A Single-layer perceptron model for predicting Diabetes

Melroy Elias Pereira
University of Adelaide
melroyelias.pereira@student.adelaide.edu.au

Abstract

Diabetes is a metabolic disease caused by low insulin in the body and thus it increases blood sugar and it is the most common disease nowadays. In the medical field, various tests are made to check the levels of blood sugar mainly fasting blood sugar test, and it is usually done by humans, therefore there is a human error in the calculation, however, with an advent, new technology such as Artificial Intelligence (AI) gives better and more accurate results using past data on diabetes, such as Pima Indian diabetes dataset which is one of the many diabetes datasets.
AI uses the algorithms from Machine Learning and deep learning to predict the classes (diabetes or not diabetes) for diabetes dataset. Results show that, many researchers were able to get an accuracy above 70%, however, my proposed approach has an accuracy of up to 85%, and moreover, the proposed approach can be used for Predicting and monitoring the diabetes.

1. Introduction

Diabetes is growing rapidly and nowadays it is also found in children's. According to the International diabetes federation, there are 463 million adults within the age of 20-79years have diabetes and if this continues by 2045 the number of diabetes in people will increase to 700 million[1] therefore, predicting and monitoring one's glucose level is important.
In the medical field, various clinical tests are made such as blood tests in the early morning, and these clinical tests are complex and requires precise clinical procedures thus human error might encounter, and doing such tests on a daily basis will be very expensive thus, AI can be used in such situation and it does the work precisely without any human intervention.
AI uses Machine learning and Deep learning algorithms to analyze diabetes and it also predicts if a person has diabetes or not, by using past data. Here, the dataset contains various variables such as glucose level, insulin, BMI, the weight of the person, skin thickness, etc. since there are more variables involved it is difficult to understand for humans, but computers can do the job.
Algorithms of Machine learning and deep learning will extract the hidden patterns from the data, and this data contains the information, and then it is used for prediction.
Here, we divide the data in three sections such as, train data, validation data and test data and we extract the pattern from the data and later it is tested on test data which we call it as unseen data, and the results are evaluated using actual data and the results get succeed when the error is very small in testing data.
In this paper, I have implemented 7 different variations of models in single layer neural network. The primary or the base model is perceptron model with perceptron loss as a loss function, since this model is a linear classifier it is difficult to predict with non-linear data hence gives an accuracy of 70%, thus I am improving perceptron's accuracy by using some other techniques so as to get an accuracy of at least 80% so that this linear classifier can still compete with other non-linear classifiers.
In summary, my contribution to the literature can be listed as follows;

- Using the shuffling algorithm to shuffle the dataset to remove the bias from data while in training phase.
- I am using L2 regularization as a penalty to the coefficients of the model so that coefficients are small and using L2 regularization coefficients can shrink but it won't become zero.
- Using other activation functions and loss functions to improve the accuracy, I am using sigmoid as my activation function and cross-entropy loss as a loss function.
- Finally improving the perceptron's inputs by combining Random forest and single-layer neural network together as Forest single-layer neural network (FSNN).

I have not found much literature on FSNN, however I did found Forest deep neural network (FDNN) [2] by Yunchuan Kong et al, by using FSNN it improved the accuracy from 70% to 78.59% near to expected result, overall by improving perceptron I was able to get an accuracy of 85% in this literature.
The organization of this paper is as follows: section 2 starts

with the perceptron algorithm (Base Model) and later, other models are proposed to improve the performance of the perceptron algorithm. Section 3: gives a detailed analysis of the experimental results of the different models. Section 4: Result comparison. Section 5: conclusion and future work. Section 6: Supplementary papers are given which contains Data analysis of PIMA Indians diabetes dataset, Algorithm Steps for the models.

1.1 Related Works

It is said that one single model cannot be used for every dataset, because whenever the new data is generated the underlying patterns in the data changes comparing to the previous data therefore the distribution of the data also changes, hence, we need to analyze for every data.

Pima Indian Diabetes dataset has been used by many researchers and many journals are published however for this literature, I have referred following papers which contains various models such as Support Vector Machine (SVM) by Madam chakradar et al [3], Random Forest (RF) by Dr. E. Jebamalar Leaveline et al [4], Artificial Neural Network (ANN) by M.S Barale [5], Multi-layer feed-forward neural network by Michael Dutt et al [6] by using other methods such as cross-validation, dimensional reduction, and outlier detection.

A case study on Current techniques for diabetes prediction by Souad Larabi-Marie-Sainte et al [7], describes the recent technologies involved in the predictions of diabetes such as Machine learning and deep learning classifiers.

An expert system for diabetes prediction using the auto-tuned Multi-layer perceptron (MLP) by Maham Jahangir et al [8] describes the use of AutoMLP which is an ensemble of four MLP's to achieve higher accuracy for predicting diabetes.

Diabetic retinopathy diagnosis using neural network arbitration by Ebenezer olaniyil [9] which describes diagnosing the diabetic retinopathy using neural networks.

In this literature, I will be using the perceptron single-layer neural network with perceptron loss as a loss function and this algorithm later is improved by other techniques such as using the shuffling algorithm, using L2 regularization and using different activation functions such as sigmoid and loss functions as a cross-entropy function.

. .

2. Methods and Algorithm

In this paper for predicting diabetes, we are using the Perceptron Algorithm of single-layer neural network and a typical perceptron looks like as shown in the Figure 1.

Perceptron:

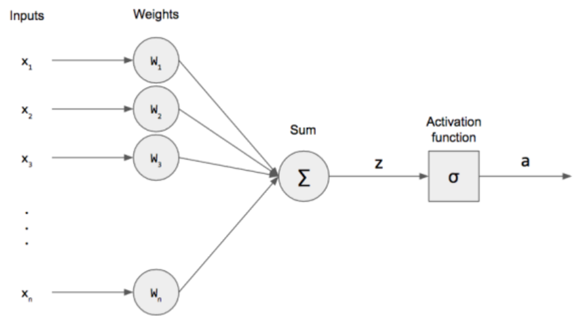


Figure 1: Perceptron

The Perceptron is a supervised machine learning algorithm used for learning and predicting binary classes and it was invented by Frank Rosenblatt, and perceptron using one input, and an output layer is also called as *Single layer neural network*.

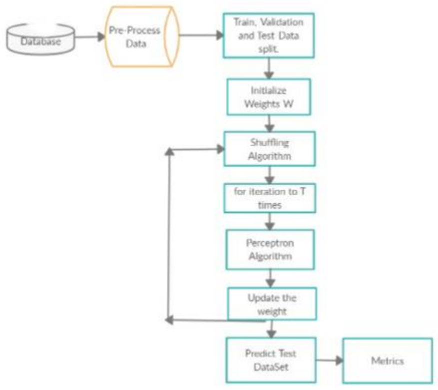
Here, $W_{1,}$ $W_{2,}$ $W_{3\ldots}W_N$ is the weights for the output layer for each feature and it can also be represented as W^T and $X_{1,}$ $X_{2,}$ $X_{3\ldots\ldots}X_N$ is the input feature for the perceptron and it can also be represented as X^T.
"z" is the output of inputs at Sum ($\sum$) in figure 1.
σ is called as a activation function, here activation function can be perceptron loss or sigmoid function, etc.
"a" is an output from the activation function.
The mathematical definition of the perceptron is that

$$\begin{cases} 1 & \text{if } W^T X > 0 \\ 0 & \text{if } W^T X \leq 0 \end{cases} \quad \text{----- (1)}$$

The value [1,0] is the threshold value for the predicted values, and the value of the prediction is always within the range of [0,1], and if $W^TX > 0$, we say that it's class 1 and if the $W^TX <= 0$, we say that it is belonging to the class 0. Often sometimes, people use -1 and 1 as a class.
Where -1 is diabetes and +1 is the not diabetes, whereas in (1) class 1 is diabetic and class 0 is non-diabetic.

.

Workflow:



Flowchart 1: Perceptron process

Working process:

Perceptron is a single layer neural network, consisting of one input layer and one output layer and without any hidden layers between the input and output layer. From Figure 1, we can see that the input layer consists of features $X_1, X_2, X_3 ..... X_N$ and each feature have it's own weight $W_1, W_2, W_3 .... W_N$. At the output layer we combine the input and weights passing to the single neuron and it is represented as Summation of inputs and it does the following operation:

Let the input be $\{(x_i, y_i)\}_{i=1}^{N}$ with some step-size as η and iterating for *T* times.

$$Z = \sum_{i=1}^{N} W_1X_1 + W_2X_2 + ... + W_NX_N \qquad \text{------ (2)}$$

From the Flowchart-1, we can see that inputs from the database are standardized using the MinMax scaler method in pre-process data therefore the values are within range of [0,1].

Scaled data are divided into three sections such as training data, validation data, and testing data. In training data we train the model and the best fit models parameter from training dataset is used for validation dataset, and in the validation dataset we use regularization as a penalty for coefficients thus it makes the weights smaller but not zero. If the validation and testing dataset comes from the same distribution they might have some similar patterns.

The perceptron algorithm starts with initializing the weights W and the weight is initialized for a single-layer neural network in my paper is (Number of features, Number of nodes in the output).

Example: if the dataset has 8 features and one output neuron then the weight matrix is (8,1).

Most of them use K-fold cross-validation, however, in this work I have used my made-up shuffling algorithm that shuffles the dataset and gives back as an input for the neural network which will be later used in the training phase.

<u>Steps for Shuffling Algorithm:</u>

<u>Step 1</u>: set shuffle count = 1
<u>Step2</u>: while(shuffle count <= shuffleTotal):

(shuffleTotal ->total number of times shuffle to be made).

<u>Step 3</u>: Take the total length of the sample size of X-Training data as "Xlength"
<u>Step4:</u> Import the random function from NumPy to generate random values within the range of (0, Xlength)
<u>Step5</u>: Random value generated will be used as the index value of each row of the X training and Ytrain data and set each row according to the random data generated.

The shuffled dataset is used as the training data in the training phase and it is iterated several times and in the meantime, it calculates the error made after prediction, however, the loss and cost functions are calculated based on the perceptron loss.

$$PerceptronLoss(x, y, w) = max\left\{0, -y < x, w >\right\} \qquad \text{----- (3)}$$
-

$$Cost = \frac{\sum_{i=1}^{N} PerceptronLoss}{N} \qquad \text{------- (4)}$$

Cost calculated has to be minimum and closer to zero thus it minimizes the prediction error, however, sometimes, it does not happen at the first iteration itself, so we need to iterate it more than once and we update the weight because keeping the same weight for every iteration gives the same results, thus we use gradient descent method to update the weight, using gradient descent we tend to push cost value to the zero, however, parameters like learning rate which should take into consideration because giving high learning rate will behave erratic(overshoot) and giving very small learning rate will not converge faster.
Weights are updated using a Gradient descent method.

$$W_{new} = W_{old} - \eta * \frac{\partial Cost(L, w)}{\partial w} \qquad \text{------- (5)}$$

This is the general formula for gradient descent.
Where,

W_{new} is the new weight

W_{old} is the old weights.

η is the learning rate

$\frac{\partial Cost(L,w))}{\partial w}$ is the gradient of cost function w.r.t weights.

$$\frac{\partial Cost(L,w)}{\partial w} = \frac{\partial \frac{1}{N}\sum_{i=1}^{N} PerceptronLoss}{\partial w}$$
$$= \frac{1}{N}\frac{\partial \sum_{i=1}^{N} max(0, -y < x, w >)}{\partial w}$$
$$= \frac{1}{N}\frac{\partial \sum_{i=1}^{N} -yx_i}{\partial w} \quad \text{------- (6)}$$

Thus, the updated equation for updating with weight is,

$$W_{new} = W_{old} + \frac{\eta}{N} * \frac{\partial Cost(L,w)}{\partial w} \quad \text{--------- (7)}$$

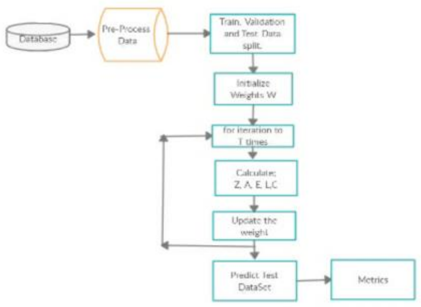
$$W_{new} = W_{old} + \frac{\eta}{N}\frac{\partial(Cost(L,w))}{\partial w} - \lambda * \eta * W_{old} \quad \text{--------- (8)}$$

Equation (7) is the gradient descent without the L2 regularization Model and Equation (8) is the gradient descent method with L2 regularization.
After T iteration, the minimum cost value is found in the training phase and then it's corresponding weights used in validation phase by following the below steps:

Step1: Get the Validation data
Step2: use the parameters from the training phase and predict and calculate the error, loss, and cost.
Step 3: update the weight using the gradient descent method with adding penalty as L2 regularization
Step4: start from step2 again.
Step5: After T iterations find the minimum cost and its corresponding parameters.

After T iteration finds the minimum cost and it's corresponding weights and uses that in predicting the test dataset. However, the output of the function is determined by considering perceptron's definition and itself acts as an activation function which transforms the Z value to either 0 or 1 and that is from (1) after adding all the inputs at the node if the resulting value is greater than 0, we call it as class "1", otherwise, we call it as class the "0".
To check the performance of the model we used metrics such as Accuracy, F1 score, and misclassification rate.
However, I found the accuracy of the algorithm about 70%, therefore, I used some methods to improve the accuracy up to 85% using different single-layer neural network, in this paper, I have used 6 other single layer models(excluding perceptron algorithm).

2.1 Single layer perceptron using sigmoid with batch gradient descent



Flowchart 2: Perceptron Gradient descent (Model 3)

In this algorithm, we are using 3 sub-models of different parameters, and the parameters are selected by trial and error method and the analysis of these models will be seen next section (Experimental Analysis).

The algorithm first starts with splitting the data into train, validation and testing data and then we initialize the weights based on (Number of features, Number of nodes in the output), after initializing we pass the input features with weights to the output layer, where it combines all the inputs according to (2) and the calculated value is then passed to the sigmoid function.
Activation function determines the output and the activation function here we used is a sigmoid function and it is non-linear activation function with values ranging from [0,1].

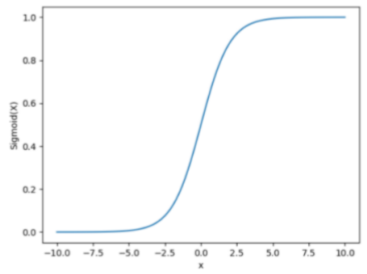


Figure 2: Sigmoid Activation Function

$$\sigma(Z) = \frac{1}{(1 + exp^{-Z})} \quad \text{----------------- (9)}$$

To determine the output we use the boundary method that if the sigmoid probability value is greater than 0.5 we say it belongs to class "1" and if the probability is below or equal to 0.5 we say it belongs to class "0".
After classifying, we check the error using loss function and cost function. In this algorithm, we used cross-entropy as our loss function and the cost function is the mean of

the loss function and we expect cost function to be small and closer to zero.

$$Loss = -(Y_a \cdot log(Y_p) + (1 - Y_a) \cdot log(1 - Y_p))$$ - ----------- (10)

$$Cost = \frac{\sum_{i=1}^{N} Loss}{N}$$ -------- (11)

And weight is updated is using batch gradient descent method and to calculate the gradient, we partially differentiate Cost function w.r.t weights W

$$\frac{\partial Cost(L, w))}{\partial w} = -\frac{1}{N}\frac{\partial \sum_{i=1}^{N}(Y_a \cdot log(Y_p) + (1 - Y_a) \cdot log(1 - Y_p))}{\partial w}$$

$$= \frac{1}{N}\sum_{i=1}^{N}(Y_p - Y_a)X$$ - ----- (12)

Y_{a} = Actual Y values
Y_p = Predicted Y values
The weight is updated using,

$$W_{new} = W_{old} - \eta * \frac{\partial Cost(L, w)}{\partial w}$$ ------- (13)

After T iteration, minimum cost value is found and the corresponding weight is taken for predicting the test data set.

<u>2.2 Single layer perceptron using sigmoid with batch gradient descent and L2 regularization</u>

The Algorithm is similar to Algorithm 2.1, but it uses L2 regularization and in this model, we have 3 sub-model with different parameters, by using L2 regularization as a penalty and makes the coefficients smaller but not zero.
Parameters are selected by trial and error method and the analysis of these models will be seen next section (Experimental Analysis).

Here we are using sigmoid as the activation function. The Algorithm first starts with initializing the weights based on (Number of features, Number of nodes in the output) after initializing we pass the input features with weights to the output layer, where it combines all the inputs according to (2) and the calculated value is then passed to the sigmoid function.

Sigmoid function determines the output based on the probability that is calculated by sigmoid. The probability value is greater than 0.5 we say it belongs to class "1" and if the probability is below or equal to 0.5 we say it belongs to class "0". After classifying, we check the error using loss function and cost function. In this algorithm we used cross-entropy as our loss function we expect cost function to be small and closer to zero, the loss function is given as (10) and cost function will be adding L2 regularization as a penalty, where λ is a tuning parameter.

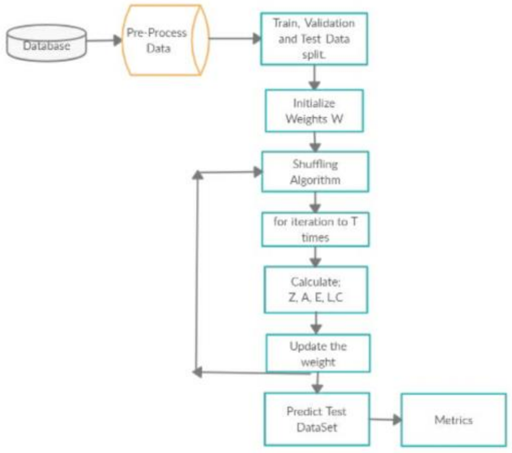
$$Cost = \frac{\sum_{i=1}^{N} crossentropyloss}{N} + \frac{\lambda \left\|W^2\right\|}{2 * N}$$ ---------- (14)

After we calculate cost, we update the weight and weight is updated using

$$W_{new} = W_{old} - \frac{\eta}{N}\frac{\partial Cost(L, w)}{\partial w} - \eta * \lambda * W_{old}$$ -------- (15)

After T iteration, minimum cost value is found and the corresponding weight is taken for predicting the validation data set and then again minimum cost value of validation dataset is found and its corresponding weight is used for predicting test data set.

2.3 <u>Single layer perceptron using sigmoid with batch gradient descent and shuffling the dataset</u>



Flowchart 3: Algorithm 3

In this algorithm, the core concept is the shuffling the dataset and using sigmoid activation function with cross-entropy loss. The algorithm of the shuffling is same as earlier shown in the perceptron algorithm.
The model starts with dividing the dataset into the train, and test data and weights will be initialized for the training

phase for the first iteration. At first iteration it takes the first shuffled data for the training phase and learns from the data and it returns for the new shuffle

After each shuffle, the shuffled dataset is undergone into the training phase to learn from the data, and after all the iterations of the training phase it updates the weight of the model using gradient descent method to minimize the cost function, the equation for gradient descent is same as (13).

Best Parameters for the model is selected after all the iterations with minimum cost function and the corresponding weight will be taken as best parameter selected for testing test dataset and every time the weight W will be learned through all the iteration without re-initializing after each shuffle and the results are averaged.

2.4 Single layer perceptron using sigmoid with shuffling the dataset and batch gradient descent –L2 regularized.

In this algorithm, the core concept is the shuffling of the dataset and using a sigmoid activation function with cross-entropy loss. The algorithm of shuffling is same as earlier show in the perceptron algorithm

Parameters are selected by trial and error method manually, however, here we also use L2- regularization as a penalty for the coefficients.

The model starts with the first shuffled dataset and the splitting the dataset into the train, validation and test data, and later weights are initialized.

After initializing we pass the input features with the weights to the output layer, where it combines all the inputs according to (2) and the calculated value later it is passed to the sigmoid function.

Now, sigmoid function decides the classes based on the probability it measured, the probability value is greater than 0.5 we say it belongs to class “1” and if the probability is below or equal to 0.5 we say it belongs to class “0”.
Later error is measured and along with loss and cost function for the model and loss function is the same as (10) and cost function is the same as (11).

After T iteration, the minimum cost value is found and the corresponding weight is taken for predicting the validation data set and then again minimum cost value of validation dataset is found and its corresponding weight is used for predicting the test data set and next shuffle starts again with the same steps as above. Metrics are used for measuring the performance of the model, here we use accuracy, F1 score and misclassification rate as metrics.

2.5 Random forest single layer neural network

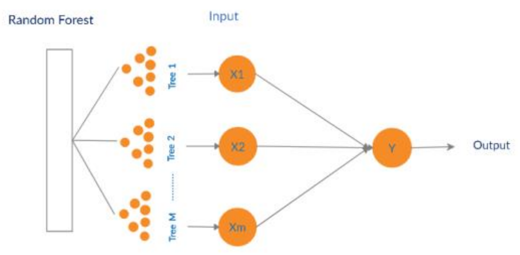


Figure 3: Random Forest single layer neural network

Random forest single-layer neural network is similar to the Forest deep neural network (FDNN), however, I have implemented for single layer neural network (perceptron).
The objective of using it for the perceptron is to improve the model performance using the same perceptron algorithm using the Machine learning outputs for the deep learning’s input. We know that random forest is a good classifier in Machine learning, therefore it is used for this paper.

The Algorithm starts with dividing the dataset into train and test datasets and the train dataset is used for training in the random forest.
Here, we encountered a problem that it gives us the output in either (1, length of Y) or (Length of Y, 1) shape. Which is smaller than the original dataset, therefore we overcome the by stacking each tree model.
Algorithm steps:

Step 1: Let, X train as (8, 700) where 8 features and 700 samples, the total trees are 100 that is T = 100
Step 2: Shuffle the X train dataset using the same algorithm from perceptron.
Step 2: Each sub-tree T_i from the main T trees is used for random forest classifier.

Example: for each iterations that is iteration =1 from T =100. We consider that iteration value as one tree and the output of that tree is stacked, here for the first iteration trees(previous tree)..

Similarly, for second iteration =2 from T=100, the tree here is 2 and the output is stacked with iteration 1.
And it continues till last Tree and finally we will have (100, 700) dataset from the original dataset of (8,700), now each row is a decision made by the random forest.

Step 3: After the stacking of all the sub-trees of main tree, we get train dataset from random forest, which is of 1’s and 0’s decision made by each tree.

Step 4: Now, the output of random forest is used as the input for the neural network, in this model we use sigmoid as the activation function and cross-entropy as the cost function.
Activation function is the same as (9) and loss function used here is cross-entropy (10) and cost function is the same as (11).
Step 5: Gradient descent is used for updating the weights and after all the iterations, the minimum cost is found and it's corresponding weights are used for predicting test dataset.
Weights are updated using the same equation (13)
Step 6: Since the shape of X dataset is changed to random forest decisions that is X data set (8,700) to (100,700), we have to transform the test data set to (100, 300) from (8, 300). We have to be carefull that number of predictors should be less than the samples otherwise n<p condition will come into consideration.
Step 7: Therefore, here we will use test dataset for random forest, and will test it(fitting with train data and predicting with test data), now it might be obvious that accuracy will be 100% for random forest model, however, for perceptron it is still has not seen the test dataset and it will be normal dataset for like 1's and 0's for the perceptron model, like we used train dataset, therefore we will have a shape of (100, 300) for the test data and each row is a decision made by the random forest.
Step 8: Now, the parameter selected from the training phase is used for the classification with test data.
Step 9: shuffle the dataset again for the random forest to remove the bias so that it gives the better output and this will be used for the neural network's input.
(*Shuffle = shuffle +1)*

Thus, we can use Machine learning outputs as inputs for neural network to take better decision.

3. Experimental Analysis:

In this paper we have worked with the single-layer neural network using the main algorithm as perceptron algorithm, however, we have also worked with other different models to improve the performance of the perceptron algorithm.
In the beginning we got an accuracy of 70% with perceptron algorithm, later with the improvements the perceptron algorithm we were able to get an accuracy of 85%.
Following are the models used for experimenting with single-layer neural networks.

1) Perceptron Algorithm without shuffling and using the perceptron loss (Model 1)
2) Perceptron Algorithm without shuffling and using the perceptron loss with L2 regularization. (Model 2)
3) Perceptron Algorithm without shuffling and using a sigmoid function and cross-entropy loss function (Model 3)
4) Perceptron Algorithm without shuffling and using a sigmoid function and cross-entropy loss function with L2 regularization (Model 4)
5) Perceptron Algorithm with shuffling and using a sigmoid function and cross-entropy loss function (Model 5)
6) Perceptron Algorithm with shuffling and using a sigmoid function and cross-entropy loss function with L2 regularization. (Model 6)
7) Perceptron Algorithm with random forest algorithm using a sigmoid and cross-entropy loss function. (Model 7)

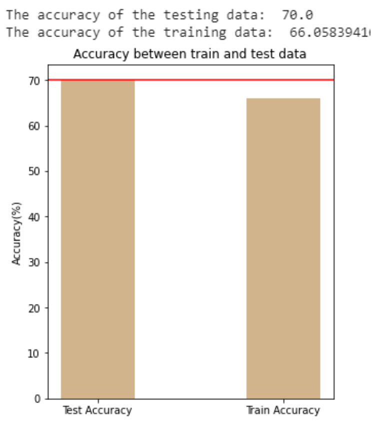
The table shows the highest accuracy that each of the model was able to get for test data:

Model Name	Accuracy	F1 score	Misclassification rate
Model-1	70%	82.35%	30.0%
Model-2	74.89%	84.79%	25.10%
Model-3	80.5%	69.33%	19.49%
Model-4	85%	75%	15%
Model-5	79.07%	68.08%	20.92%
Model-6	85%	72.72%	15%
Model-7	78.59%	68.78%	21.40%

Table 1: Model Performance

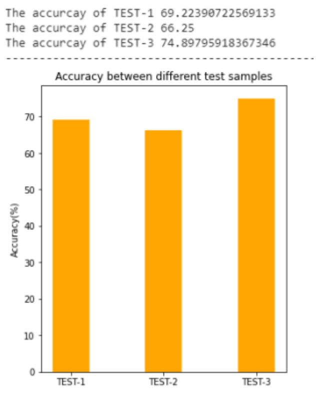
Model 1 and Model 2 are the perceptron algorithm based on perceptron loss function, In model-1, we are achieving 70% accuracy, whereas in Model-2 we are getting slightly better results that is 74.89% accuracy.
In model-1, the perceptron is trained without shuffled data therefore the accuracy was dropped, however, model-2 used shuffled dataset, therefore, it is important to shuffle the data to remove the bias from data, and it also makes the entire train data to learn.
In Model-1 we trained the data from training dataset after training we found the minimum cost and its corresponding weight. The weights from the training phase are used in the validation phase for validation data and now, we find the minimum cost from the validation with its corresponding weights, and the weights from the validation will be used for testing data prediction.
However, in model-2 we learn the parameter by shuffling the dataset, such that at each shuffling we obtain the

accuracy for that shuffled data, and after all the shuffling's we average the result.



```
The accuracy of the testing data:  70.0
The accuracy of the training data:  66.0583941
```

Figure 4: Model 1 Accuracy test data vs train data



```
The accurcay of TEST-1 69.22390722569133
The accurcay of TEST-2 66.25
The accurcay of TEST-3 74.89795918367346
----------------------------------------------
```

Figure 5: Model 2 Accuracy between test models

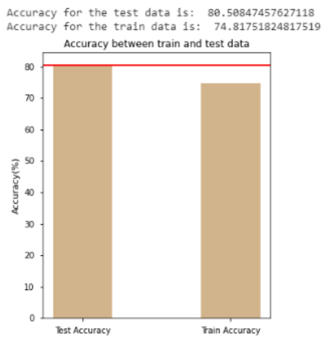
Model 3 and Model 4 are the perceptron algorithm which has a higher performance than the Model 1 and Model 2. In

Model 3 we were able to get an accuracy of 80.50% and in the Model-4 we got an accuracy of 85%.

In model 3, data is split into 70-30 rule, whereas in model 4 the data is split into 60-30-10, in these models we do not shuffle the data. In model 3&4 we are using sigmoid as the activation function and cross-entropy loss as a loss function and the weights are updated using gradient descent.

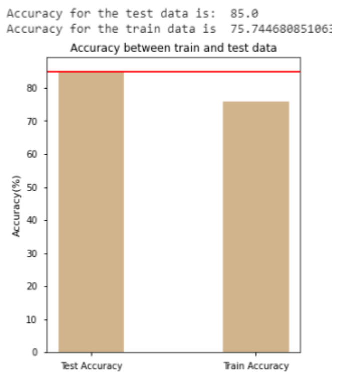
In model 3, the process is simpler, we pass the train data to iteration and at the first iteration, we predict the data using initialized parameter and then the error is measured and loss is calculated. We expect that cost will be very small and minimum cost found with its corresponding weight is then used for predicting test data. Later the weights are updated using gradient descent.

In model 4, the dataset is passed to make the prediction by taking the initialized parameter, and the corresponding error, loss and cost is calculated after each iteration, and the cost with minimum cost value is taken with its corresponding weight as a parameter and then the weights are used for validation data to train with L2 regularization and we expect that both validation and test data come from the same distribution, so the parameters selected from validation should work with a test dataset



```
Accuracy for the test data is:  80.50847457627118
Accuracy for the train data is:  74.81751824817519
```

Figure 6: Model 3 Accuracy between Test and Train data



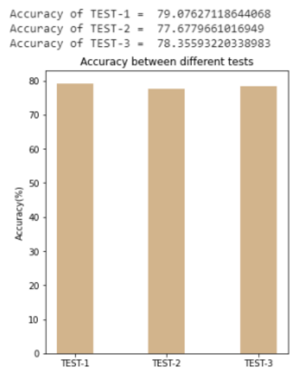
```
Accuracy for the test data is:  85.0
Accuracy for the train data is  75.7446808510б3
```

Figure 7: Model 4 Accuracy between Test and Train data

Model 5 and Model 6 we use the shuffled dataset for the prediction and we found that Model-5 was able to get the average accuracy of 79% and Model- 6 was able to get 85% accuracy for testing data.

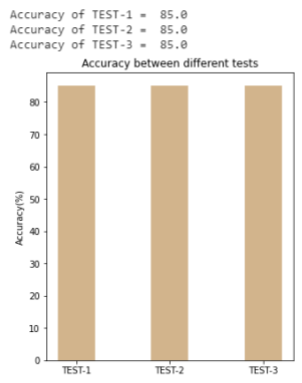
In Model – 5 the dataset is divided into train and test data and the train data was shuffled before passing to the training phase, and in the training phase after prediction, it calculates the error, loss, and cost of the model and then the weight is updated, later the minimum cost is selected with its corresponding weights and used for predicting test data, thus it completes one cycle and makes prediction for the first shuffle data, similarly it is done for every shuffle and results are stored and at the end results are averaged.

In Model-6, the work process remains same however, after selecting the weight we use that for the validation data and starts the iterative process for validation data and uses L2 regularization and later the minimum costs is found and its corresponding weights later it is selected form validation and is used for the testing test data, and this completes one cycle and starts again with next shuffle and the results are averaged.



```
Accuracy of TEST-1 =  79.07627118644068
Accuracy of TEST-2 =  77.6779661016949
Accuracy of TEST-3 =  78.35593220338983
```

Figure 8: Model 5 Accuracy between test models



```
Accuracy of TEST-1 =  85.0
Accuracy of TEST-2 =  85.0
Accuracy of TEST-3 =  85.0
```

Figure 9: Model 9 Accuracy between test models

The idea came when I was given 5 options with [1,0,0,1,1] where "1" is Yes and"0" is No, since I didn't know which one to choose, so I went with "1" because, 1 was occurred 3/5 times than "0" that is 2/5 and the decision I took was right, so I thought why not we give 1's and 0's to the neural network because neural network is intelligent enough to take better decision, however, I found very few papers that mentioning forest deep neural network, so I studied and applied to improve the perceptron performance.

Model-7 is the forest singular neural network (FSNN), in this model we use inputs from the outputs of the random forest to the neural network, and the output is predicted using sigmoid activation function with cross-entropy loss function.

In this model the inputs for the random forest are shuffled and makes the prediction in 1's and 0's, and later it is used as the input for the perceptron where the inputs are in 1's and 0's from several trees.

The main focus was to improve the perceptron performance, and the perceptron has 70% accuracy, however, I found that FSNN is able to give 78.59% or 74.39% (it depends on shuffle for the train data), thus it is higher than the perceptron.

One shortcoming I found is that it takes a lot of time to learn the data as the number of trees are increasing, however, I will try to improve the model performance and speed in the future.

4. Comparative Study

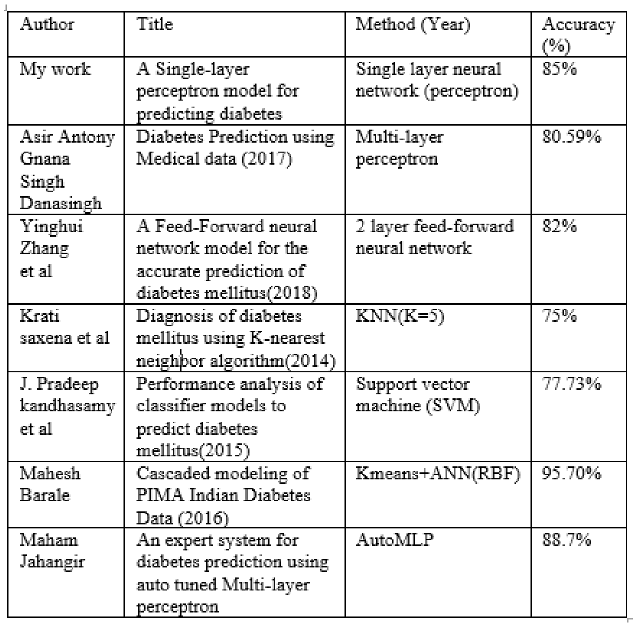
A comparative study between my proposed approach and some of the best papers available is shown in Table 2. Authors have used many different models ranging from machine learning models to deep learning.

Asir Antony Gnan singh danasingh et al has proposed Multi-layer perceptron [4] with the accuracy of 80.59%

Yinghui zhang proposed 2 layer feed-forward neural network [10] method with the accuracy of 82% and krathi saxena et al used KNN for diabetes dataset [11] and achieved 75% accuracy and J Pradeep kandhasamy et al proposed an ensemble of classifier using naiye bayes, ANN and SVM and achieved the 77.76% accuracy with SVM [12]. However, my work is able to give an accuracy of 85% which is better than the best papers referred.

However, there are more better models are available than the single-layer neural network and has achieved an accuracy of 95% by using K-means+ANN(RBF) with 10 fold cross-validation by M.S barale [5], and accuracy of 88.7% by using autotuned Multi-layer perceptron neural network by Maham Jahangir et al.[8]

My model did not perform as well as these model because single-layer neural network is a linear classifier and the data we used is non-linear data, however, there are methods available to reduce the dimensions of dataset so that data is almost linear, but our dataset was small and removing the samples and features results in loosing information from the data, thus I removed samples only with zeroes and it was still non-linear, therefore the proposed approach is good enough to classify 85% of the data.



Author	Title	Method (Year)	Accuracy (%)
My work	A Single-layer perceptron model for predicting diabetes	Single layer neural network (perceptron)	85%
Asir Antony Gnana Singh Danasingh	Diabetes Prediction using Medical data (2017)	Multi-layer perceptron	80.59%
Yinghui Zhang et al	A Feed-Forward neural network model for the accurate prediction of diabetes mellitus(2018)	2 layer feed-forward neural network	82%
Krati saxena et al	Diagnosis of diabetes mellitus using K-nearest neighbor algorithm(2014)	KNN(K=5)	75%
J. Pradeep kandhasamy et al	Performance analysis of classifier models to predict diabetes mellitus(2015)	Support vector machine (SVM)	77.73%
Mahesh Barale	Cascaded modeling of PIMA Indian Diabetes Data (2016)	Kmeans+ANN(RBF)	95.70%
Maham Jahangir	An expert system for diabetes prediction using auto tuned Multi-layer perceptron	AutoMLP	88.7%

. Table 2: Comparative study table

5. Conclusion and Future Work

In this literature, for prediction of diabetes using the PIMA Indians dataset, we used single-layer neural network and we were able to achieve 70% accuracy, however, by improving the model we were able to achieve accuracy of 85%. To improve the accuracy we used L2 regularization, and we also shuffled the dataset to learn the weights without re-initializing and at the end we combined random forest with single neural network to optimize the performance of perceptron by giving better inputs, however, without the pre-processing the data it was difficult to achieve such good accuracy.
For future work I will improve the forest single-layer neural network to give better results and to execute faster so that it can be used by anyone easily.

References

[1] International diabetes federation Atlas (2019) (URL: https://www.idf.org/aboutdiabetes/what-is-diabetes/facts-figures.html#:~:text=Diabetes%20facts%20%26%20figures,-Last%20update%3A%2012&text=In%202019%2C,will%20rise%20to%20700%20million)

[2] Yunchuan Kong and Tianwei Yu, "A deep neural network model using random forest to extract feature representation for gene expression data classification" pp , 1-9, 2018

[3] Madam Chakradar and Alok Aggarwal, " A amchine learning based approach for identification of Insulin resistance with non-invasive parameters using Home-IR", pp, 2055-2064, 2020

[4] Dr. D Asir antony gnana singh, Dr.E.jebamalar leavline, and B. Shanawaz Baig, "Diabetes prediction using medical data", pp,1-8,2017.

[5] M.S Barale and D.T shirke, " Cascaded modelling for PIMA indian diabetes data", pp, 1-5, 2016.

[6] Micheal Dutt, Vimala Nunavath and Morten Goodwin, " A multi-layer feed forward neural network approach for diagnosing diabetes"pp,300-305, 2018

[7] Souad Larabi-Marie-sainte, Linah aburahmah, rana almohaini, and tanzila saba, " Current techniques for diabetes prediction: review and caste study", pp, 1-18, 2019.

[8] Maham Jahangir, mehreen ahmed, hammad Afzal, khawar khurshid, and raheel Nawaz, "An expert system for diabetes prediction using auto tuned multi-layer perceptron", pp,1-7,2017.

[9] Ebenezer olaniyi, Adefemi Adeyemi Adekunle, Oyebade K. Oyedotun, and Adnan khashman, "diabetic retinopathy diagnosis using neural network arbitration", pp, 179-192, 2017.

[10] Yinghui Zhang, Zihan Lin, Yubeen Kang, Ruoci Ning, and Yuqi Meng, "A feed forward neural networks model for the accurate prediction of diabetes mellitus", pp, 151-155, 2018.

[11] Krati Saxena, Dr. zubair khan, and shefali singh, "Diagnosis of diabetes mellitus using K-nearest neighbor algorithm", pp, 1-7, 2014.

[12] J. Pradeep kandhasamy, S. Balamurali, "Performance analysis of classifier model to predict diabtes mellitus", pp, 45-51, 2015.

[13] MD. Kamurul hasan, MD. Ashraful alam, Dola Das, Eklas Hossain, Mahamudul Hassan, "Diabetes prediction using ensembling of different machine learning classifiers", pp, 1-19, 2017.

6. Supplementary Material

Algorithm steps for the models:

Perceptron Algorithm Model 1:

Step 1: Get the pre-processed dataset and divide the data into train, validation and test dataset.

Step 2: pass the train data in training phase

Step 3: Iterate with gradient descent method till T times and save all the values in vault.

Step4: return the vault (stored values) and find the cost with minimum and its corresponding weight.

Step 5: weights are then used with validation dataset and after T iteration, find the minimum cost value with its corresponding weights for the testing test dataset.

Step 6: save the result.

Perceptron Algorithm Model 2:

Step 1: Get the pre-processed dataset and divide the data into train, validation and test dataset.

Step 2: shuffle the train data and pass the first shuffled data in training phase.

Step 3: Iterate with gradient descent method till T times and save all the values in vault.

Step4: find the minimum cost in the vault (stored values) and its corresponding weight.

Step 5: weights are then used with validation dataset and after T iteration, find the minimum cost value with its corresponding weights for the testing test dataset.

Step 6: save the result in report.

Step7: shuffle = shuffle +1 and start the next shuffle and continue the same process till total shuffle (Number of times the shuffles to be made).

Step8: After all the shuffles, get the report and average the results.

Perceptron Algorithm Model 3:

Step 1: Get the pre-processed data and divide the data itrain and test dataset.

Step 2: pass the train dataset for T iteration in gradient descent method.

Step 3:save all the results in vault.

Step 4: return the vault, find the minimum cost and its corresponding weights of training phase.

Step5: use the weights for testing test dataset.

Step 6: evaluate with metrics.

Perceptron Algorithm Model 4:

Step 1: Get the pre-processed data and divide the data into train , validation data, and test dataset.

Step 2: pass the train dataset for T iteration in gradient descent method.

Step 3: save all the results in vault.

Step 4: return the vault, find the minimum cost and its corresponding weights of training phase.

Step5: use the weights for training validation dataset and after the T iteration of gradient descent find the minimum cost of validation phase with its corresponding weight..

Step 6: use the weights for testing test dataset.

Step 7: evaluate with metrics

Perceptron Algorithm Model 5:

Step 1: get the pre-processed dataset, and divide the data into two section, train data and test data.

Step 2: shuffle = 1, while(shuffle < Total Shuffle): shuffle the train dataset and pass the shuffled data to the training phase.

Step3: after the T iteration in training phase, save the all results in vault and find the minimum cost of this shuffle and corresponding its weight.

*Step 4: weight is then passed to the test data set and results are stored in report. Next ->Shuffle = shuffle + 1
and start the process again from step 2.*

Step 5: after all the shuffle get the report and average the results.

Perceptron Algorithm Model 6:

Step 1: get the pre-processed dataset, and divide the data into three section, train data ,validation data, and test data.

Step 2: shuffle = 1, while(shuffle < Total Shuffle): shuffle the train dataset and pass the shuffled data to the training phase.

Step3: after the T iteration in training phase, save the all results in vault and find the minimum cost of this shuffle and corresponding its weight.

Step 4: weight is then passed to the validation data set and minimum cost is calculated and corresponding its is found.

Step 5: pass the weights from validation phase to the test dataset and save the results in report. Next ->
Shuffle = shuffle + 1 and start the process again from step 2.

Step 6: after all the shuffle get the report and average the results.

Perceptron algorithm Model 7:

(Its given in the random forest single layer neural network)

Experimental Results for the test data:

Model 1:

Loss function: Perceptron loss
Dataset: Not shuffled
Batch Gradient descent for updating weights

Model Name	Accuracy	F1 score	Misclassification rate
Model - 1.1	79.66%	78.78%	35%
Model – 1.2	70%	82.35%	30%
Model – 1.3	70%	82.35%	30%

Model 2:

Loss function: Perceptron loss

Dataset: Not shuffled
Gradient descent for updating weights

Model Name	Accuracy	F1 score	Misclassification rate
Model 2.1	69.22%	81.81 %	30.77%
Model 2.2	70%	82.35 %	30%
Model 2.3	74.89%	84.79 %	2510%

Model 3:

Loss function: cross entropy loss
Dataset: not shuffled
Gradient descent for updating weights.

Model Name	Accuracy	F1 score	Misclassification rate
Model - 3.1	79.66%	72.09%	20.33%
Model – 3.2	78.81%	71.26%	21.18%
Model – 3.3	80.5%	69.33%	19.49%

Model 4:

Loss function: cross entropy loss
Dataset: not shuffled
Gradient descent for updating weights.

Model Name	Accuracy	F1 score	Misclassification rate
Model - 4.1	77.5%	60.89%	22.5%
Model – 4.2	85%	75%	15%
Model – 4.3	82.5%	72%	17.5%

.

Model 5:
Loss function: cross entropy loss
Dataset: shuffled
Gradient descent for updating weights.

Model Name	Accuracy	F1 score	Misclassification rate
Model - 5.1	79.07%	68.08%	20.92%
Model – 5.2	77.67%	58.99%	22.32%
Model – 5.3	78.35%	62.03%	21.64%

Model 6:
Loss function: cross entropy loss
Dataset: shuffled
Gradient descent for updating weights.

Model Name	Accuracy	F1 score	Misclassification rate
Model - 6.1	85%	72.72%	15%
Model – 6.2	85%	72.72%	15%
Model – 6.3	85%	72.72%	15%

Model 7:
Loss function: cross entropy loss
Dataset: shuffled
Gradient descent for updating weights.

Model Name	Accuracy	F1 score	Misclassification rate
Model - 7.1	78.59%	68.78%	21.40%
Model – 7.2	64.02%	57.17%	35.79%
Model – 7.3	65.45%	56.23%	35.54%

Data Analysis

- Data used here is PIMA Indian diabetes data, available at (https://www.kaggle.com/uciml/pima-indians-diabetes-database)
- The Pima Indian diabetes dataset has 768 samples and 9 features.
- The dataset has a target variable that is “Outcome”, which contains 1’s and 0’s.
- Class 1 is a diabetic and class 0 is a non-diabetic.
- The dataset is highly imbalanced, having total 268 diabetic and 500 non-diabetic.

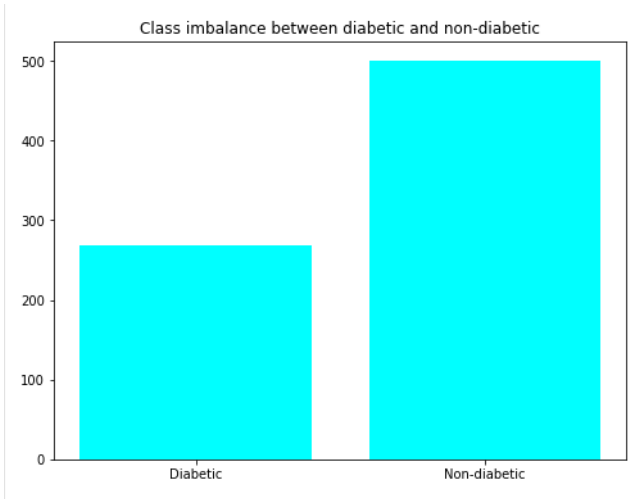


Figure 10. class imbalance

- Furthermore, the datasets are not linear with the target variable.

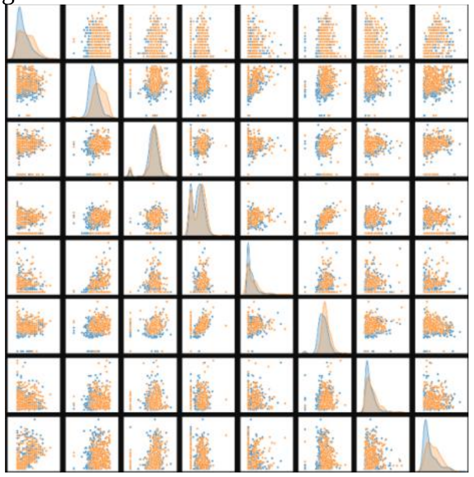


Figure 11.Feature relation with target v

Since the relation with target is not linear, using single layer neural network may pose difficulties in prediction accuracy, because single layer neural network is a linear classifier and the dataset using here using is non-linear. However, dimensional reduction techniques can get the dataset almost linear but we have small datasets and doing so will result in loose of information’s.

Therefore we can remove the missing values from the data.

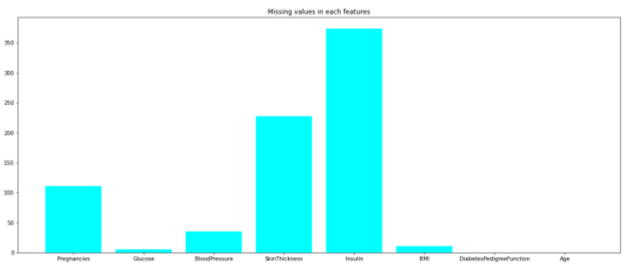


Figure 12 Missing values from each features

Therefore after removing the missing values it is better to use scaler method to shrink the sample values within [0,1] using MinMax scaler method, now we can proceed with the classification model.