Assignment 1

Nguyen Phuc Thai
a1846505

1. Introduction

This report aims to use Perceptron to predict whether a person have diabetes. The Perceptron algorithm applied in this report is the one outlined in the lecture slide and will be described in section 2 [2], and the sample dataset is obtained from CSIE [1], for this report, the scaled version of the diabetes dataset is used (all predictors are scaled to values between 1 and -1, and the target will be either 1 (diab or -1). The data consists of 768 observations and 8 predictors variable describing biological factors of people with and without diabetes, and the dataset contains more observations from class 1 than those from class -1. All model building process and analysis are done using Python and the associated numpy package.

2. Methodology

2.1. Baseline algorithm

From the sample dataset, 80% of the observations will be used for model training and validation, and the remaining 20% will be kept separate for the final prediction. Additionally, 20% of the data for model training will be used as validation data. All data separation are done with stratification according to the target variable.

Single layer Perceptron was applied to build a predictive algorithm. Firstly, a random vector containing a set of weights (the dataset comes with eight predictor variables, hence eight weights for corresponding to each of them are randomly generated, and their value will be between 0 and 1). Then each observation from the training data will be "iterated" through the weight vector, and each time an observation is computed using the weights, the weights will be updated if it made a wrong prediction, and not updated if the prediction is correct. The way that each observation iterates through the weight vector is by computing the dot product between the weight vector and the vector containing the predictor variables of each observation: $x_i.w_t$ .Where x_i is the vector containing the predictor variable of an observation and w_t is the weight. If the results is greater than 0, then the observation will be classified 1, and it will be -1 other wise. When an error is made, the weight is updated using the following formula:

$$w_{s+1} = w_s + \eta y_i x_i \tag{1}$$

And for the whole dataset for training the model, the weights will be updated as:

$$w_{t+1} = w_t + \eta \sum_{i=1}^{n} y_i x_i * 1_{y_i*(x_i.w_t)<0} \tag{2}$$

Where:

- w is the weight vector
- t indicate the weight vector as it is, t+1 indicate the updated weight vector after the entire dataset is iterated (which we will call an epoch)
- s, s+1 indicates the weight update sequence in one epoch
- x_i is the predictor variable for the ith observation
- y_i is the label of the ith observation
- η is the learning rate of the model

The formula is the result of minimizing the average perceptron loss with respect to the weights, calculated as:

$$R_n(w) = \frac{1}{n}\sum_{i=1}^{n} max\left[0, -y_i * (x_i.w_t)\right] \tag{3}$$

And minimization is done via gradient descent:

$$w_{t+1} = w_t - \eta' \frac{\delta R_n(w_t)}{w_t} \tag{4}$$

(All derivation are done accordingly to the lecture slide [2], and $\eta' = \eta * \frac{1}{n}$)

This iterative process will be done through multiple epochs to keep on updating the weight to either reach or get as close to a minimum as possible [4]

2.2. Additional implementation

The outcome of the Perceptron will be a classifier that will classify the observation based on the following equation:

$$\sum_{z=1}^{8} x_{i_z} w_z \tag{5}$$

(z corresponds to the $z^{th} feature$)

Which is a linear function of the inputs. Consequently, if the boundary between the two classes, is not linearly separable according to the predictors, Perceptron would not be able to reach a good prediction. However, this project is limited to the use of Perceptron, no other algorithm will be looked at.

The baseline algorithm does not have a well-defined stopping criteria, aside from the number of epochs to train the weights. For this project, early stopping will be implemented such that when the model reach a minimum point (either local or global), any more attempts to update the weight may pushed the weights out of the optimal point and getting worse result. To implement this, during the training run, validation accuracy will be calculated at each epoch as a measure to keep track of the model performance after an update. To indicate that the model may have already reached an optimal set of weight, the number of epochs since the last improved accuracy was observed is used. For instance, at an arbitrary epoch 10, a validation accuracy was observed to be 0.7, and after the next 10 epochs (also an arbitrary number used for example), no value greater than 0.7 is observed, the training will be stopped. The set of weights that returned the best validation accuracy will be returned should the training run hit the early stopping criteria.

Another experimentation done in this project will be controlling the number of observations to be iterated before updating the weights. Similar to stopping criteria, the baseline algorithm also does not mentioned if the weight should be updated during an epoch or not, for example, as soon as one error is found, immediately update the weights, or use the weight to predict the entire dataset and use all the error to make the update. This may have significant effects on the training process as for gradient descent, a learning rate too large or too small is undesirable as both could deteriorates the optimization process(Geron 2019) [3]. Hence, if the weights are updated using the aggregated error after the whole dataset is iterated, the resulting update could be a step too far, even if the learning rate is set to a small value. This case was observed in this experiment through the oscilating validation and training accuracy in successive epoch when weights are updated after a epoch. The parameter used to control the number of observation that passed through the weight before updating will be called batch size.

Additionally, with smaller batch size, the order in which the observations passed through the weight could be altered from one epoch to another, as their order does affect how the weight are changed. Hence, this report also experiments whether shuffling the dataset affect the accuracy of the model.

Finally, adding a bias and tuning the optimal learning rate for the model will also be considered for this report. For the former, having a bias will be viewed as having another predictor variable in the dataset, all observations will have the same value of 1 for this variables (in the no bias case, all value will be 0, hence can be ignored). An additional weight will also be added to the weight vector and the training process can be done consistently with what was outline in the previous subsection. For the learning rate, this will involves testing the model with different learning rates to see if a different value lead to a significant improvement in performance.

3. Experimental analysis and results

In the experiment, all models built and tested start with the same set of random weight, and their learning curve through each epoch (their training/validation accuracy) are observed. The default specification of the model will have learning rate of 0.001, no bias term and will be trained through 50 epochs.

To verify that the baseline algorithm work, a dummy classification problem that is perfectly linearly separable were created and tested using the created model. The model were able to achieve perfect accuracy scores on that problem, the result is assessible in the provided link in the implementation-code section.

3.1. Batch size and shuffle

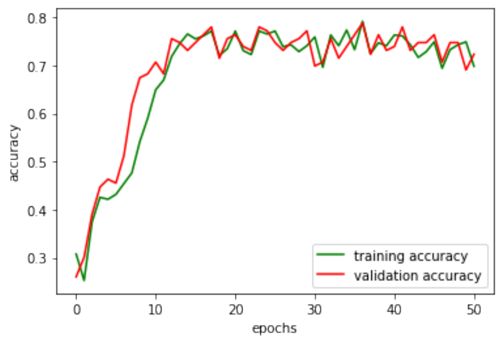

Figure 1. update weight for every observation

Looking from figure one to three, as the batch size increases, the volatility of the validation and training accuracy from one epoch to another also increases. Even though the higher performances for the last batch size is comparable to the other 2 when they converge, this updating weight after the whole dataset has iterated through the weights seems

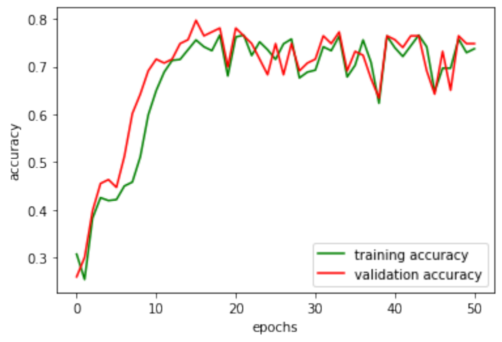

Figure 2. update weight after 10% of the observations passed

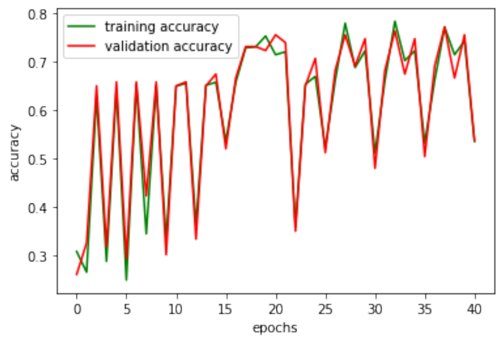

Figure 3. update weight after the whole dataset has passed

undesirable. For the other two, updating weight for every observations result in a slightly more stable convergence.

One surprisising results from the experiment was that shuffling also create unstability in the optimization process similar to what was observed with the effect of batch size.

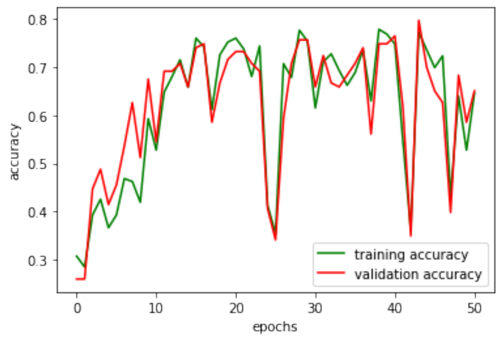

Figure 4. update weight for every observation and shuffling the data

Models in figures 4 and 5 has the same specification as those in figure 1 and 2, except for the shuffling data. Yet their accuracy measures do not converge, although they also reached similar accuracies measure at their best.

Although shuffling data or have a larger number of batch

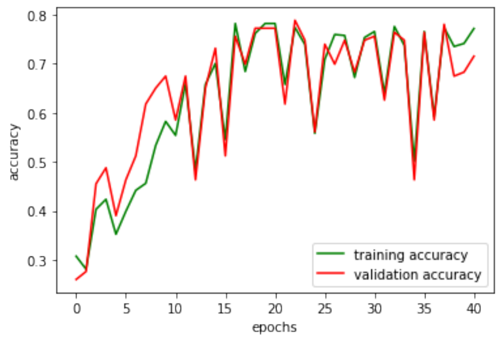

Figure 5. update weight after 10% of the observations passed and shuffling the data

size can still lead to a Perceptron with decent predictive power on the dataset, methods with converging accuracy will be preferred for this project, and hence will be used for tuning the model.

3.2. Tuning and improving the model

Tuning the model in this project involves testing whether the permance of the Perceptron can be improved by adding a bias term to the model, running the training process with an early stopping criteria and running the model with different learning rates

3.2.1 Bias, early stopping and learning rate

As mentioned, to include a bias, the weights vector of the Perceptron and the number of features for each observation will be extended to 9 weights and 9 features, respectively. For the weights, the extra term is the value of the bias, which will also be randomly initiated as the weights, a value between 0 and 1. As for the extra feature, all observations will have a value of 1 for this extra feature, and hence equation (5) becomes:

$$\sum_{z=1}^{8} x_{i_z} w_z + bias \tag{6}$$

This helps the Perceptron to be more active in determining what class an observation belong to should all features has the value of 0, as if the default setting is used, the model will classify an observation with all-zero features as a negative class, and if this is an error, the weight cannot be updated with this information as the update equation 1 when this error is encountered will result in a net 0 update.

For early stopping, the model is programmed to stop after running 12 epochs without finding a set of weight with a better validation accuracy. A more rigorous early stopping criteria could be applied, such as running a certain number of epochs without finding a set of weight that returns a better validation accuracy by 5% or more. However, one goal of using early stopping is to help the model trains more efficiently if a computationally heavy model is run, which is not the case for the Perceptron, so only the loose criteria will be applied.

Finally, the learning rate tested in this project are 0.1, 0.01, 0.001 and 0.0001, ranging from updating the weight relatively quick to relatively slow.

3.2.2 Results

Bias	Learning rate	Early stopping (epoch run)	Training accuracy	Validation Accuracy
No	0.1	No	0.739	0.772
No	0.1	Yes (23)	0.758	0.805
Yes	0.1	No	0.768	0.707
Yes	0.1	Yes (18)	0.770	0.780
No	0.01	No	0.725	0.756
No	0.01	Yes (29)	0.786	0.780
Yes	0.01	No	0.731	0.732
Yes	0.01	Yes (28)	0.758	0.789
No	0.001	No	0.699	0.724
No	0.001	Yes (30)	0.772	0.780
Yes	0.001	No	0.711	0.642
Yes	0.001	Yes (50)	0.770	0.780
No	0.0001	No	0.436	0.472
No	0.0001	Yes (50)	0.428	0.480
Yes	0.0001	No	0.436	0.488
Yes	0.0001	Yes (37)	0.418	0.504

Table 1. Results of model testing

The presented results show that validation and training accuracy are relatively close to each other, indicating that the model is not overfitting. Although it is unusual that the validation accuracy is higher than training accuracy most of the time, but this could be due to random sampling effect as the dataset is somewhat small.

Looking at how results vary by bias, models without it tend to perform as least as good as those that do have one. Although the performance differences is not very significant, it would still be favourable to not have a bias in the model.

Having early stopping criteria helps improve the model accuracy. Furthermore, the improvement is even more significant compared to the former as all models with this criteria outperform those without it. This indicates that the criteria is implemented appropriately, and did not stop the training process prematurely, rather than any mathematical or systematical improvement to the model. Also, most model with it actually stopped before reaching 50 epochs, and only two out of eight did not, signalling that early stopping is does help the process to be more efficient.

Finally, for the learning rate, the three highest one gives very comparable results, and only $1e-4$ return models that dip in performance signifcantly. This can be expected when a learning rate is too low, the model's convergence rate is also reduced, and the model takes significantly longer to converge.

For the three other learning rates, samples of their learning curves are as follow:

All three models from figure 6 are Perceptron without bias, use early stopping and only differs in learning rate. Despite reaching comparable validation accuracy, the last of them requires significantly more epochs to converge (around 20), while both models with higher learning rates seem to converge before the fifth epoch.

3.2.3 Performance on the test dataset

Using the result from the model with the best validation accuracy (no bias, learning rate=0.1 and with early stopping), the test accuracy that this model obtain is 0.734. Although it is not as high as the validation results, it is not considerably far enough to say there is overfitting.

It should be noted that the Perceptron did not achieve a great accuracy, this is because the model only works well if the classes can be separate via a linear function. However, it is not the case for this model:

The distribution shown in the boxplots 7 for the two classes has considerable overlapping ranges for multiple predictors. Only Glucose (top right plot) and Insulin (third row, left plot) does not, but are not perfectly separable. As a result, the model did not perform so well

4. Implementation-code

All codes for this project are accessible from `https://github.com/phucthai67/COMP-SCI-7318-assignment-1.git`

5. Conclusion

This project involves implementing a Perceptron algorithm on the diabetes dataset and testing various parameters and model specification to improve its performance.

The testing includes: controlling batch size to influence how the weight is updated for a stable and quick convergence, shuffling the dataset to change how the data and the weight interact with each other, using bias term to help the model to have more autonomy in determining what an observation with all-zero features should be classified, early

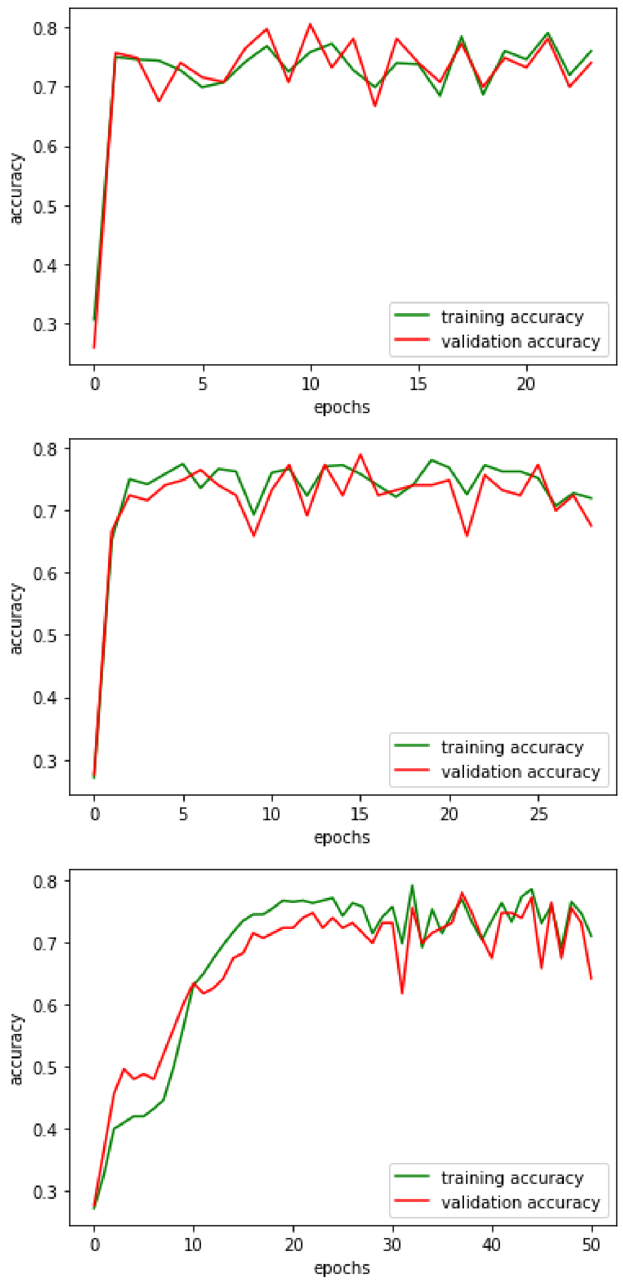

Figure 6. Learning rate 0.1, 0.01 and 0.001 from top to bottom

stopping to improve model efficient and help the model returns the optimal weights and changing learning rate to influence convergence speed.

Should the Perceptron be tested further, the effect of learning rate could be explored further as in this project, the learning behaviou of learning rate 0.1 and 0.01 are very similar to each other, hence higher learning rate could be further examined to see how they affect the model.

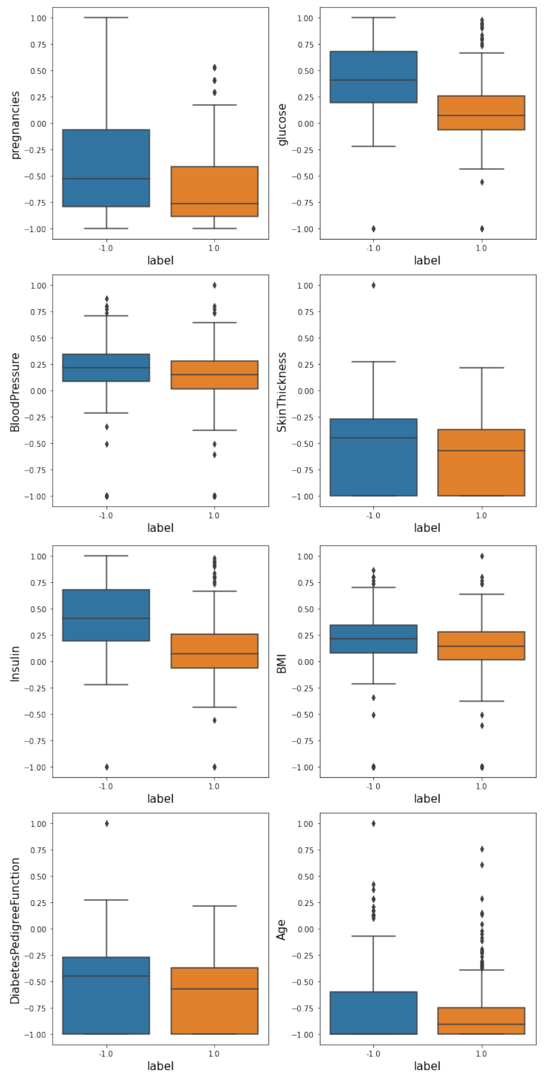

Figure 7. Boxplots of features against the label

References

[1] CSIE , LIBSVM Data: Classification (Binary Class), `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

[2] Dick, A & Shi, J 2022, Supervised Learning: KNN, Perceptron, Logistic Regression, Part 3' lecture notes distributed in COMP SCI 7318 University of Adelaide, on 12 September 2022.

[3] Géron, A 2019,'Training models' in *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, 2nd edn, O'Reilly Media, Incorporated, Sevastopol

[4] Higham, CF & Higham DJ (2018), ’Deep Learning: An Introduction for Applied Mathematicians’, Cornell University, DOI:10.48550/ARXIV.1801.05894.