



Predicting Diabetes using ANFIS



- Harsh Agarwal 18085027
 - Nitesh Naman Prasad 18085037
- 

Abstract

We studied the ANFIS (Adaptive Network based Fuzzy Inference System) proposed by Jang(1993). We implemented a simple version of ANFIS for Diabetes predictor with Gaussian membership functions and tuned it. Our model had a test accuracy of 81.30%, train accuracy of 85.42%, test F1-Score of 74.72% and test F1-Score of 78.44%. We further analysed the variation of performance metrics and losses with epochs, constructed the confusion matrix, plotted graph of the membership functions learnt by the model and discussed our observations.

Introduction

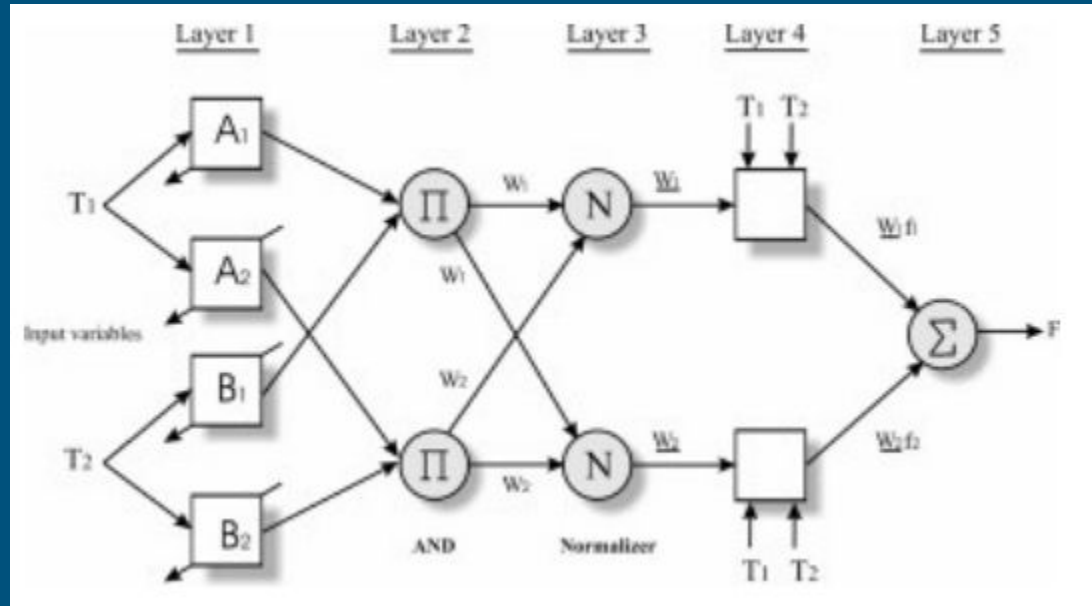
In the recent times, due to availability of large amount of data and high computational power, Artificial Intelligence (AI) and Machine Learning (ML) algorithms play a very important role in automation. In today's world models have been trained rigorously and have achieved high accuracy. But a feature used for classification cannot be a crisp value exactly, i.e., there is some amount of fuzziness involved. To account for this, we use membership function to estimate the relatedness of an element to a set. This feature plays a very important role in identifying unclear examples in the dataset. Fuzzy neural networks (FNN) combine the Fuzzy Logic and Neural networks in single model. We study Adaptive-Neural-network-based Fuzzy Inference System (ANFIS) (Jang, 1993) and implement a model to that predicts whether a person is having diabetes or not.

Model

1. Architecture and Equations for Forward Pass

The main advantage of Adaptive Network based Fuzzy Inference System over other inference system is that the parameters used for its membership functions can be changed. Hence we can apply Machine Learning (ML) algorithms to train these parameters and build a model that fits into the given dataset. Moreover, ANFIS can be stacked with any other Deep Learning models, which makes it more interesting to study and unique from other Fuzzy Neural Networks

ANFIS is made up of five layers as discussed in the following subsections.



Architecture of ANFIS

Input Layer

This contains the input features of the data used by the model. The data has to be standardised before it is being fed to the model. Mathematically, our input data will be a n dimensional vector for each example (data point).

For n input features x_i ($i = 1, 2, \dots, n$),

Input: x_1, x_2, \dots, x_n

Output: x_1, x_2, \dots, x_n

Fuzzification Layer

This layer is made up of mn nodes (where m is the desired number of fuzzy rules and n is the number of input features). For every input feature i , ($i = 1, 2, \dots, n$) there are nodes a_{ij} where $j = 1, 2, \dots, m$. A Gaussian membership function with trainable parameters μ (mean) and σ^2 (variance), is applied to the input features to obtain the output for this layer.

For each node a_{ij} where $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$

Input: x_i

Output: $N_{i1}(x_i), N_{i2}(x_i), \dots, N_{im}(x_i)$

where N_{ij} is the Gaussian membership function with mean μ_{ij} and variance σ_{ij}^2 .

Rule Layer

This layer takes the product (algebraic intersection, \cap_a) of the respective rules obtained in the previous layer. The output of this layer is fed as input for the next layer.

Each node a_j in this layer, takes $x_{1j}, x_{2j}, \dots, x_{nj}$ from the previous layer, where n is the number of features and $j = 1, 2, \dots, m$, where m is the number of fuzzy rules. It gives product of all these values, i.e., $a_j = \prod_{i=1}^n x_{ij}$, as the output which is then normalised.

For each node j ($j = 1, 2, \dots, m$),

Input: $w_{1j}, w_{2j}, \dots, w_{nj}$

$$w_j = w_{1j} \cdot w_{2j} \cdots w_{nj}$$

$$\text{sum} = w_1 + w_2 + \cdots + w_m$$

Output: $w_j^* = w_j / \text{sum}$. Here, w_j^* is the normalized weight.

De-fuzzification layer

It has m nodes which is the number of fuzzy rules. As the name says, this layer is used for defuzzification. Each node takes the corresponding previous node output and the input layer values and does the following $(a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n) \cdot w$ where w is the weight of the corresponding rule layer and $a_0, a_1, a_2, \dots, a_n$ are trainable parameters.

For each j ($j = 1, 2, \dots, m$),

$$\text{Input: } f_j = a_{0j} + a_{1j}x_1 + \dots + a_{nj}x_n$$

$$\text{Output: } w_j^* f_j$$

where a_0, a_1, \dots, a_n are trainable parameters.

Output layer

All the outputs from previous layer is added and an activation function (sigmoid function, in our case) is applied.

For each j ($j = 1, 2, \dots, m$)

$$\text{Input: } X = \sum_{j=1}^m w_j^* f_j$$

$$\text{Output: } \sigma(X)$$

where $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. If $x > 0$, $\sigma(x) > 0.5$, so we assign a label 1 to such an input (and 0 otherwise).

Predicting Diabetes using ANFIS

- Harsh Agarwal 18085027
- Nitesh Naman Prasad 18085037

Abstract

We studied the ANFIS (Adaptive Network based Fuzzy Inference System) proposed by Jang(1993). We implemented a simple version of ANFIS for Diabetes predictor with Gaussian membership functions and tuned it. Our model had a test accuracy of 81.30%, train accuracy of 85.42%, test F1-Score of 74.72% and test F1-Score of 78.44%. We further analysed the variation of performance metrics and losses with epochs, constructed the confusion matrix, plotted graph of the membership functions learnt by the model and discussed our observations.

Introduction

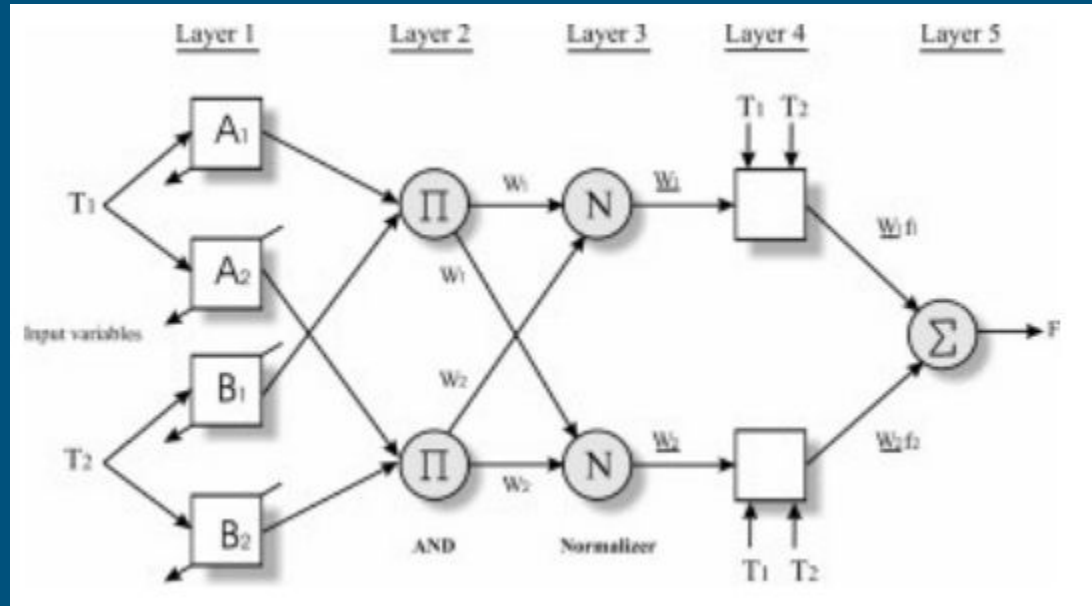
In the recent times, due to availability of large amount of data and high computational power, Artificial Intelligence (AI) and Machine Learning (ML) algorithms play a very important role in automation. In today's world models have been trained rigorously and have achieved high accuracy. But a feature used for classification cannot be a crisp value exactly, i.e., there is some amount of fuzziness involved. To account for this, we use membership function to estimate the relatedness of an element to a set. This feature plays a very important role in identifying unclear examples in the dataset. Fuzzy neural networks (FNN) combine the Fuzzy Logic and Neural networks in single model. We study Adaptive-Neural-network-based Fuzzy Inference System (ANFIS) (Jang, 1993) and implement a model to that predicts whether a person is having diabetes or not.

Model

1. Architecture and Equations for Forward Pass

The main advantage of Adaptive Network based Fuzzy Inference System over other inference system is that the parameters used for its membership functions can be changed. Hence we can apply Machine Learning (ML) algorithms to train these parameters and build a model that fits into the given dataset. Moreover, ANFIS can be stacked with any other Deep Learning models, which makes it more interesting to study and unique from other Fuzzy Neural Networks

ANFIS is made up of five layers as discussed in the following subsections.



Architecture of ANFIS

Input Layer

This contains the input features of the data used by the model. The data has to be standardised before it is being fed to the model. Mathematically, our input data will be a n dimensional vector for each example (data point).

For n input features x_i ($i = 1, 2, \dots, n$),

Input: x_1, x_2, \dots, x_n

Output: x_1, x_2, \dots, x_n

Fuzzification Layer

This layer is made up of mn nodes (where m is the desired number of fuzzy rules and n is the number of input features). For every input feature i , ($i = 1, 2, \dots, n$) there are nodes a_{ij} where $j = 1, 2, \dots, m$. A Gaussian membership function with trainable parameters μ (mean) and σ^2 (variance), is applied to the input features to obtain the output for this layer.

For each node a_{ij} where $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$

Input: x_i

Output: $N_{i1}(x_i), N_{i2}(x_i), \dots, N_{im}(x_i)$

where N_{ij} is the Gaussian membership function with mean μ_{ij} and variance σ_{ij}^2 .

Rule Layer

This layer takes the product (algebraic intersection, \cap_a) of the respective rules obtained in the previous layer. The output of this layer is fed as input for the next layer.

Each node a_j in this layer, takes $x_{1j}, x_{2j}, \dots, x_{nj}$ from the previous layer, where n is the number of features and $j = 1, 2, \dots, m$, where m is the number of fuzzy rules. It gives product of all these values, i.e., $a_j = \prod_{i=1}^n x_{ij}$, as the output which is then normalised.

For each node j ($j = 1, 2, \dots, m$),

Input: $w_{1j}, w_{2j}, \dots, w_{nj}$

$$w_j = w_{1j} \cdot w_{2j} \cdots w_{nj}$$

$$\text{sum} = w_1 + w_2 + \cdots + w_m$$

Output: $w_j^* = w_j / \text{sum}$. Here, w_j^* is the normalized weight.

De-fuzzification layer

It has m nodes which is the number of fuzzy rules. As the name says, this layer is used for defuzzification. Each node takes the corresponding previous node output and the input layer values and does the following $(a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n) \cdot w$ where w is the weight of the corresponding rule layer and $a_0, a_1, a_2, \dots, a_n$ are trainable parameters.

For each j ($j = 1, 2, \dots, m$),

$$\text{Input: } f_j = a_{0j} + a_{1j}x_1 + \dots + a_{nj}x_n$$

$$\text{Output: } w_j^* f_j$$

where a_0, a_1, \dots, a_n are trainable parameters.

Output layer

All the outputs from previous layer is added and an activation function (sigmoid function, in our case) is applied.

For each j ($j = 1, 2, \dots, m$)

$$\text{Input: } X = \sum_{j=1}^m w_j^* f_j$$

$$\text{Output: } \sigma(X)$$

where $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. If $x > 0$, $\sigma(x) > 0.5$, so we assign a label 1 to such an input (and 0 otherwise).

2. Membership Functions

Gaussian membership function is the widely used membership function. Yet, other membership functions like Generalized Bell membership function, Triangular fuzzy number, Trapezoidal fuzzy number, etc., can also be applied. However, it is recommended to choose a function with smooth curves, so that they are differentiable (helpful for optimizer while training), and it also helps us to learn nonlinear functions. Given below is Gaussian Membership Function we used in the Fuzzification Layer,

$$N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$


where μ is the mean and σ^2 is the variance.

Experiment



Dataset

We use the "[Pima Indians Diabetes Database](#)" from Kaggle. The dataset contains 768 examples and 8 features (plus one outcome). It is worthy to note that this dataset only consists of females who are at least 21 years old.

| Feature | Short Description |
|--|--|
| <i>Pregnancies</i> | Number of times pregnant |
| <i>Blood Pressure</i> | Plasma glucose concentration a 2 hours in an oral glucose tolerance test |
| <i>Blood Pressure</i> | Diastolic blood pressure (mm Hg) |
| <i>Skin Thickness</i> | Triceps skin fold thickness (mm) |
| <i>Insulin</i> | 2-hour serum insulin (mu U/ml) |
| <i>BMI</i> | Body mass index (weight in kg/(height in m) ²) |
| <i>Diabetes Pedigree Function</i> | Diabetes pedigree function |
| <i>Age</i>  | Age (years) |
| <i>Outcome</i> | Class variable (0 or 1) |

Pre-processing

We split the dataset into train and test data using `sklearn.model_selection.train_test_split()`. The model requires each feature to have a mean 0 and standard deviation 1. Hence, we standardize each column of the dataset before using it.

It was observed that the dataset did not have any missing data-so no additional processing were necessary.

Settings

We conducted our experiment with the following hyperparameters:

- We used `sklearn.model_selection.train_test_split()` with `test_split()` set to 0.16. Moreover, `random_state` was set to 2021.
 - Total number of fuzzy rules used in the model was set to 14.
 - Binary Cross Entropy was the loss function used.
 - Adam Optimizer was used with learning parameter $\alpha = 0.01$ to optimize the loss function in 1000 epochs.
 - Data points for plotting graphs were taken at every 10 epochs.
 - Gaussian membership function was used for each fuzzy rule. The weights, μ , σ were randomly initialized from normal distribution.
 - We omit the Pregnancies column from the dataset, as it doesn't have a direct relation with the disease of interest.
-

Results

We use the following notations to define the performance metrics we use in this report:

- TP (True Positive): Model predicts Positive and it is actually Positive.
- FP (False Positive): Model predicts Positive but it is actually Negative.
- FN (False Negative): Model predicts Negative but it is actually Positive.
- TN (True Negative): Model predicts Negative and it is actually Negative.

Now, we define the performance metrics of our interest.

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$$

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$F1 \text{ Score} = \frac{2*Precision*Recall}{Precision+Recall}$$



Predicting Diabetes using ANFIS



- Harsh Agarwal 18085027
 - Nitesh Naman Prasad 18085037
- 

Abstract

We studied the ANFIS (Adaptive Network based Fuzzy Inference System) proposed by Jang(1993). We implemented a simple version of ANFIS for Diabetes predictor with Gaussian membership functions and tuned it. Our model had a test accuracy of 81.30%, train accuracy of 85.42%, test F1-Score of 74.72% and test F1-Score of 78.44%. We further analysed the variation of performance metrics and losses with epochs, constructed the confusion matrix, plotted graph of the membership functions learnt by the model and discussed our observations.

Introduction

In the recent times, due to availability of large amount of data and high computational power, Artificial Intelligence (AI) and Machine Learning (ML) algorithms play a very important role in automation. In today's world models have been trained rigorously and have achieved high accuracy. But a feature used for classification cannot be a crisp value exactly, i.e., there is some amount of fuzziness involved. To account for this, we use membership function to estimate the relatedness of an element to a set. This feature plays a very important role in identifying unclear examples in the dataset. Fuzzy neural networks (FNN) combine the Fuzzy Logic and Neural networks in single model. We study Adaptive-Neural-network-based Fuzzy Inference System (ANFIS) (Jang, 1993) and implement a model to that predicts whether a person is having diabetes or not.

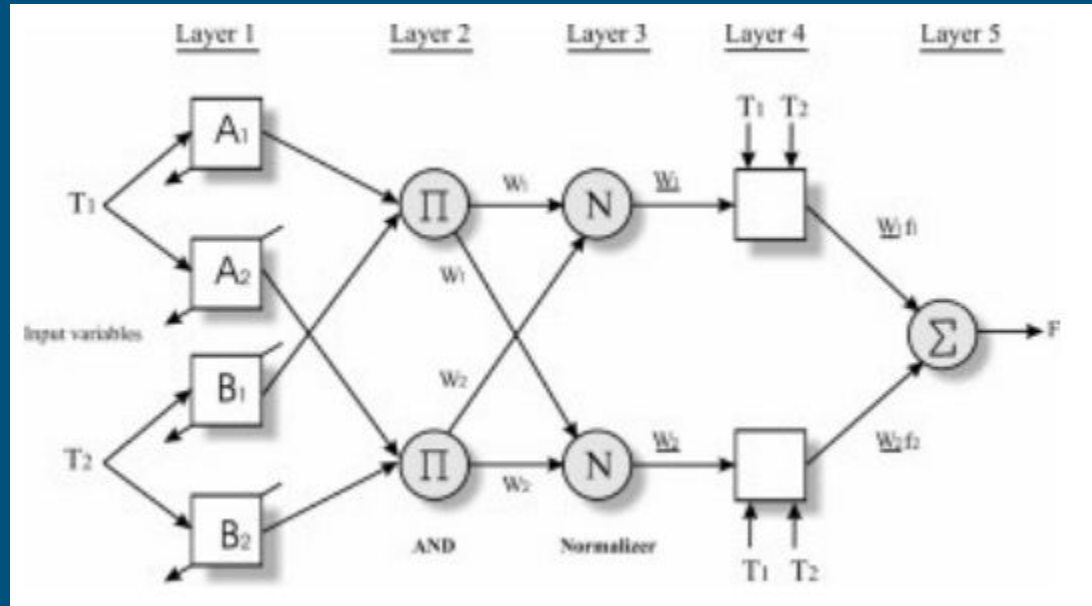
Model



1. Architecture and Equations for Forward Pass

The main advantage of Adaptive Network based Fuzzy Inference System over other inference system is that the parameters used for its membership functions can be changed. Hence we can apply Machine Learning (ML) algorithms to train these parameters and build a model that fits into the given dataset. Moreover, ANFIS can be stacked with any other Deep Learning models, which makes it more interesting to study and unique from other Fuzzy Neural Networks

ANFIS is made up of five layers as discussed in the following subsections.



Architecture of ANFIS

Input Layer

This contains the input features of the data used by the model. The data has to be standardised before it is being fed to the model. Mathematically, our input data will be a n dimensional vector for each example (data point).

For n input features x_i ($i = 1, 2, \dots, n$),

Input: x_1, x_2, \dots, x_n

Output: x_1, x_2, \dots, x_n

Fuzzification Layer

This layer is made up of mn nodes (where m is the desired number of fuzzy rules and n is the number of input features). For every input feature i , ($i = 1, 2, \dots, n$) there are nodes a_{ij} where $j = 1, 2, \dots, m$. A Gaussian membership function with trainable parameters μ (mean) and σ^2 (variance), is applied to the input features to obtain the output for this layer.

For each node a_{ij} where $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$

Input: x_i

Output: $N_{i1}(x_i), N_{i2}(x_i), \dots, N_{im}(x_i)$

where N_{ij} is the Gaussian membership function with mean μ_{ij} and variance σ_{ij}^2 .

Rule Layer

This layer takes the product (algebraic intersection, \cap_a) of the respective rules obtained in the previous layer. The output of this layer is fed as input for the next layer.

Each node a_j in this layer, takes $x_{1j}, x_{2j}, \dots, x_{nj}$ from the previous layer, where n is the number of features and $j = 1, 2, \dots, m$, where m is the number of fuzzy rules. It gives product of all these values, i.e., $a_j = \prod_{i=1}^n x_{ij}$, as the output which is then normalised.

For each node j ($j = 1, 2, \dots, m$),

Input: $w_{1j}, w_{2j}, \dots, w_{nj}$

$$w_j = w_{1j} \cdot w_{2j} \cdots w_{nj}$$

$$\text{sum} = w_1 + w_2 + \cdots + w_m$$

Output: $w_j^* = w_j / \text{sum}$. Here, w_j^* is the normalized weight.

De-fuzzification layer

It has m nodes which is the number of fuzzy rules. As the name says, this layer is used for defuzzification. Each node takes the corresponding previous node output and the input layer values and does the following $(a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n) \cdot w$ where w is the weight of the corresponding rule layer and $a_0, a_1, a_2, \dots, a_n$ are trainable parameters.

For each j ($j = 1, 2, \dots, m$),

$$\text{Input: } f_j = a_{0j} + a_{1j}x_1 + \dots + a_{nj}x_n$$

$$\text{Output: } w_j^* f_j$$

where a_0, a_1, \dots, a_n are trainable parameters.

Output layer

All the outputs from previous layer is added and an activation function (sigmoid function, in our case) is applied.

For each j ($j = 1, 2, \dots, m$)

$$\text{Input: } X = \sum_{j=1}^m w_j^* f_j$$

$$\text{Output: } \sigma(X)$$

where $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. If $x > 0$, $\sigma(x) > 0.5$, so we assign a label 1 to such an input (and 0 otherwise).

2. Membership Functions

Gaussian membership function is the widely used membership function. Yet, other membership functions like Generalized Bell membership function, Triangular fuzzy number, Trapezoidal fuzzy number, etc., can also be applied. However, it is recommended to choose a function with smooth curves, so that they are differentiable (helpful for optimizer while training), and it also helps us to learn nonlinear functions. Given below is Gaussian Membership Function we used in the Fuzzification Layer,

$$N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$


where μ is the mean and σ^2 is the variance.

Experiment



Dataset

We use the "[Pima Indians Diabetes Database](#)" from Kaggle. The dataset contains 768 examples and 8 features (plus one outcome). It is worthy to note that this dataset only consists of females who are at least 21 years old.

| Feature | Short Description |
|--|--|
| <i>Pregnancies</i> | Number of times pregnant |
| <i>Blood Pressure</i> | Plasma glucose concentration a 2 hours in an oral glucose tolerance test |
| <i>Blood Pressure</i> | Diastolic blood pressure (mm Hg) |
| <i>Skin Thickness</i> | Triceps skin fold thickness (mm) |
| <i>Insulin</i> | 2-hour serum insulin (mu U/ml) |
| <i>BMI</i> | Body mass index (weight in kg/(height in m) ²) |
| <i>Diabetes Pedigree Function</i> | Diabetes pedigree function |
| <i>Age</i>  | Age (years) |
| <i>Outcome</i> | Class variable (0 or 1) |

Pre-processing

We split the dataset into train and test data using `sklearn.model_selection.train_test_split()`. The model requires each feature to have a mean 0 and standard deviation 1. Hence, we standardize each column of the dataset before using it.

It was observed that the dataset did not have any missing data-so no additional processing were necessary.

Settings

We conducted our experiment with the following hyperparameters:

- We used `sklearn.model_selection.train_test_split()` with `test_split()` set to 0.16. Moreover, `random_state` was set to 2021.
 - Total number of fuzzy rules used in the model was set to 14.
 - Binary Cross Entropy was the loss function used.
 - Adam Optimizer was used with learning parameter $\alpha = 0.01$ to optimize the loss function in 1000 epochs.
 - Data points for plotting graphs were taken at every 10 epochs.
 - Gaussian membership function was used for each fuzzy rule. The weights, μ , σ were randomly initialized from normal distribution.
 - We omit the Pregnancies column from the dataset, as it doesn't have a direct relation with the disease of interest.
-

Results

We use the following notations to define the performance metrics we use in this report:

- TP (True Positive): Model predicts Positive and it is actually Positive.
- FP (False Positive): Model predicts Positive but it is actually Negative.
- FN (False Negative): Model predicts Negative but it is actually Positive.
- TN (True Negative): Model predicts Negative and it is actually Negative.

Now, we define the performance metrics of our interest.

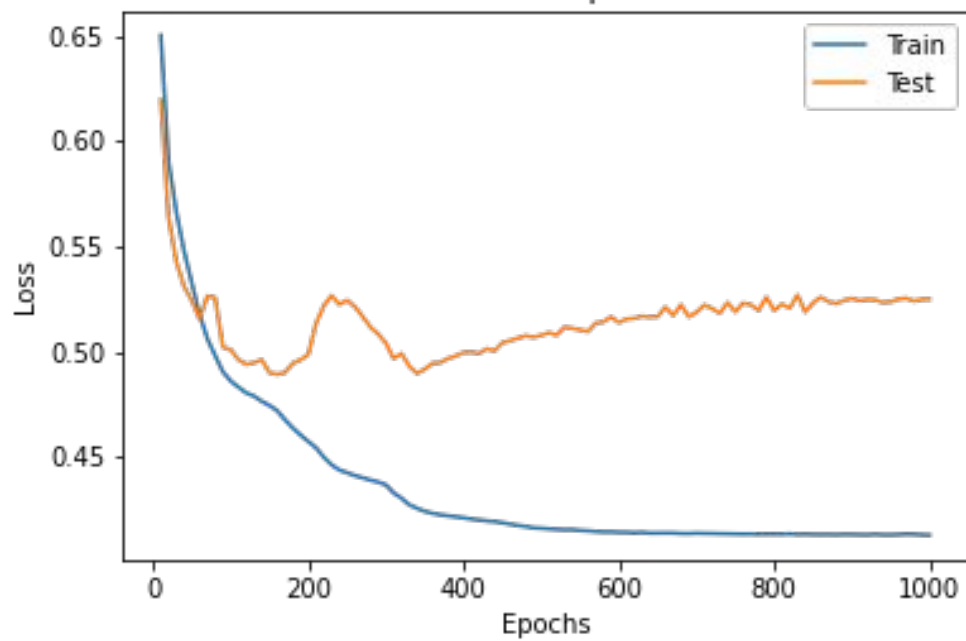
$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$$

$$Precision = \frac{TP}{TP+FP}$$

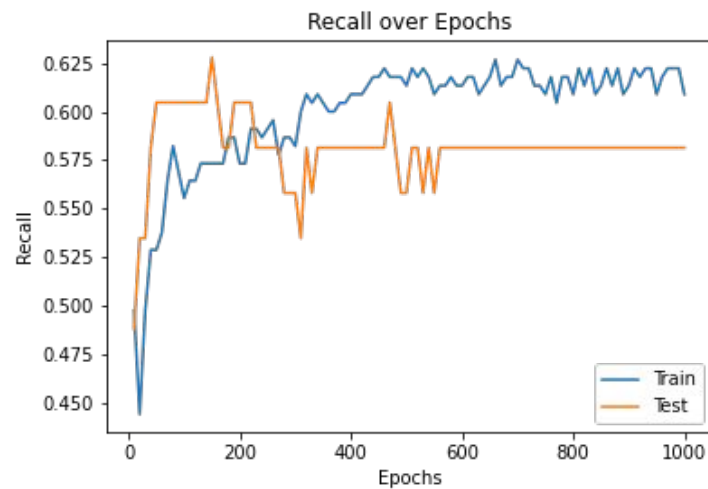
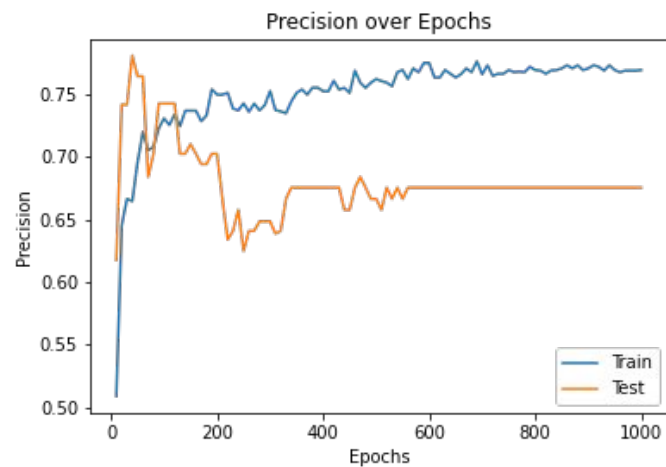
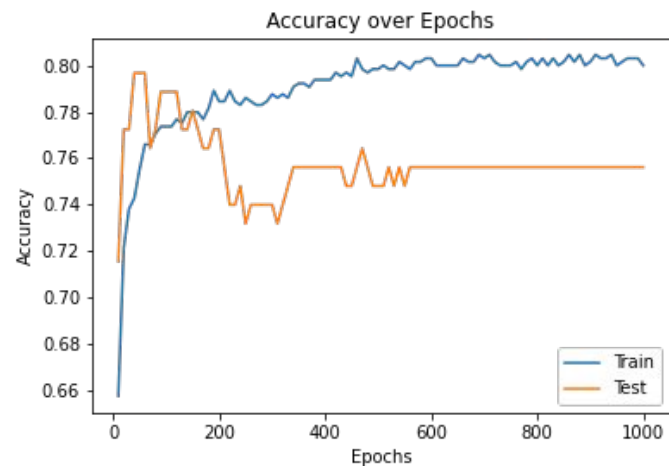
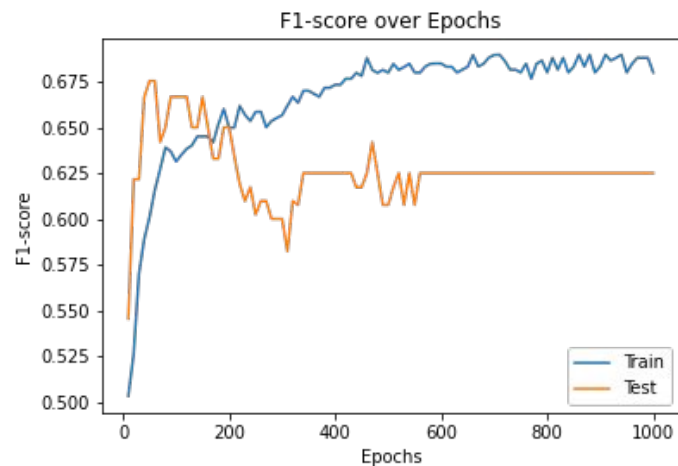
$$Recall = \frac{TP}{TP+FN}$$

$$F1 \text{ Score} = \frac{2*Precision*Recall}{Precision+Recall}$$

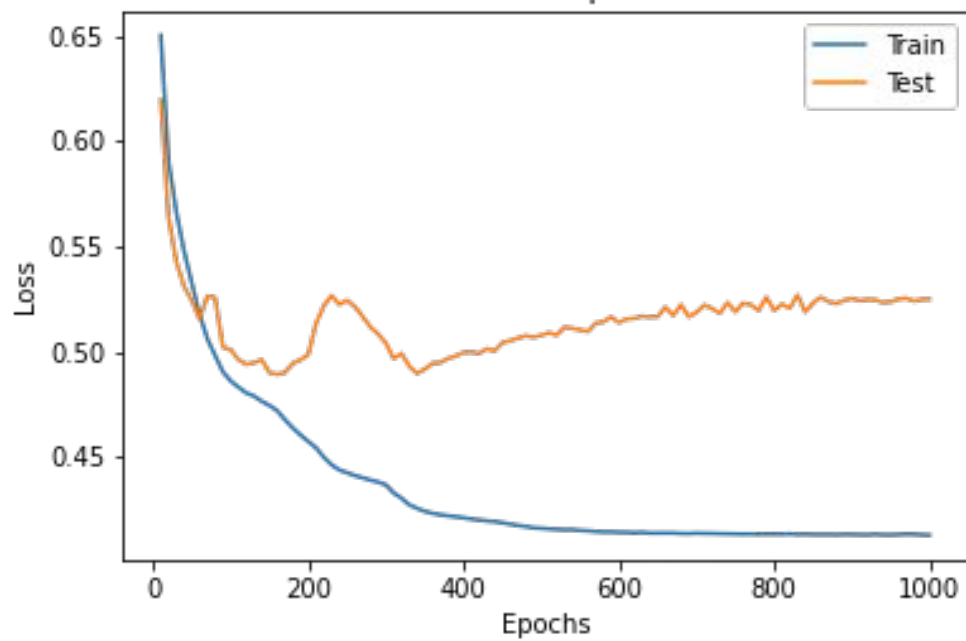
Loss over Epochs



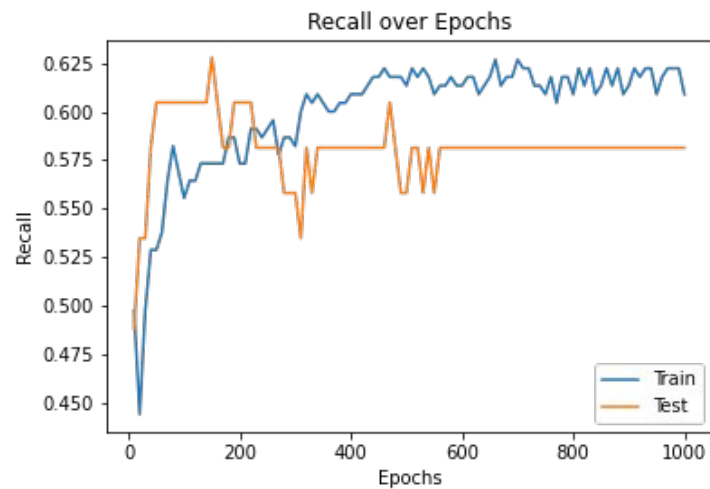
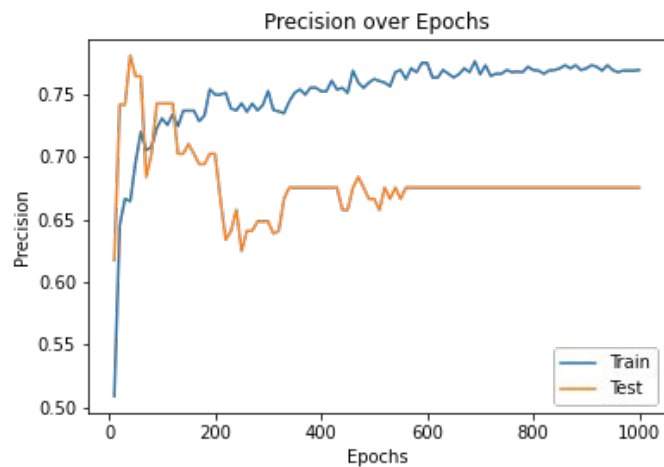
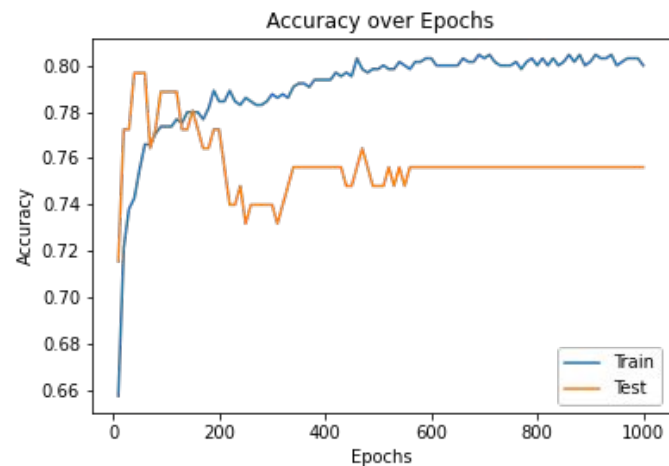
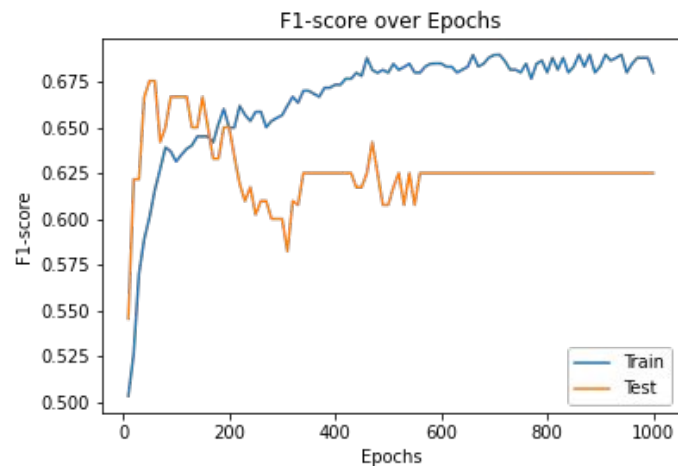
Performance Metrics:



Loss over Epochs



Performance Metrics:



2. Membership Functions

Gaussian membership function is the widely used membership function. Yet, other membership functions like Generalized Bell membership function, Triangular fuzzy number, Trapezoidal fuzzy number, etc., can also be applied. However, it is recommended to choose a function with smooth curves, so that they are differentiable (helpful for optimizer while training), and it also helps us to learn nonlinear functions. Given below is Gaussian Membership Function we used in the Fuzzification Layer,

$$N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$


where μ is the mean and σ^2 is the variance.

Experiment



Dataset

We use the "[Pima Indians Diabetes Database](#)" from Kaggle. The dataset contains 768 examples and 8 features (plus one outcome). It is worthy to note that this dataset only consists of females who are at least 21 years old.

| Feature | Short Description |
|--|--|
| <i>Pregnancies</i> | Number of times pregnant |
| <i>Blood Pressure</i> | Plasma glucose concentration a 2 hours in an oral glucose tolerance test |
| <i>Blood Pressure</i> | Diastolic blood pressure (mm Hg) |
| <i>Skin Thickness</i> | Triceps skin fold thickness (mm) |
| <i>Insulin</i> | 2-hour serum insulin (mu U/ml) |
| <i>BMI</i> | Body mass index (weight in kg/(height in m) ²) |
| <i>Diabetes Pedigree Function</i> | Diabetes pedigree function |
| <i>Age</i>  | Age (years) |
| <i>Outcome</i> | Class variable (0 or 1) |

Pre-processing

We split the dataset into train and test data using `sklearn.model_selection.train_test_split()`. The model requires each feature to have a mean 0 and standard deviation 1. Hence, we standardize each column of the dataset before using it.

It was observed that the dataset did not have any missing data-so no additional processing were necessary.

Settings

We conducted our experiment with the following hyperparameters:

- We used `sklearn.model_selection.train_test_split()` with `test_split()` set to 0.16. Moreover, `random_state` was set to 2021.
 - Total number of fuzzy rules used in the model was set to 14.
 - Binary Cross Entropy was the loss function used.
 - Adam Optimizer was used with learning parameter $\alpha = 0.01$ to optimize the loss function in 1000 epochs.
 - Data points for plotting graphs were taken at every 10 epochs.
 - Gaussian membership function was used for each fuzzy rule. The weights, μ , σ were randomly initialized from normal distribution.
 - We omit the Pregnancies column from the dataset, as it doesn't have a direct relation with the disease of interest.
-

Results

We use the following notations to define the performance metrics we use in this report:

- TP (True Positive): Model predicts Positive and it is actually Positive.
- FP (False Positive): Model predicts Positive but it is actually Negative.
- FN (False Negative): Model predicts Negative but it is actually Positive.
- TN (True Negative): Model predicts Negative and it is actually Negative.

Now, we define the performance metrics of our interest.

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$$

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

$$F1\ Score = \frac{2*Precision*Recall}{Precision+Recall}$$

Predicting Diabetes using ANFIS

- Harsh Agarwal 18085027
- Nitesh Naman Prasad 18085037

Abstract

We studied the ANFIS (Adaptive Network based Fuzzy Inference System) proposed by Jang(1993). We implemented a simple version of ANFIS for Diabetes predictor with Gaussian membership functions and tuned it. Our model had a test accuracy of 81.30%, train accuracy of 85.42%, test F1-Score of 74.72% and test F1-Score of 78.44%. We further analysed the variation of performance metrics and losses with epochs, constructed the confusion matrix, plotted graph of the membership functions learnt by the model and discussed our observations.

Introduction

In the recent times, due to availability of large amount of data and high computational power, Artificial Intelligence (AI) and Machine Learning (ML) algorithms play a very important role in automation. In today's world models have been trained rigorously and have achieved high accuracy. But a feature used for classification cannot be a crisp value exactly, i.e., there is some amount of fuzziness involved. To account for this, we use membership function to estimate the relatedness of an element to a set. This feature plays a very important role in identifying unclear examples in the dataset. Fuzzy neural networks (FNN) combine the Fuzzy Logic and Neural networks in single model. We study Adaptive-Network-based Fuzzy Inference System (ANFIS) (Jang, 1993) and implement a model to that predicts whether a person is having diabetes or not.

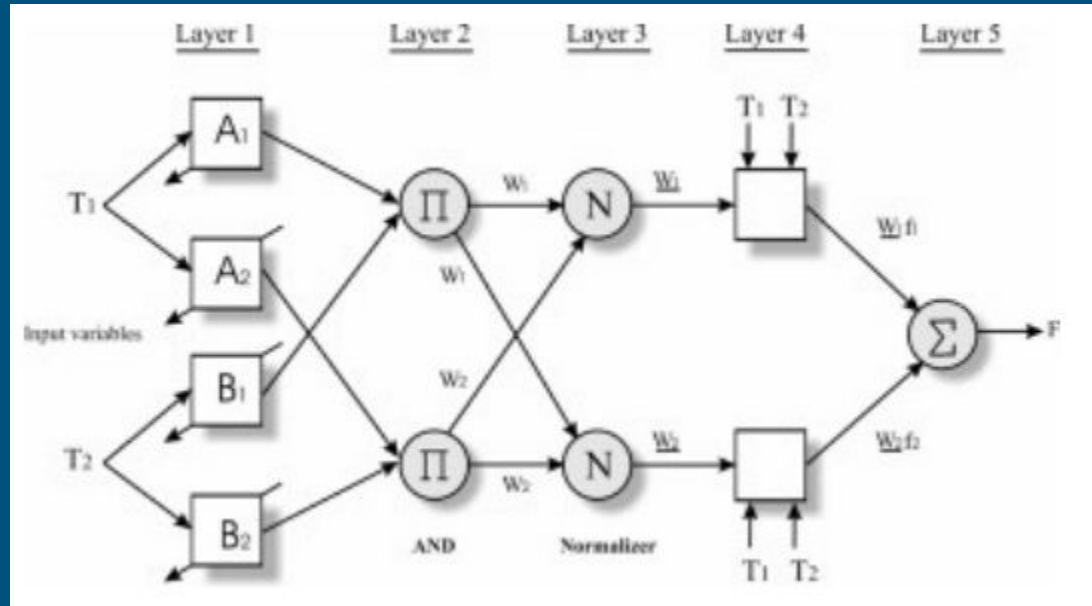
Model



1. Architecture and Equations for Forward Pass

The main advantage of Adaptive Network based Fuzzy Inference System over other inference system is that the parameters used for its membership functions can be changed. Hence we can apply Machine Learning (ML) algorithms to train these parameters and build a model that fits into the given dataset. Moreover, ANFIS can be stacked with any other Deep Learning models, which makes it more interesting to study and unique from other Fuzzy Neural Networks

ANFIS is made up of five layers as discussed in the following subsections.



Architecture of ANFIS

Input Layer

This contains the input features of the data used by the model. The data has to be standardised before it is being fed to the model. Mathematically, our input data will be a n dimensional vector for each example (data point).

For n input features x_i ($i = 1, 2, \dots, n$),

Input: x_1, x_2, \dots, x_n

Output: x_1, x_2, \dots, x_n

Fuzzification Layer

This layer is made up of mn nodes (where m is the desired number of fuzzy rules and n is the number of input features). For every input feature i , ($i = 1, 2, \dots, n$) there are nodes a_{ij} where $j = 1, 2, \dots, m$. A Gaussian membership function with trainable parameters μ (mean) and σ^2 (variance), is applied to the input features to obtain the output for this layer.

For each node a_{ij} where $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, m$

Input: x_i

Output: $N_{i1}(x_i), N_{i2}(x_i), \dots, N_{im}(x_i)$

where N_{ij} is the Gaussian membership function with mean μ_{ij} and variance σ_{ij}^2 .

Rule Layer

This layer takes the product (algebraic intersection, \cap_a) of the respective rules obtained in the previous layer. The output of this layer is fed as input for the next layer.

Each node a_j in this layer, takes $x_{1j}, x_{2j}, \dots, x_{nj}$ from the previous layer, where n is the number of features and $j = 1, 2, \dots, m$, where m is the number of fuzzy rules. It gives product of all these values, i.e., $a_j = \prod_{i=1}^n x_{ij}$, as the output which is then normalised.

For each node j ($j = 1, 2, \dots, m$),

Input: $w_{1j}, w_{2j}, \dots, w_{nj}$

$$w_j = w_{1j} \cdot w_{2j} \cdots w_{nj}$$

$$\text{sum} = w_1 + w_2 + \cdots + w_m$$

Output: $w_j^* = w_j / \text{sum}$. Here, w_j^* is the normalized weight.

De-fuzzification layer

It has m nodes which is the number of fuzzy rules. As the name says, this layer is used for defuzzification. Each node takes the corresponding previous node output and the input layer values and does the following $(a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n) \cdot w$ where w is the weight of the corresponding rule layer and $a_0, a_1, a_2, \dots, a_n$ are trainable parameters.

For each j ($j = 1, 2, \dots, m$),

$$\text{Input: } f_j = a_{0j} + a_{1j}x_1 + \dots + a_{nj}x_n$$

$$\text{Output: } w_j^* f_j$$

where a_0, a_1, \dots, a_n are trainable parameters.

Output layer

All the outputs from previous layer is added and an activation function (sigmoid function, in our case) is applied.

For each j ($j = 1, 2, \dots, m$)

$$\text{Input: } X = \sum_{j=1}^m w_j^* f_j$$

$$\text{Output: } \sigma(X)$$

where $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. If $x > 0$, $\sigma(x) > 0.5$, so we assign a label 1 to such an input (and 0 otherwise).

2. Membership Functions

Gaussian membership function is the widely used membership function. Yet, other membership functions like Generalized Bell membership function, Triangular fuzzy number, Trapezoidal fuzzy number, etc., can also be applied. However, it is recommended to choose a function with smooth curves, so that they are differentiable (helpful for optimizer while training), and it also helps us to learn nonlinear functions. Given below is Gaussian Membership Function we used in the Fuzzification Layer,

$$N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$


where μ is the mean and σ^2 is the variance.

Experiment



Dataset

We use the "[Pima Indians Diabetes Database](#)" from Kaggle. The dataset contains 768 examples and 8 features (plus one outcome). It is worthy to note that this dataset only consists of females who are at least 21 years old.

| Feature | Short Description |
|--|--|
| <i>Pregnancies</i> | Number of times pregnant |
| <i>Blood Pressure</i> | Plasma glucose concentration a 2 hours in an oral glucose tolerance test |
| <i>Blood Pressure</i> | Diastolic blood pressure (mm Hg) |
| <i>Skin Thickness</i> | Triceps skin fold thickness (mm) |
| <i>Insulin</i> | 2-hour serum insulin (mu U/ml) |
| <i>BMI</i> | Body mass index (weight in kg/(height in m) ²) |
| <i>Diabetes Pedigree Function</i> | Diabetes pedigree function |
| <i>Age</i>  | Age (years) |
| <i>Outcome</i> | Class variable (0 or 1) |

Pre-processing

We split the dataset into train and test data using `sklearn.model_selection.train_test_split()`. The model requires each feature to have a mean 0 and standard deviation 1. Hence, we standardize each column of the dataset before using it.

It was observed that the dataset did not have any missing data-so no additional processing were necessary.

Settings

We conducted our experiment with the following hyperparameters:

- We used `sklearn.model_selection.train_test_split()` with `test_split()` set to 0.16. Moreover, `random_state` was set to 2021.
 - Total number of fuzzy rules used in the model was set to 14.
 - Binary Cross Entropy was the loss function used.
 - Adam Optimizer was used with learning parameter $\alpha = 0.01$ to optimize the loss function in 1000 epochs.
 - Data points for plotting graphs were taken at every 10 epochs.
 - Gaussian membership function was used for each fuzzy rule. The weights, μ , σ were randomly initialized from normal distribution.
 - We omit the Pregnancies column from the dataset, as it doesn't have a direct relation with the disease of interest.
-

Results

We use the following notations to define the performance metrics we use in this report:

- TP (True Positive): Model predicts Positive and it is actually Positive.
- FP (False Positive): Model predicts Positive but it is actually Negative.
- FN (False Negative): Model predicts Negative but it is actually Positive.
- TN (True Negative): Model predicts Negative and it is actually Negative.

Now, we define the performance metrics of our interest.

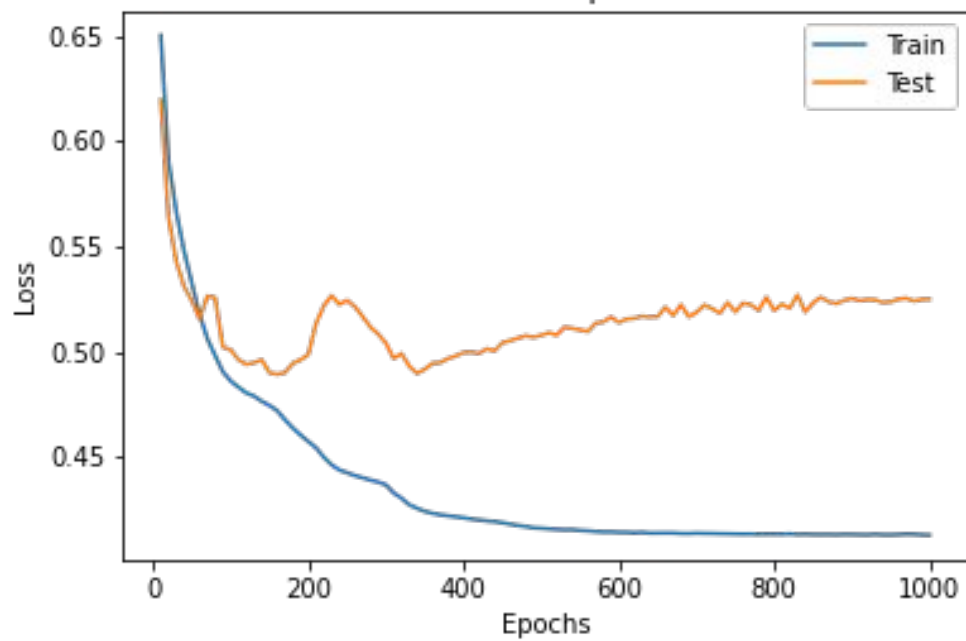
$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$$

$$Precision = \frac{TP}{TP+FP}$$

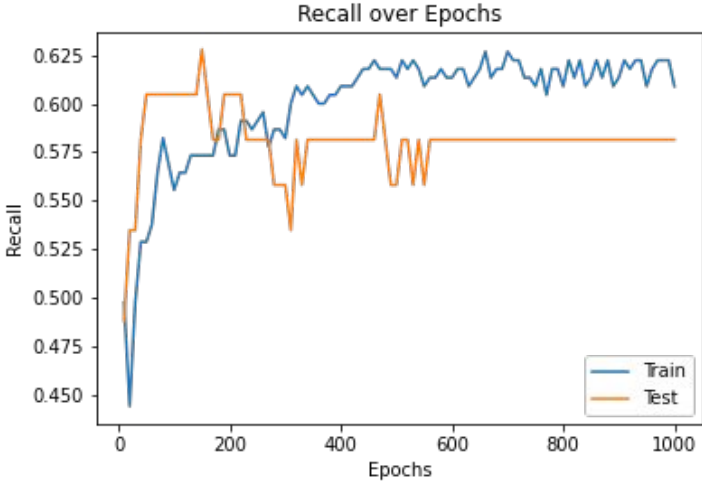
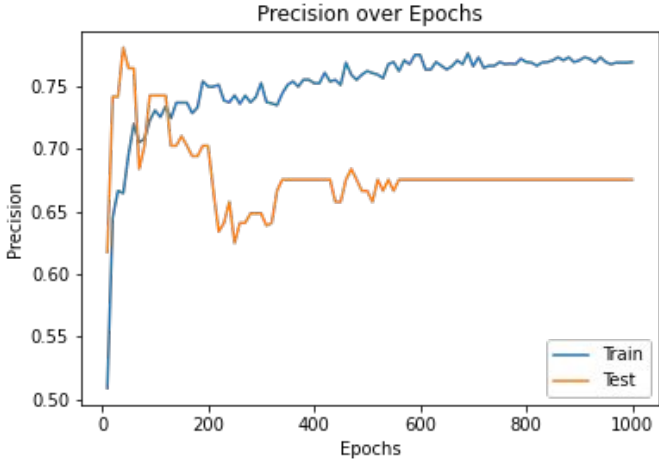
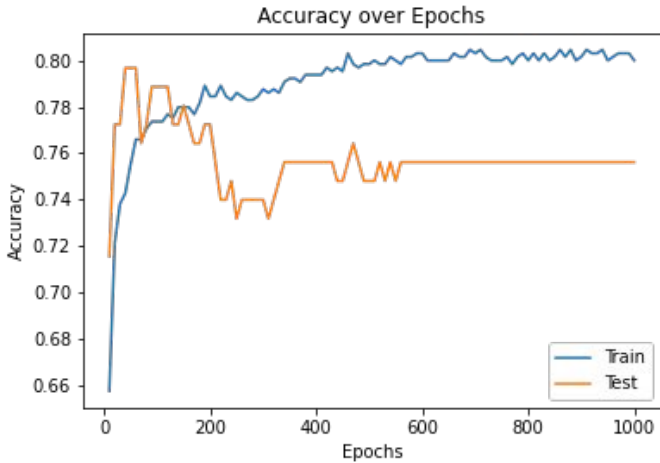
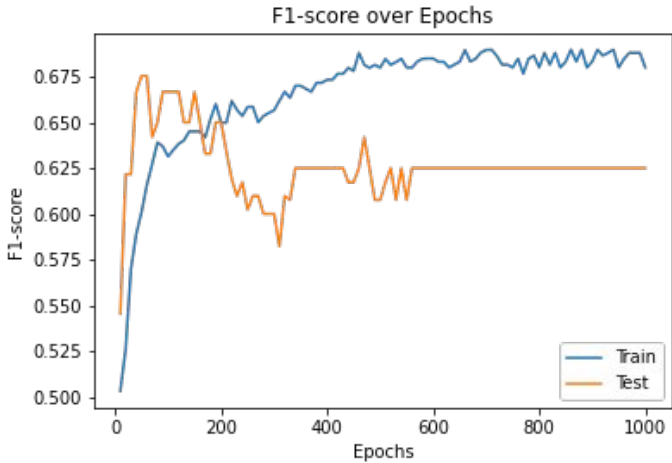
$$Recall = \frac{TP}{TP+FN}$$

$$F1 \text{ Score} = \frac{2*Precision*Recall}{Precision+Recall}$$

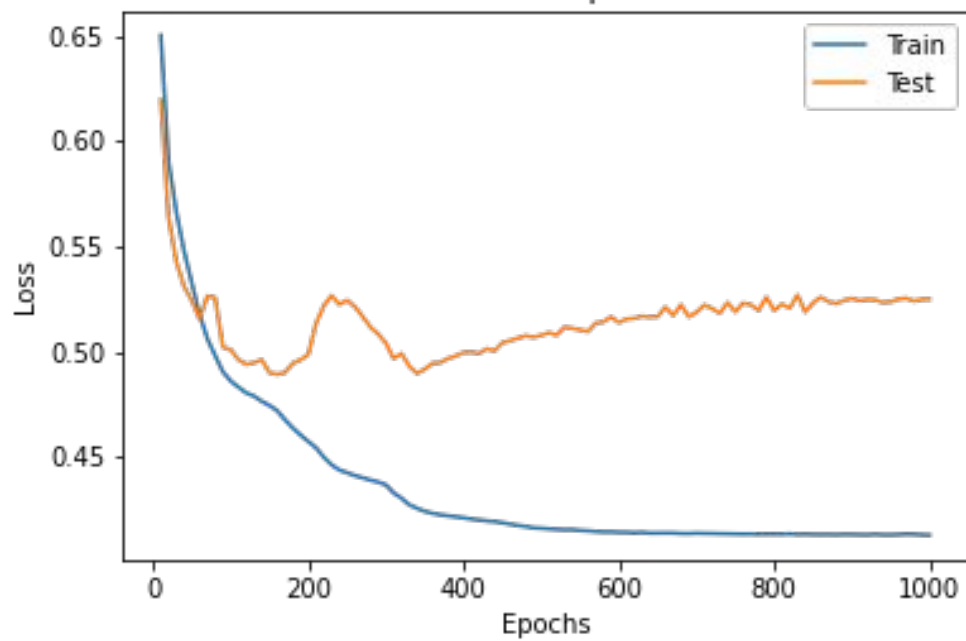
Loss over Epochs



Performance Metrics:



Loss over Epochs



Performance Metrics:

