 0.664.

According to this experiment, I found out the setting weight and bias as random numbers (normal distributed) can give a better result than initialise them as 0. This randomness gives a model better prediction even before start training. [6] Hence the model used random initialization update its parameters on a better base.

4.2. Change with epochs

For the primitive model, as every training batch is one, one round of model training is checking data for 500 times (training data size), so the model converges very quickly. After 2 rounds training, the training accuracy converged to 0.656. The same performance on testing data, after 2 rounds, the test accuracy converged to 0.642.

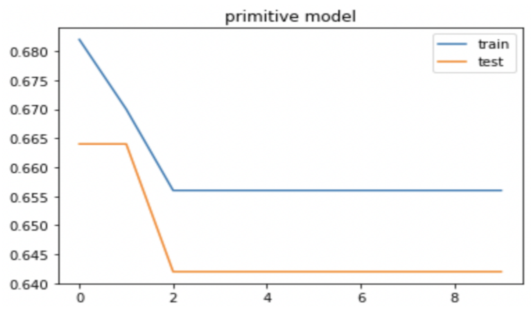

Figure 2. Accuracy plot of primitive model

As for SoftMax model, 200 epochs were run to see the change of accuracy. Both train and test accuracy of the baseline model (with learning rate as 1) increased very quickly the first 25 epochs, after 25 epochs, it starts converging with a very slow rate. Train accuracy converge to 0.786 and test accuracy converge to 0.765.

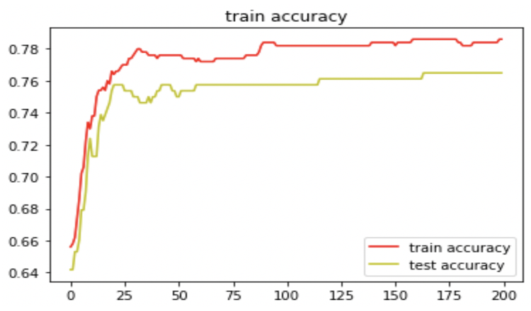

Figure 3. Accuracy of baseline softmax model; learning rate 1

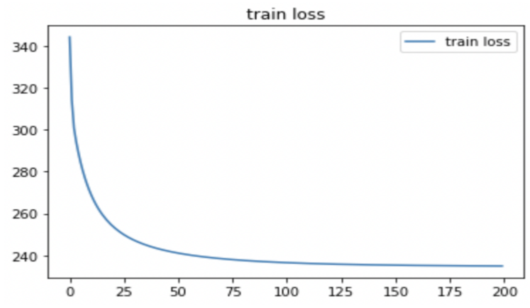

Figure 4. Loss of baseline softmax model; learning rate 1

4.3. Learning rate adjustment

Different learning rate (0.001, 0.01, 0.1, 1, 10) were tested on SoftMax model. With learning rate 0.001 and 0.01, the learning rate is too slow, the model did not learn, the training accuracy keep parallel to x axis as 0.656 and the test accuracy keep parallel to x axis as 0.642.

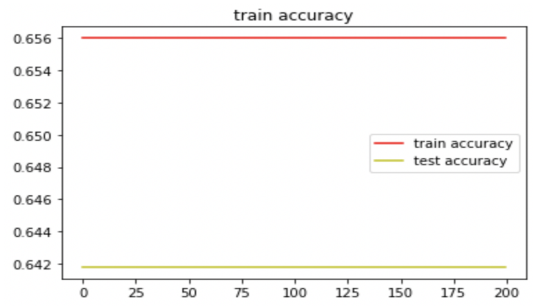

Figure 5. change with learning rate 0.001

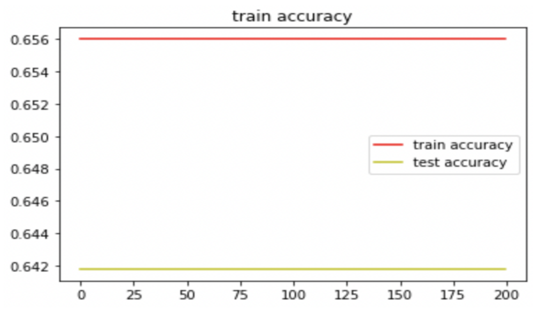

Figure 6. change with learning rate 0.01

With learning rate as 0.1, the model's accuracy did not change the first 25 epochs, after 25 epochs, it improves very steady. With learning rate as 1(see Figure 3), the accuracy increased very quickly the first 25 epochs, after 25 epochs, it starts converging with a very slow rate.

With learning rate as 10, the model is very unstable, Accuracy keeps fluctuatingbut its amplitude of fluctuating gradually decreases with the increase of epoch, and finally

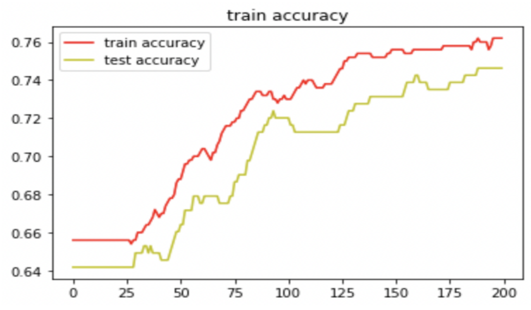

Figure 7. change with learning rate 0.1

converges after 75 epoch. But even after convergence, it still maintains a small amplitude.

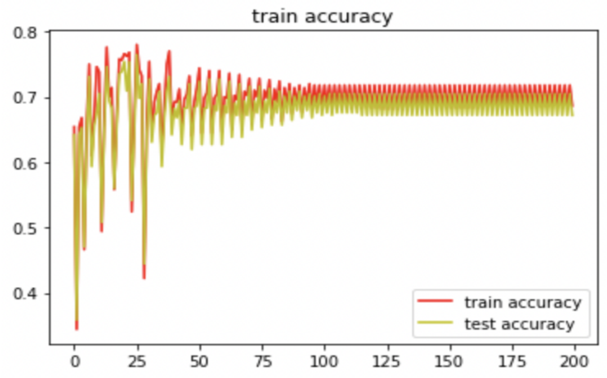

Figure 8. change with learning rate 10

In this case, 0.001 and 0.01 learning rate are too small, 10 learning rate is too big. Both 0.1 and 1 is good learning rate for this case, and with 0.1 learning rate, the model learns more stable.

4.4. Final models compare

Scikit-learn build-in perceptron model was implemented as a benchmark, it gives a train accuracy as 0.7 and test accuracy as 0.687. The best train and test accuracy of primitive model are: 0.682 and 0.664; The best train and test accuracy of SoftMax model are: 0.786 and 0.765.

The primitive model gives a slightly worse performance than benchmark and the SoftMax model gives a slightly higher performance than the benchmark.

SoftMax model can perform better, Because of its large model capacity. In primitive model, the size of weight is 8 and the size of bias is 1. But in SoftMax model, the size of weight is 16 (8 x 2) and the size of bias is 2, which means SoftMax model have more parameters to adjust to learn data better.

5. Reflection

In this project, 2 different implementations of perceptron were built up from scratch. One is primitive model another is SoftMax model. Few parameters and hyper parameters were tested to approach the best performance. The best performance was given by SoftMax model with initialize parameters as random numbers and learning rate as 1. The final test accuracy is 0.765, which is better than the Scikit-learn perceptron model (test accuracy as 0.687)

There is one strange phenomenon in the primitive model, both train and test accuracy are dropping at the first 2 rounds. This might because the random initialization is very good or might have other reasons. This requires further research by other scholars.

Single layer perceptron is indeed a linear classifier, it can only work perfectly on linear separatable data. The model compacity is not enough for complicated data.

Due to the limited time, I would like to try more different loss functions and try to add some optimizations, such as Stochastic gradient descent. This are the experiment I would like to test in future.

References

[1] https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database 1

[2] https://towardsdatascience.com/what-is-a-perceptron-210a50190c3b 1

[3] https://ecwuuuuu.com/post/sigmoid-softmax-binary-class 2

[4] https://machinelearningmastery.com/cross-entropy-for-machine-learning 2

[5] https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21 2

[6] https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78 3