Artificial Intelligence Neural Networks Training in the classification model

A Neural Network data model by training the classification model of a breast cancer prediction output in MATLAB.

Michail M, Chaidos I, Manios A
COMPUTER SCIENCE YEAR 3 ARTIFICIAL INTELLIGENCE

SCHOOL OF ARCHITECTURE, COMPUTING & ENGINEERING

BSc in Computer Science

Michail Markou, Chaidos John, Manios Athanasios

CN6005 – ARTIFICIAL INTELLIGENCE

UEL NUMBER

2020732 (MM), 1966133 (CJ), 2020737 (MA)

Date

11/12/2021

Contents

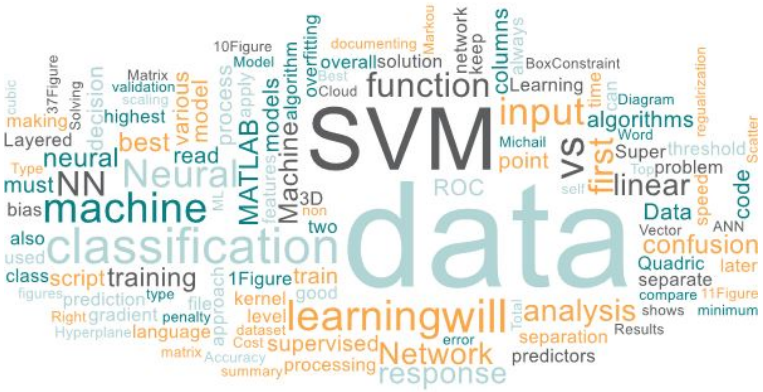


Figure 1 Word Cloud

1. Abstract

We will investigate a case study from Wisconsin for breast cancer data and analyze various variables to determine if breast tissue is malignant or benign. Through the analysis process, we will explore various techniques, such as SVM and neural networks, to determine the result of this set of classification data and which methods offer the highest degree of accuracy in determining our original performing algorithmic supervised machine learning approaches as previously reported in MATLAB.

As we will see in the first run from the Classification Learner which also provides automatic grade point analysis of each algorithm approach at acceptable levels, the SVM has the best algorithms that ranks first for scoring and in particular Cubic / Quadratic has the highest accuracy with 95, 2% while in second place followed by the Neural Networks with the highest score 94.4% of a score prediction based on the classification of trained data for the classification of test data. So clearly the Square / Cubic SVM with 10 predictors is the best algorithms for classifying our data.[1]

We will teach the learning machine[2] learn from the data by building a model / representation based on them [1] and selecting some features[3] from the samples, statistically we will conclude in which class / category the inputs are classified (trying to avoid bias in the ANN system), analyzing the entire pipeline / workflow from raw data to the final model of supervised machine learning forecast for finished production and drawing conclusions, predictions and categorizations.

2. Machine Learning Algorithms

2.1. Description of input Data Set

In the coursework the data analysis for ranking will use data from the cancer patients dividing it into benign and malignant. In data science and automation as there is always some software with an interface

[1] Each run produces different accuracy but overall SVM is better than Neural Networks here.

[2] Machine learning is an old terminology that has existed in everyday life for decades (but at the consumer ready level it has been for the last five years) now as we rely more and more on it because it can analyze speed and see what we cannot see because of huge volume of data (big data). You would say you are doing another form of life (in time and in the end). Usages in fintech, ad-tech, social-media, overall science, and Software as a Service (SaaS or anything-as-a-service) embedded like spam filter and email categorization or language grammar support and sentence prediction et al.

[3] Aka feature engineering heuristic is divided into 4 categories, the (1) one is when someone thinks about them & introduces them (supervised labeled data) (kitchen sink approach)) (Unlabeled data, infer structure out of the data where you project the data in a space and look for clusters) where the features are learned and "inserted" by the machine itself (how do you understand the distance between points in space plays big role because it will judge if the points are common to each other or not). Reinforcement learning (3) -decision points - system kitchen self-repetition (kitchen sink approach - brute force in learning) that succeeds and plays on its own based on the rules (a game e.g., robot broom sweeper). Deep Learning (4) when we do not have as our first choice the understanding (he has his own reputation of "things" as an entity - another mind) how something happens but we will invest later in another system on top of that for clarification is more advanced than all techniques with the flaw of providing huge computing resources and data to learn. The feature selection is inspired by the rubber duck debugging technique of software engineering on page 37

(API) that enters the data to be processed later tends to require strict criteria for raw data to have consistency in the flow and processing of this data but and avoid errors about their variations.

Our Database input is in the repository URL kaggle_wisconsin_cancer_data which data come in the form of comma separate values (csv) of 569 patient samples and 32 columns of which 30 are typical medical data of patients with numeric scientific notation and follow the US / UK decimal and group separation notation (“,” <- thousands, “.” <- decimal point / full stop).

As there must be a universal consistency before processing this data because there must always be the rationale of future respective imports for the process of their automation in the following data depending on how we handle them we must choose the appropriate clearance. The first 2 columns are considered a separate reason in that the patient's id is not directly related to analysis statistics as we want an overall picture in the extraction of results so at this stage, we delete it or keep it in a separate file / variable for later processing.

Our program will be in the form of script language MATLAB will read on the spot the file will not be used manual way to edit and convert csv to excel or some ready import wizard GUI in MATLAB so in the first phase we read the whole file as it is and delete the first column holding the second because it is the response to our predictors (it is the target in our inputs) but also the Limitation of our script in the analysis (natural language process e.g., GPT-3 language model) NLP characteristics and classification because it is string / text will turn it into numeric data, something it can "translate". It is also good from the beginning but later we can do the so-called feature selection, i.e. from this medical data which needs to be analyzed in order to make an assessment of the classification of future corresponding problems through neural network training giving us new output to unknown input , in the present problem we keep the first 10 columns from the so-called features or otherwise from the original data input unprocessed we keep the columns 3-13 for features (predictors) and the 2 for labeled / target / response (not output we got after training) .

```
% Read Wisconsin Breast Cancer database
% import Data
clear; clc; close all;

% first col is decimal second is string then the rest 29 is features and
throw/delete headeline first row of csv text
fid = fopen('data.csv','r');
Data = textscan(fid, ['%d,%[^,],',repmat('%f,',1,29),'%f\n'],
'HeaderLines',1);
fclose(fid);

n = length(Data);       % number of features + target
n = n-2;                % ignore first two attributes/Features (ID of
patient, target of cancer Type)
P = length(Data{1});    % number of patterns/samples

%% prepare Data sets
x = zeros(P,n);
for pat=1:P
    for i=1:n
        x(pat,i) = double(Data{i+2}(pat)); % fill row by row entering Data
cell variable
    end
end

t = zeros(1,P); % targets
for i=1:P
    if char(Data{2}(i)) == 'M'
        t(i) = 1; % Malignant
    else
        t(i) = 0; %Benign
    end
end
t = t'; % transpose in order for proper dimensions inputs & targets to
process later
```

Figure 2 read input dataset self-documenting code in MATLAB

2.2. Problem Solving & Data Analysis

The solution of a problem first presupposes the evaluation analysis at a high abstract level of its peculiarities in order to make a correct approach by choosing the most efficient or acceptable solution always depending on the cost-time. Two more common forms of "applying supervised machine learning" are the algorithms / techniques Super vector machine & Neural Networks where they belong to the large category of models: regression (when the response is continuous) & classification (when the response belongs to a set of classes) and apply to data where these algorithms generate models that record trends in datasets: `machine-learning-models`

Below we analyze these 2 categories of machine learning models that are accessible to problems where a model is created that predicts a response to them (predict a response).

2.2.1. Super Vector Machine (SVM) - fitcsvm

An SVM is a learning machine that sorts data by finding the linear decision boundary (hyperplane) that separates all data points of one class from those of another class in 2D or 3D by creating a circle if they cannot be separated in two-dimensional level (below directly for this). The best sublevel/hyperplane for

an SVM is the one with the largest margin between the two margins, when the data is linearly separable. If the data is not separated linearly, a loss function is used to punish points on the wrong side of the superlevel. SVMs sometimes use a kernel transformation to convert nonlinearly separable data to higher dimensions where a linear resolution threshold can be found. Supports out-of-the-box binary classification and with some modification the binary classification medium You approach multiclass-classification when it has more than 2 different data. Used for image / face / text recognition and spam filter et al.

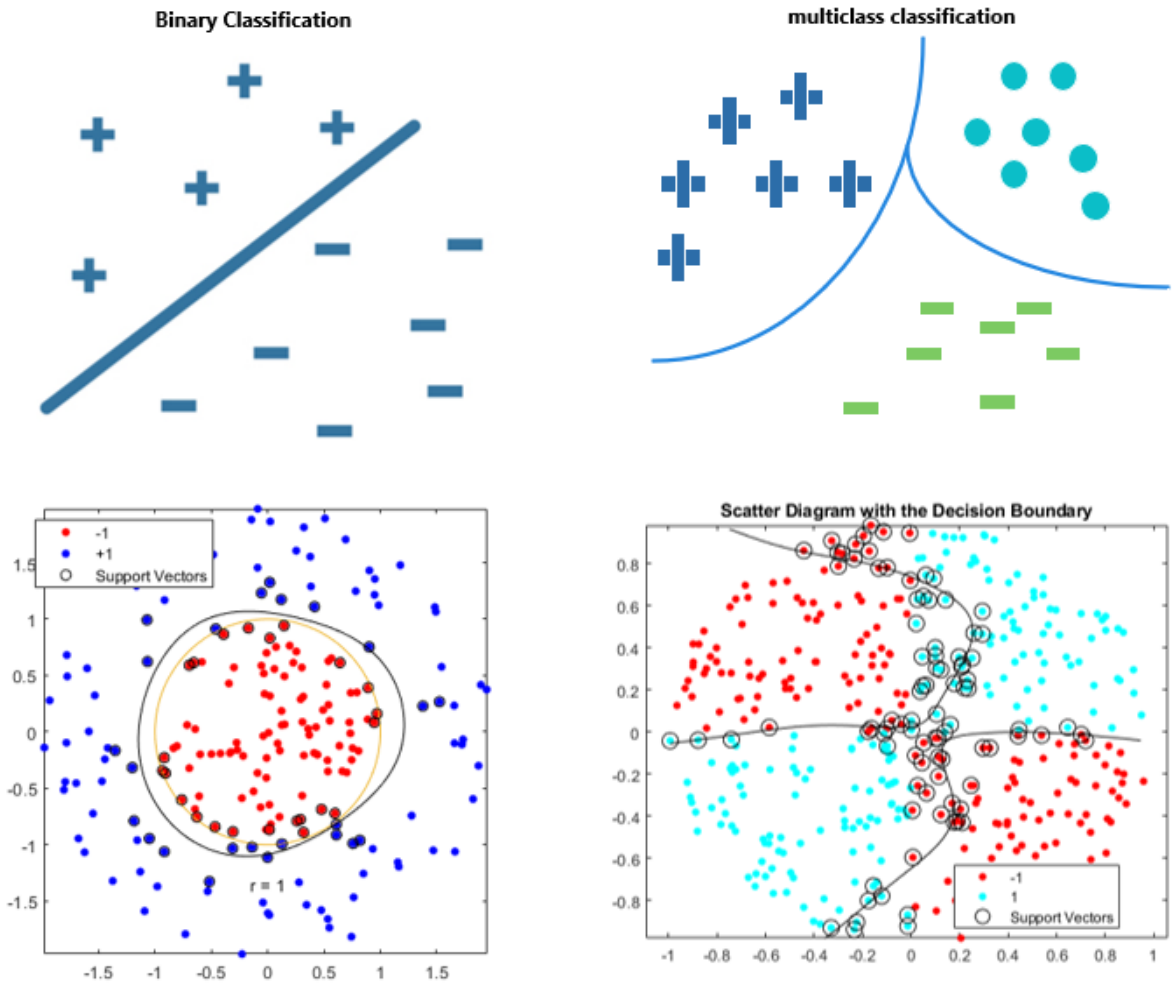


Figure 3 SVM algorithm type models

2.2.1.1. Code

```
% Popular Machine Learning Models for Classification
% Support Vector Machine (SVM) category
% https://www.mathworks.com/discovery/machine-learning-models.html

% 1 preparing dataset, dividing it train/test set - (features (columns)
samples/records/id's/inputs/classes(rows) predictors (cols + rows),
targets/responses/classes/labels/outputs for division and classification)
% 2 preparing validation set out of training set (kfold cv (cross
validation))
% 3 Feature selection
% 4 fiding best parameters (hyper)
% 5 test the model with test set
% 6 visualize hyperplane

clear; close all; clc; % if you want to "re-train" comment out this line

%% preparing dataset

load wdbc.mat

targets_num = t; % responses
%%

% binary classification 569 samples to make into shape...
%X = zeros(569,30);
X = x; % 569 samples (rows) 30 features (columns)
y = targets_num(1:569);

%%
% 80:20 Data categorization
rand_num = randperm(size(X,1)); % 569 samples
X_train = X(rand_num(1:round(0.8*length(rand_num))),:); % 80
y_train = y(rand_num(1:round(0.8*length(rand_num))),:); % target

X_test = X(rand_num(round(0.8*length(rand_num))+1:end),:); % 20 ... 81:end-
to-100 of samples
y_test = y(rand_num(round(0.8*length(rand_num))+1:end),:); % target
%% CV partition

c = cvpartition(y_train,'k',5);
%% feature selection show only 2 random features out of features.size
% below we display this categories in plot output by mapping textFeature name
headerline from csv to columns of the predicted output graph

opts = statset('display','iter');
classf = @(train_data, train_labels, test_data, test_labels)...
    sum(predict(fitcsvm(train_data, train_labels,'KernelFunction','rbf'),
test_data) ~= test_labels);

[fs, history] = sequentialfs(classf, X_train, y_train, 'cv', c, 'options',
opts,'nfeatures',2);
%% Best seperation hyperparameter MSE - weight(x) kernelScale - bias(y)
BoxConstraint
% https://www.mathworks.com/help/stats/support-vector-machines-for-binary-
classification.html
```

```
X_train_w_best_feature = X_train(:,fs);

Md1 =
fitcsvm(X_train_w_best_feature,y_train,'KernelFunction','rbf','OptimizeHyperp
arameters','auto',...

'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName',...
      'expected-improvement-plus','ShowPlots',true)); % Bayes' Optimization
use.


%% Final test with test set
X_test_w_best_feature = X_test(:,fs);
test_accuracy_for_iter = sum((predict(Md1,X_test_w_best_feature) ==
y_test))/length(y_test)*569; %100

%% hyperplane validation draw output

figure;
hgscatter =
gscatter(X_train_w_best_feature(:,1),X_train_w_best_feature(:,2),y_train); %
features values X_train_w_best_feature 1 col and col 2 compare
hold on;
h_sv=plot(Md1.SupportVectors(:,1),Md1.SupportVectors(:,2),'ko','markersize',8
);
gscatter_malignant_group = hgscatter(2);
gscatter_malignant_group.Color = 'r';
X_label_matrix='';
X_flag_exit = 0;
Y_label_matrix='';
Y_flag_exit = 0;
searchMappingColumns0=0;
searchMappingRowsY0=0;
searchMappingColumns1=0;
% map data from X_train_w_best_feature to textFeatures
% X_train_w_best_feature{1}(1); only if there is cell like json array
% X_train_w_best_feature{2}(1);
for searchMappingColumns=1:30
    for searchMappingRowsY=1:569
        %disp(X_train_w_best_feature(1:1,1:1));
        %disp(X_train_w_best_feature(1:1,2:2));
        if
(x(searchMappingRowsY:searchMappingRowsY,searchMappingColumns:searchMappingCo
lumns) == X_train_w_best_feature(1:1,1:1) && Y_flag_exit == 0) % row from 1
to 1 and column from 1 to 1
            Y_label_matrix = textFeatures(searchMappingColumns);
            searchMappingColumns0=searchMappingColumns;
            searchMappingRowsY0 = searchMappingRowsY;
            Y_flag_exit = 1;
            %disp(Y_label_matrix);
            %break;
        end
    end
    for searchMappingRowsX=1:569
        %disp(X_train_w_best_feature(1:2));
```

```
        if
        (x(searchMappingRowsX:searchMappingRowsX,searchMappingColumns:searchMappingCo
        lumns) == X_train_w_best_feature(1:1,2:2) && X_flag_exit == 0) % row from 1
        to 1 and column from 2 to 2
                    % do not label different predictor feature but same value
        resulting in same axes name, after first iteration of for Y
                    % we want to search same row record with same column feature of
        same patient
                    % or to make sure that a record has a number but this number
                    % presents multilpe times inside the data set we do a compare of
        not same column to
                    % check again but with any of the other columns left to check
                    % (e.g., same number in multiple columns but second
                    % feature/column appears in another column not same record,
        different row yet
                    % different record)
                    if (searchMappingColumns0 ~= searchMappingColumns ||
        searchMappingRowsX == searchMappingRowsY0)%searchMappingColumns0 ~=
        searchMappingColumns && searchMappingRowsX == searchMappingRowsY0
                        X_label_matrix = textFeatures(searchMappingColumns);
                        X_flag_exit = 1;
                        %disp(X_label_matrix);
                    %else
                        %continue;
                    end
                end
            end
            if (Y_flag_exit == 1 && X_flag_exit == 1)
                break;
            end
        end
        gscatter_malignant_group.Parent.XLabel.String = X_label_matrix;
        gscatter_malignant_group.Parent.YLabel.String = Y_label_matrix;
        gscatter_benign_group = hgscatter(1);
        gscatter_benign_group.Color = 'b';

        % test set of data put them one by one.

        gscatter(X_test_w_best_feature(:,1),X_test_w_best_feature(:,2),y_test,'rb','x
        x')

        % decision plane
        XLIMs = get(gca,'xlim');
        YLIMs = get(gca,'ylim');
        [xi,yi] = meshgrid([XLIMs(1):0.01:XLIMs(2)],[YLIMs(1):0.01:YLIMs(2)]);
        dd = [xi(:), yi(:)];
        pred_mesh = predict(Md1, dd);
        redcolor = [1, 0.8, 0.8];
        bluecolor = [0.8, 0.8, 1];
        pos = find(pred_mesh == 1);
        h1 = plot(dd(pos,1),
        dd(pos,2),'s','color',redcolor,'Markersize',5,'MarkerEdgeColor',redcolor,'Mar
        kerFaceColor',redcolor);
        pos = find(pred_mesh == 2);

        h2 = plot(dd(pos,1),
        dd(pos,2),'s','color',bluecolor,'Markersize',5,'MarkerEdgeColor',bluecolor,'M
        arkerFaceColor',bluecolor);
        uistack(h1,'bottom');
        uistack(h2,'bottom');
        legend([hgscatter;h_sv],{'Benign','Malignant','support vectors'})
        hold off
```

2.2.1.2. *Figures - Plots*

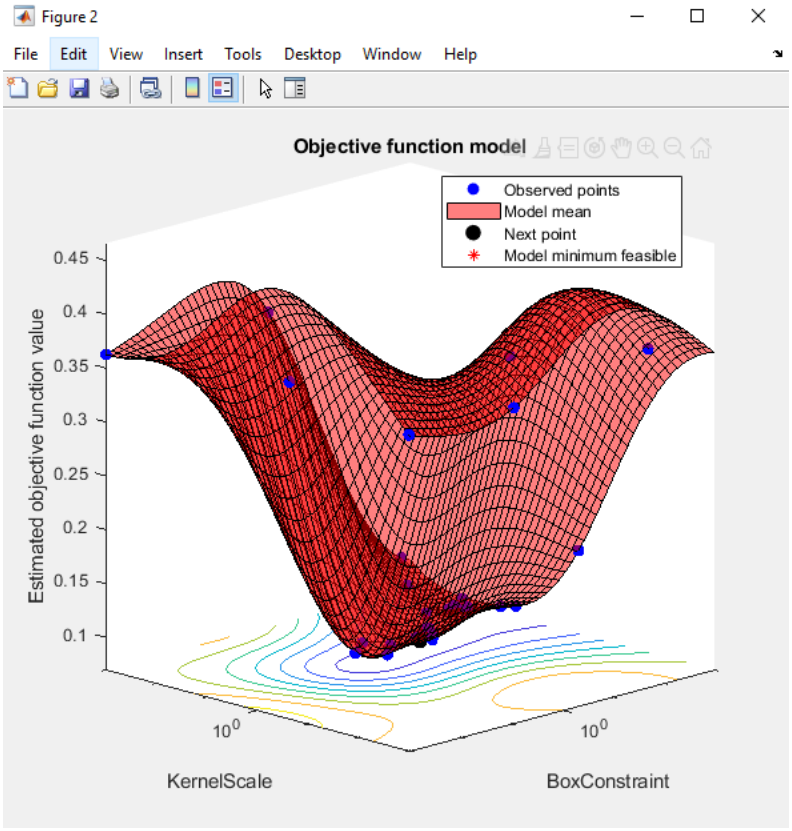
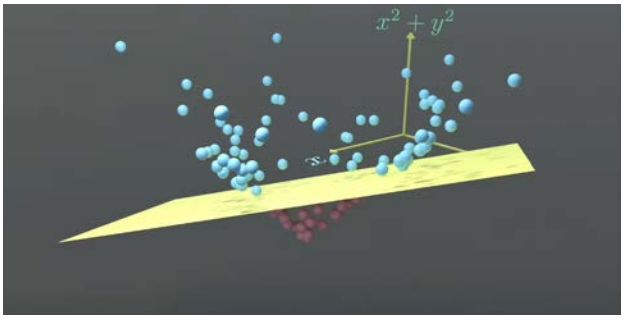
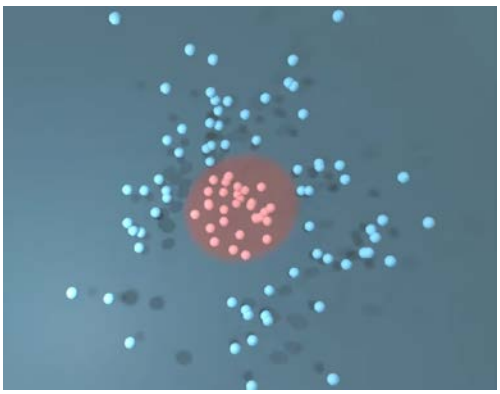
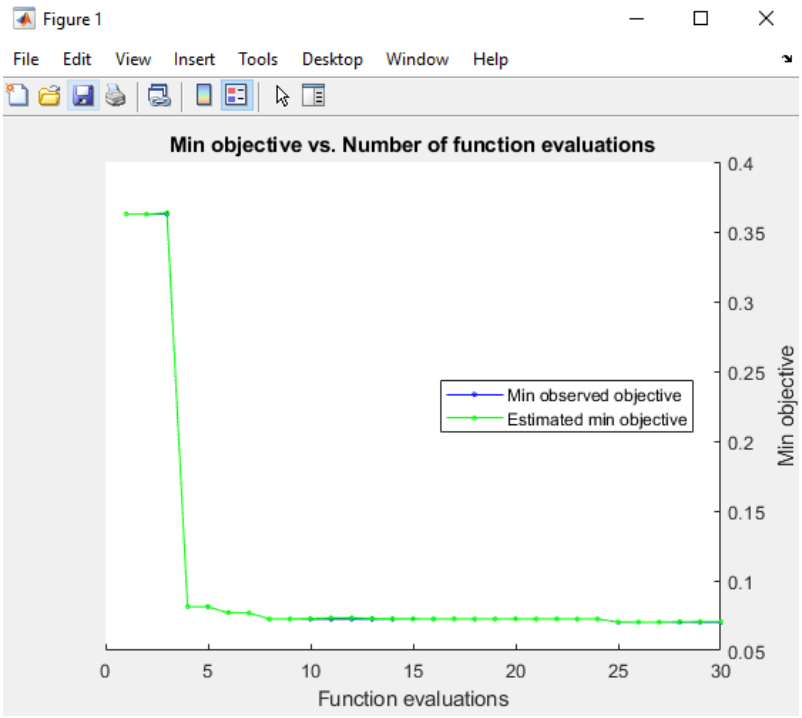


Figure 4 Hyperplane for 3D non-linear separation by scaling (before apply the data to kernel function e.g., cubic or linear) and BoxConstraint for overfitting(or regualrization) and penalty the data [1] [2]







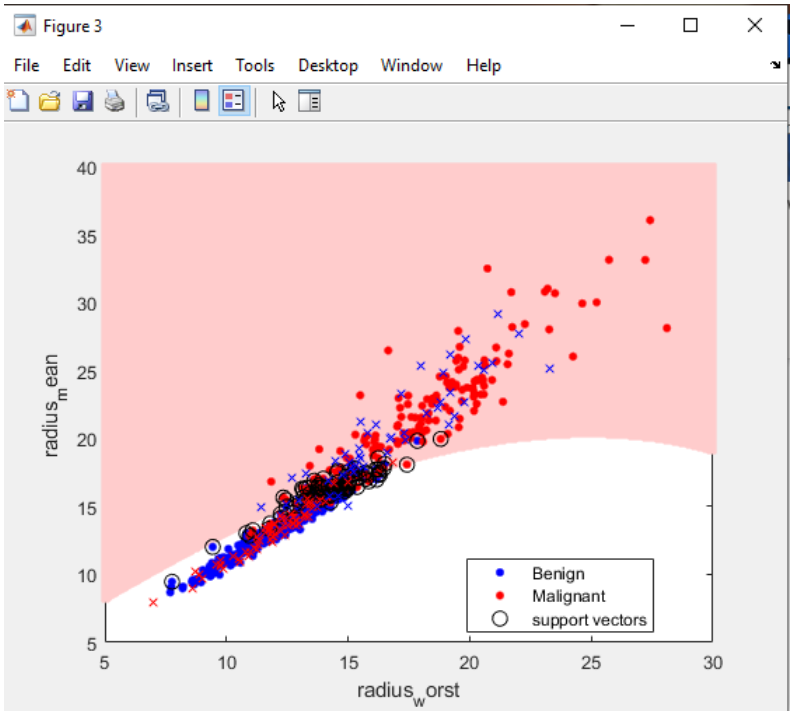


Figure 5 Scatter Diagram of B-M Types

2.2.1.3. Comparison SVM vs NN

Control + scroll mouse in for zoom-in or save image as

Figure 6 SVM vs Neural Network on various train time, Accuracy (validation), Total Cost, prediction speed, Model Type info

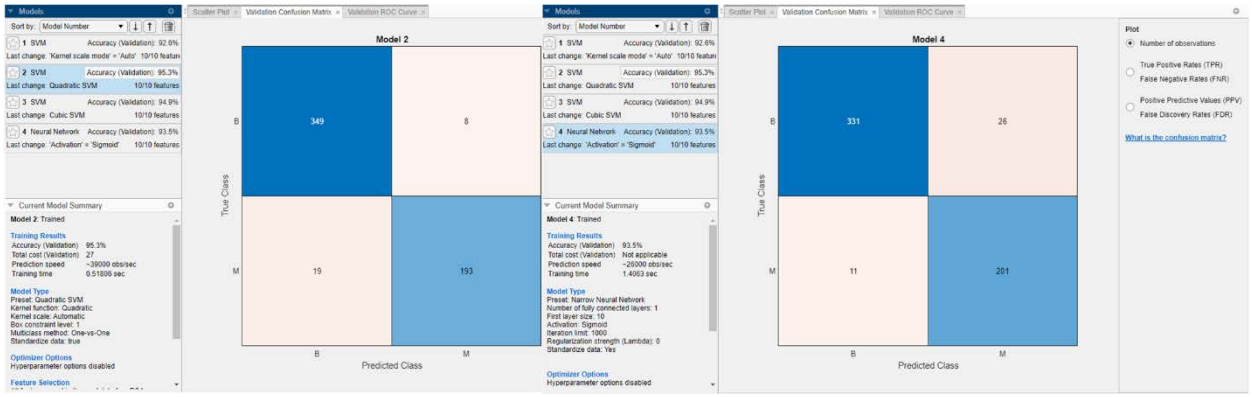


Figure 7 SVM vs NN confusion Matrix

Figure 8 SVM vs NN ROC compare shows Top Right Best Results of summary overall confusion matrix for decision making on point (Quadric SVM)

Quadric SVM as Average is the best compared to NN for this classification problem

2.2.2. Artificial Layered Neural Network (ANN)

Inspired by the human brain, a neural network consists of highly connected neural networks (with possibly hidden intermediates between input layer + output layer) that connect the inputs to the desired outputs. The network is trained by repeatedly modifying (repetition is the mother of learning) the strong points of the connections, so that the training inputs correspond to the training responses. As a result, the correct distribution of new unknown data from new inputs based on the trained network model.

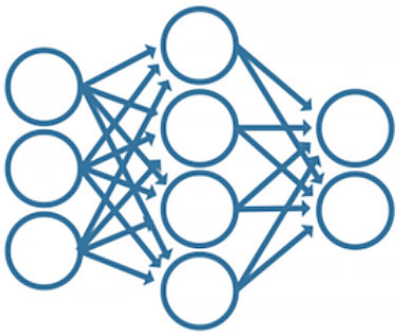


Figure 9 Layered NN

The workflow / pipeline step designs of a neural network are 4 (we can verify this from "*nnstart*" || "*classificationLearner*" steps)

1. Gathering Data
2. prepare data in proper format (predictors & responses) by feature selecting and keeping same matrix dimensions[4]
3. Categorize Data (Train, Validation and Test)[5]
4. Choosing a ML model
    a. Network Architecture by assigning Hidden layers and Neuros per layer
5. Evaluation of Train Network based on input model and visualize or text read the outputs[6]
    a. With/without validation data
    b. Feature selection
6. Finding best hyperparameters by tuning
7. Step 4
8. Re-train and test the model with test set[7]
9. Prediction & visualize final reports in plots and hyperplanes

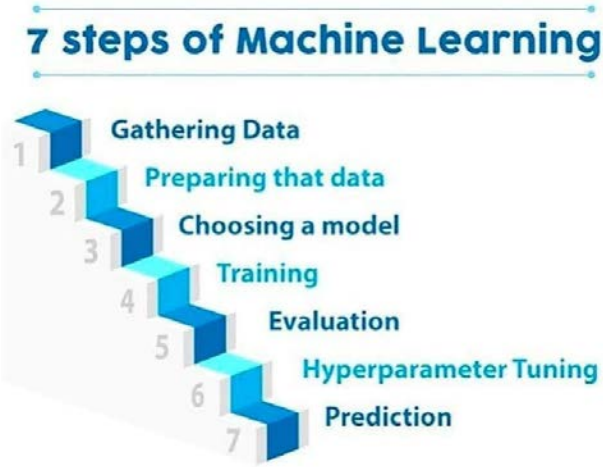


Figure 10 steps of Machine Learning (ML)

[4] Usually on columns (depends on implementation) e.g., 10x569 input, 1x569 target/output

[5] Prepare validation set out of training set (e.g., kfold CV (cross-validation)) that protects against overfitting by partitioning the data set into folds and estimating accuracy of each fold for small data sets or holdout validation for bigger data sets or no validation data to begin with for a NN testing purposes of how its designed in its core before stepping the actual intense tests.

[6] Usually, a harmony of the visual output line parameters and not far apart should be consistent among plots

[7] Keep an average score in a holder place for comparing the best effect

2.2.2.1. Training-Test-Validation

Perhaps the most important part before the initialization of neural network parameters because it will give us Insights how the network is designed but also if the data sets are enough to appear correctly or process the training to evaluate the results and compare them with testing[8].

The process starts by randomly selecting (for the 3 categories random selection) a set of total data by separating them (in our example 569 samples) i.e., applying on them a set of mathematical functions / algorithms for the 3 categories of data and starting the training from random order at the Cartesian level through the algorithm implementations of the network due to the software / framework use[9]:

Train

The data (random subset of the total) in which the network will be "trained" to produce results. That is, based on the predictors / inputs in responses / targets, how much error there is and has managed to reduce the algorithm by changing each complete repetition of samples for per epoch weights and bias (for adjustment to reduce error) to be considered "trained" to produces with new data correct categorization based on what he has already learned by distinguishing patterns through the repetition of entering the same data over and over again and comparing them with the targets, that is, what he should get.

An example of a well-trained network is to look at the diagram (so-called training curve, errors (y logarithmic axes) in terms of number of iterations (x integer numeric axes)) mean square error (mse[10]) for train data (blue line should shrink/to bottom right/approach Figure 11) with a mathematical approach: mse = (target– output)^2 per sample output in terms of the amounts of epochs (iterations) have run the data for training leaking it. In our example per 569 (adding all the mse to a total_mse) mse calculations are divided by how many times the neural network ran by re-entering the same data and comparing with the outputs e.g., sum_mse_of_input / epochs (e.g., $x = \frac{sum\_mse}{10}$).

Test

A random subset of the total data used to check how good is the neural network training based on the train data set. It is usually the most critical curve because it applies this comparison / prediction to new data (not seen in training) to make the network useful for classification.

It is symbolized by a red line (hence the color Figure 11) and should be as parallel as possible to the line of train data if it increases there is a possibility of a possible neural network, i.e., overfitting [3] losing the concept of generalization and this only affects new data not trained or high (y) local minimum (without validation data this becomes more pronounced during training).

[8] That is, the train categorizes based on what it has seen (pattern recognition classification) while the test ranks based on what the instructor has not seen, i.e., has built a relationship between the data to recognize (it does not only recognize ranking while train recognizes (is influenced by the input with the effect of changing neuron), builds relationship and ranks).

[9] That is why we expect every time we run the network, we get a different training curve reaching a different minimum.

[10] 0 is not realistic as an error target.

Does not affect network judgment in terms of interconnection (does not use weight & bias changes per epoch) of neurons (simply triggers neurons when subjected to a threshold based on criteria)[11] simply remarkable data based on what it learned from the train dataset based on error reduction (y = (t - x) .^ 2) relative to approach zero.

Validation

Subset of stochastic data (green line Figure 11) of the total data for error-based verification. That is, a third of the data by amount makes sense to continue training due to stagnation (local minimum) change the increase of the training curve (overfit) validation after a certain number of repetitions (epochs) to local minimum or some overfitting training (prevents by identification) is usually characterized by an immediate increase along with the red line test data (overfit).

At the beginning of the "construction" of the neural network we usually do not put in the data separation and this for control because it prevents us to look for any core errors in the selection of techniques / algorithms of the parameters for network training (see above)

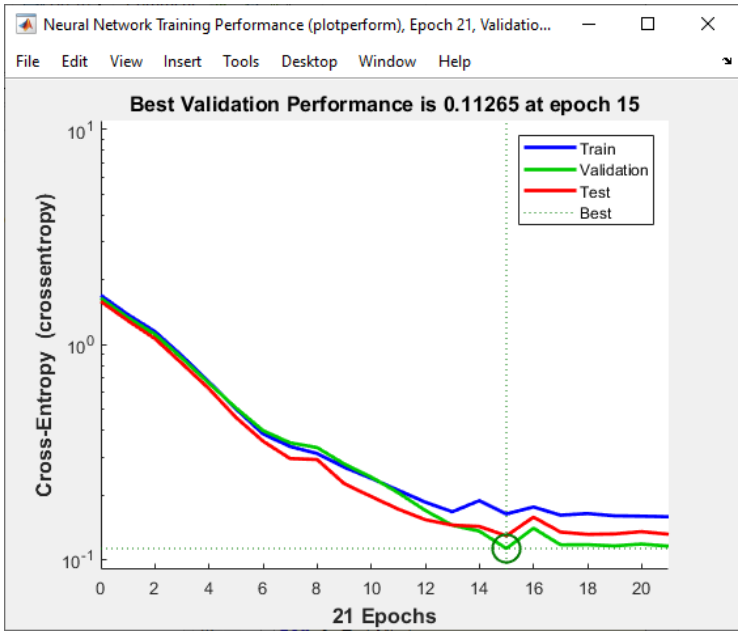


Figure 11 training curve

[11] That is, in each run it will not have improved the errors of the test data

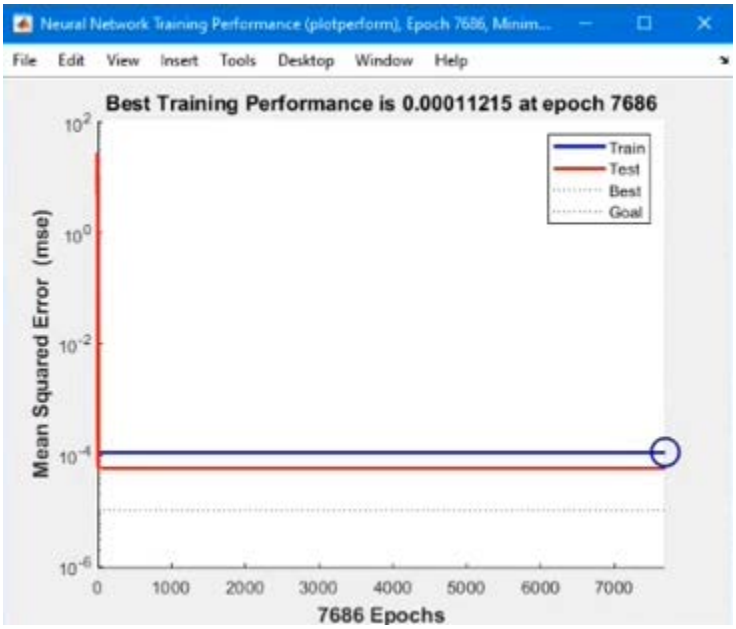


Figure 12 Local minimum stuck (both lines must) but close to solution function by minimizing the total expected (from data) error

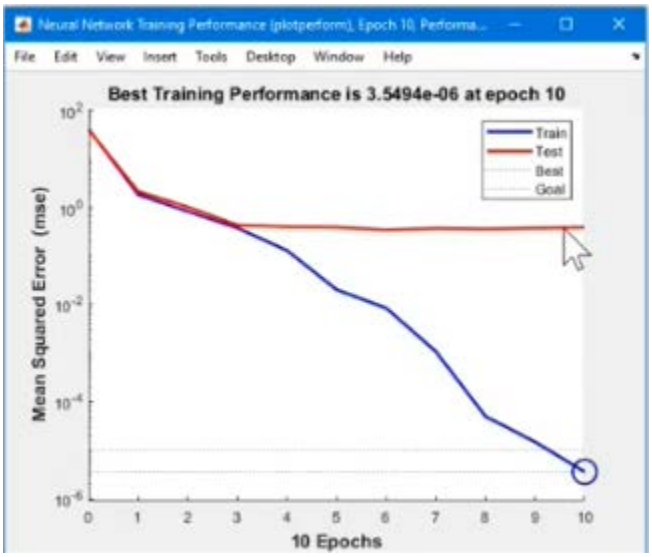


Figure 13 Example of strong neural network failing (overfitting) on new data classification (there is not a local minimum error cause of train data success)

2.2.2.2. Neural Network Architecture Parameters

The most important architectural criteria of network designs as a landmark are the hidden levels that the neural network will consist of but also how many neurons are interconnected per layer. The depth of the problem will be understood in order to use a proper device to solve it. There is no ideal other than its estimation by the neural network designer. Simple problems require simple neural networks with a small number of neurons and hidden levels, more complex problems are estimated to increase the resources in neurons and hidden layers. As mentioned above the use of incorrect parameters for the coupling of levels and neurons leads to the failure of proper training or response in the presence of new data for evaluation.

In our specific problem it has been used after observing results after training 1 hidden layer with 10 neurons with sigmoid transfer function[12] (we verify it with a command *net.layers{1}.transferFcn* Figure 20)[13] [4] (as well as 5 would be marginally sufficient for this example) with 1500 epochs using an iterative on-the-fly variety of training functions (trainscg, trainrp, traingdx[14]) Figure 24 Figure 23.

The objectives are to re-train with the same data feed (as mentioned above) responding to new results due to different initializations of conditions and examples at the Cartesian axis level. The smaller the marker repetition error (epoch) the better the ranking of binary or non-data. To achieve this, the data train (blue line) is adjusted in the training step weights & bias[15] to come closer to the affordable result and expected from the beginning knowing the (supervised ML).

Summary

1. Justify your choices about:
    a. The division of data into training-test-validation.
        i. Achieves proper training and new prediction / separation of the neural network avoiding errors and locally minimal by creating sensitivity in the synapses of neurons by recognizing patterns. (Supervised ML)
    b. The initial conditions (initial choice of weights and biases).
        i. They are used to handle loss function reductions per epoch calculating better arrival at the solution through monitoring of the error function reductions and existing per neuron in the input. Each iteration (subset of epoch) is adjusted through the gradients of the loss landscape, ie the backpropagation technique. The information gradient for a specific error is calculated in relation to the initial error and the neurons between that particular neuron and the error. When weight updates are great there are negative scenarios. This is because when the weights are swinging back and forth, it is likely that we have either had a very oscillating path to our world minimum. In addition, when they are large, it may be that you are constantly skipping the optimum, getting worse performance than necessary. [5]
        ii. *new_weight = old_weight - learning rate * gradient update*

        The biases for each layer i are *net.b{i}*. So for a two layer network the biases are *net.b{1}* and *net.b{2}*.

[12] In the Hidden layers a sigmoid transfer function is recommended

[13] That is, they are non-linear neurons (MLP perceptrons) with any differentiable function e.g., $a = \frac{1}{1+e^{-h}}$

[14] Traingdx consists of special extra parameters of learning rate and momentum which they are tend to help in overcoming local minimum threshold errors. Command to display *net.trainParam.lr* or *.mc* [17] Figure 24 Figure 22

[15] Weight initialization occurs on the fly during train-data stage process but bias is pre-defined beforehand (with zeros). We can evaluate this by running *net.b{1}* and *net.IW{1}* in the console or script window before train the NN with train-data input and after them Figure 9. It's worth noting that both of them via mathematical functions are being manipulated so manual override is not advised. Further we can examine *net.biases{1} net.layerWeights{2} net.inputWeights{1} net.layerWeights{2}.learnParam* which differs from *net.trainParam.mc net.trainParam.lr* that has been assigned in script code *net.trainParam.lr = 0.1 net.trainParam.mc = 0.4* Figure 14 Figure 12

The weights to layer i from input j are *net.IW{i,j}*. For a typical two layer network *net.IW{1,1}* will exist, while *net.IW{2,1}* will be empty because the input only goes to layer 1.

The weights to layer i from layer j are *net.LW{i,j}*. For a typical two layer network *net.LW{2,1}* will contain the weights to layer 2 from layer 1 and the other layer weights will be empty.

You should also take into account the input and output processing, if you are wanting to reproduce the input-output network function yourself. These functions and settings are available for a two-layer network with these properties: [6]

- *net.inputs{1}.processFcns*
- *net.inputs{1}.processSettings*
- *net.outputs{2}.processFcns*
- *net.outputs{2}.processSettings*

c. The number of layers of the neural network.
   i. They achieve network training speed through the generalization of problem type, type by applying non-linear transformations. With hidden layers a perceptron is transformed into a deep network of storage and data processing. Each layer is used to identify an object e.g., if we have a car in visual image one layer will be for if there is glass another for if there are lights another if there are wheels et al. [7]

d. The number of neurons in each layer.
   i. They determine the speed of training and arrival in a minimum error as well as generalization. Each neuron has its own transfer function output, i.e., the threshold for triggering inputs and transferring this information to a mesh network. Usually without a hidden layer they are called shallow. A layer cannot process large amounts of multidimensional information and solves 1 problem for which it was created [8]

e. The type of transfer functions.
   i. In machine learning, the sums of each node are weighted and the sum passes through a nonlinear function known as the activation function or transfer function.
   ii. In machine learning, the activation function is used more often, while I think the "transfer function" is used more often in signal processing. Therefore, whoever uses them as two different terms should be clearer.
   iii. A value (signal strength) to verify whether the neuron will be activated and then calculate an output from it. So, the whole process can transfer a signal from one layer to another.
   iv. *transfer_function = activation function + output function* [9]

f. The type of training (learning algorithm) you used.
   i. There are several algorithms for training the neural network in the example we used:
      1. trainscg [10] (scaled conjugate gradient backpropagation)
      2. trainrp [11] (Resilient backpropagation)
      3. Traingdx [12] (Gradient descent with momentum and adaptive learning rate backpropagation)
   ii. The above belong to a variety of species between gradient descent and conjugate gradient which are quite efficient if there are many parameters saving memory / resources (computational) but it is slow, it has to do with how it approaches the

loss function with a mathematical model (numerical precision) and processing speed et al [13].

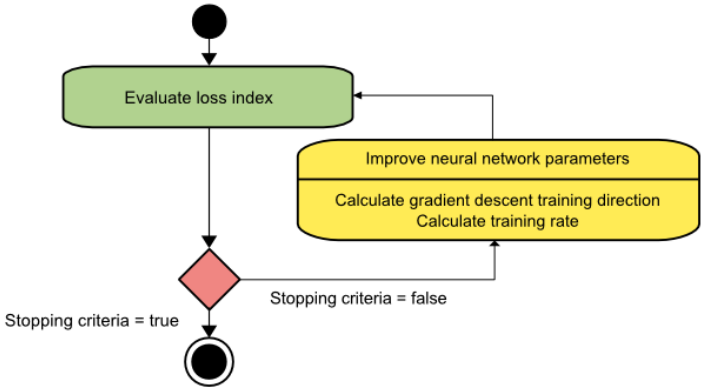


Figure 14 Loss function for gradient descent

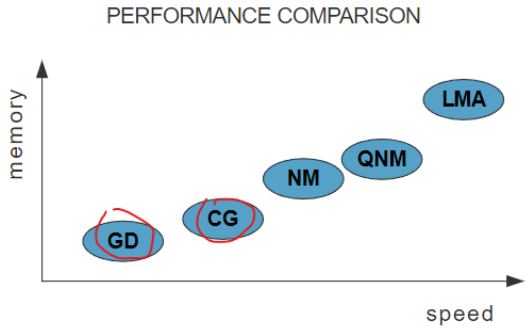


Figure 15 GD (gradient descent) CG (conjugate gradient) are slowest in speed but good in memory consumption

g. Number of epochs.
    i. Number useful for trying to reduce the error. A complete training cycle from the entire dataset. Each epoch we feed the network with different patterns and starting from a different starting point helping in generalization. Heuristic, one incentive is that (especially for large but finite training sets) the network is given the opportunity to see the previous data to adjust the model parameters so that the model is not biased towards the latest data points during duration of training. As there is no ideal as number epochs will give us a good result wants trial and error (iteration != epoch but a subset). [14]
h. The value of the learning rate and the momentum.
    i. Used to minimize the error function but also to avoid a local minimum. We set it before the start of training and it makes a smaller gradient update. it will take us longer to converge, but it probably has no overflow and oscillation less severely in epochs. So, it indirectly helps weights see 1.e.iv [5]

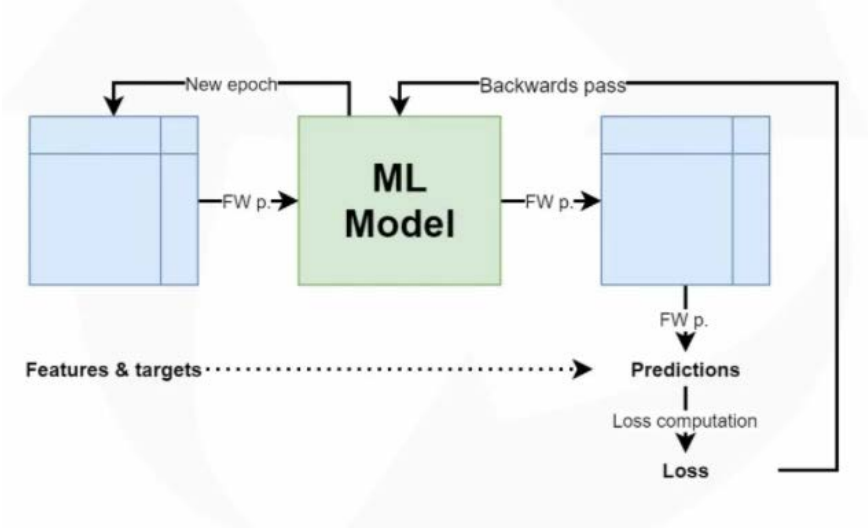


Figure 16 high-level machine learning process for supervised learning scenarios

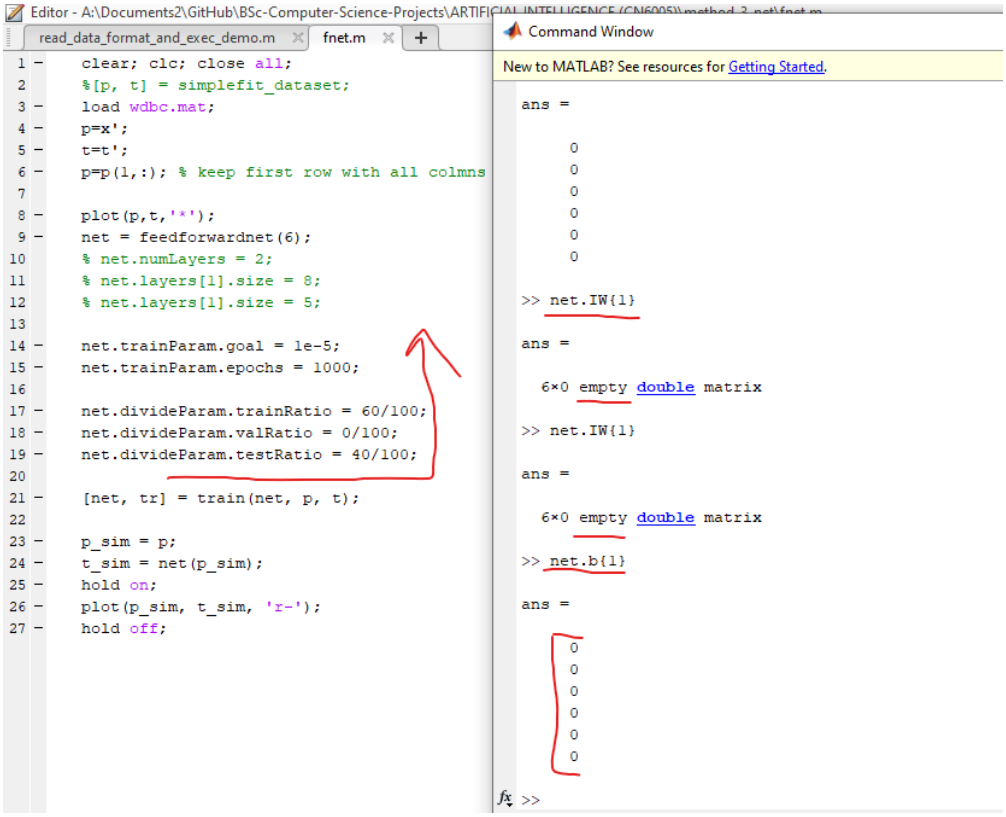


Figure 17 Weight & Bias initialization

2.2.2.3. Training Neural Network Examples for Cancer

(Control + mouse scroll to zoom in or right click save picture as)

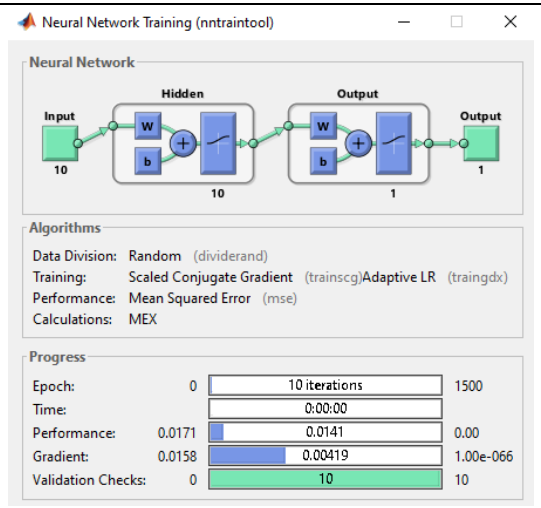


Figure 18 view(net)

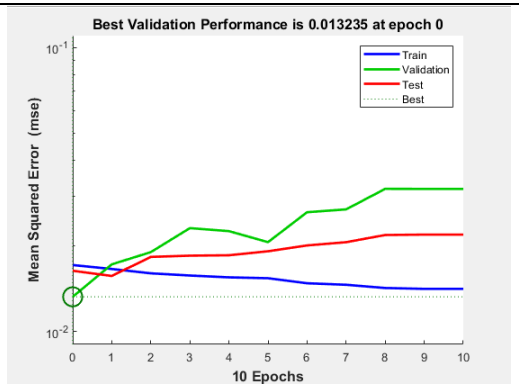
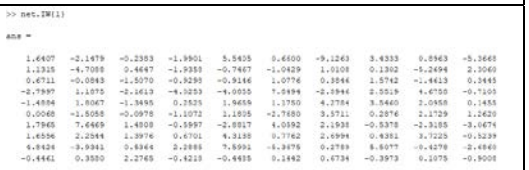
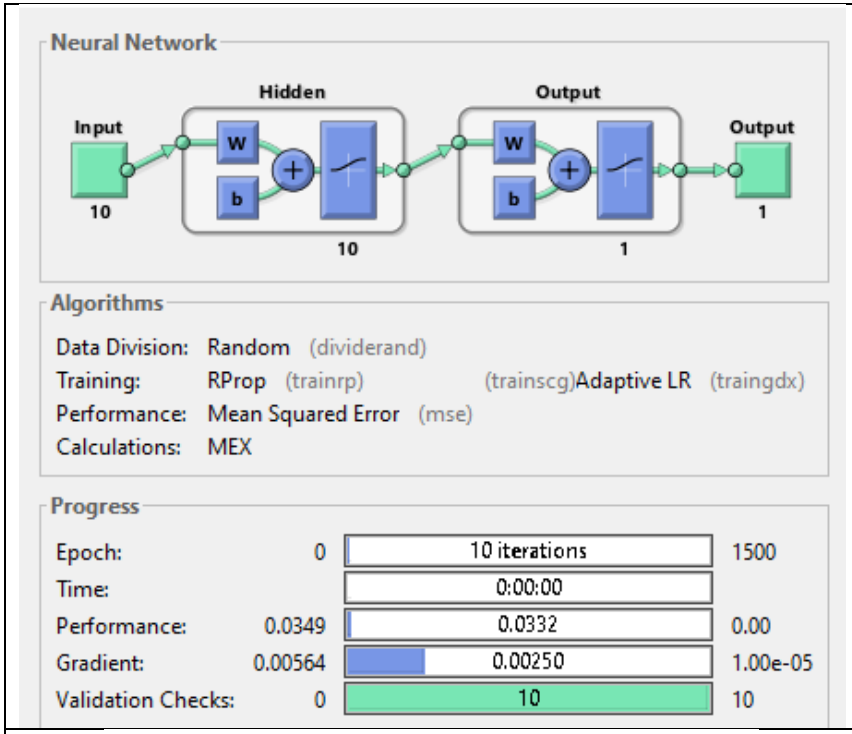


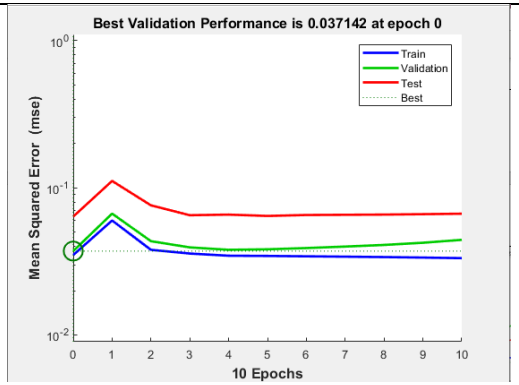
Figure 19 15 re-train times


```
>> net.layers{1}

ans =

    Neural Network Layer

            name: 'Hidden'
      dimensions: 10
     distanceFcn: (none)
   distanceParam: (none)
       distances: []
          initFcn: 'initnw'
     netInputFcn: 'netsum'
   netInputParam: (none)
       positions: []
           range: [10x2 double]
            size: 10
     topologyFcn: (none)
     transferFcn: 'logsig'
   transferParam: (none)
        userdata: (your custom info)
```

```
>> net.b{1}

ans =

   -7.8001
   -3.1603
   -2.2789
   -0.0152
    1.7831
    2.5049
    6.4097
    5.2486
    4.7571
   -4.3827
```

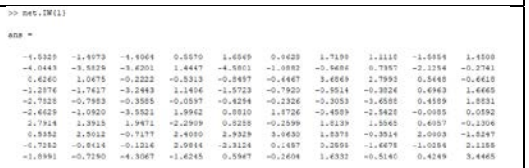


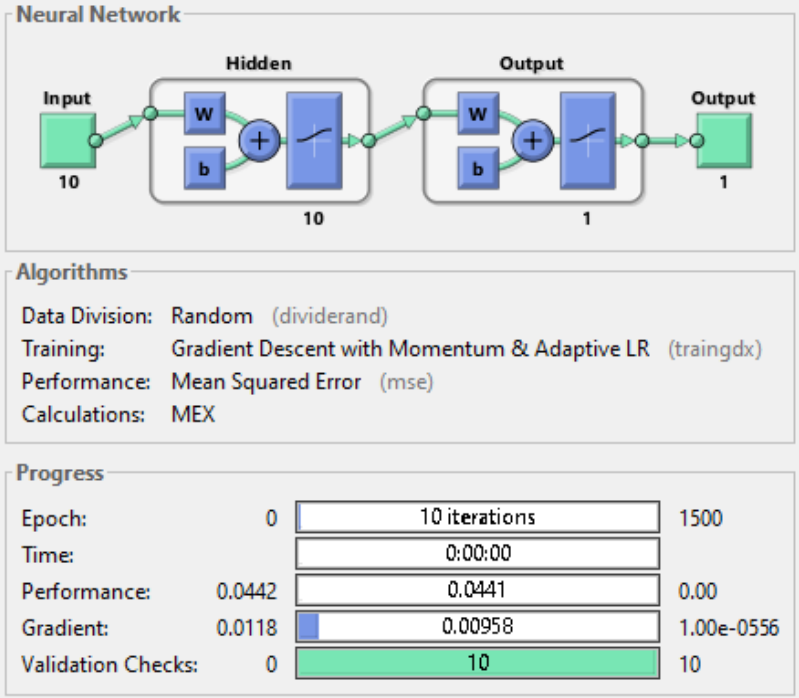


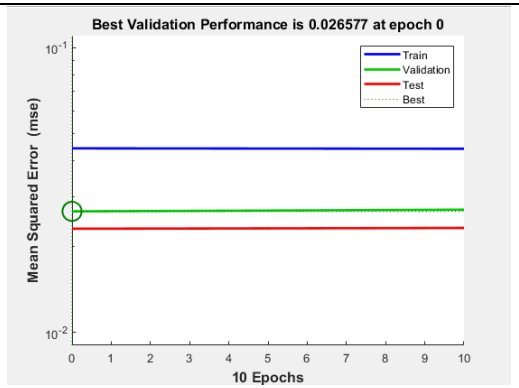



```
Neural Network Layer

          name: 'Hidden'
    dimensions: 10
  distanceFcn: (none)
distanceParam: (none)
     distances: []
       initFcn: 'initnw'
  netInputFcn: 'netsum'
netInputParam: (none)
     positions: []
         range: [10x2 double]
          size: 10
  topologyFcn: (none)
  transferFcn: 'logsig'
transferParam: (none)
      userdata: (your custom info)
```

```
>> net.b{1}

ans =

    2.1788
   -3.6911
    2.9451
   -1.4997
    0.3857
   -1.3237
    2.0892
   -1.8647
    0.7464
    1.2309
```





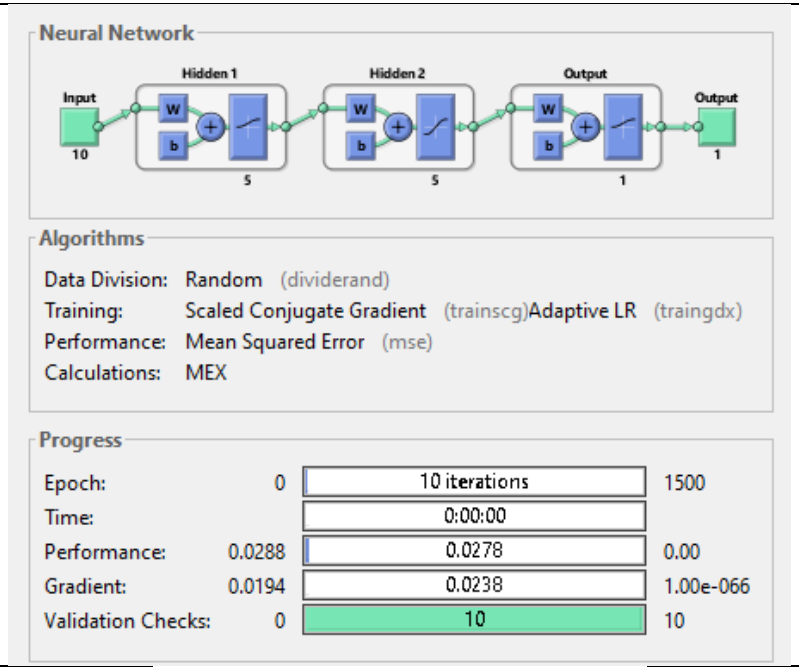


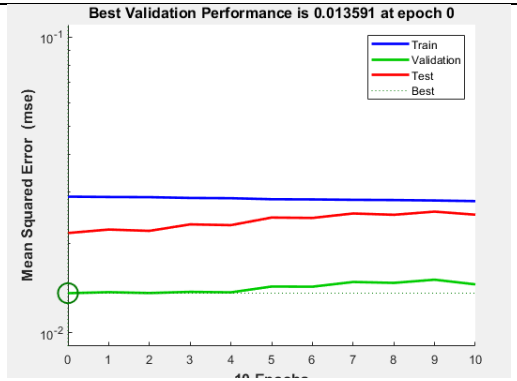
Local minimum


```
Neural Network Layer

            name: 'Hidden'
      dimensions: 10
     distanceFcn: (none)
   distanceParam: (none)
       distances: []
          initFcn: 'initnw'
     netInputFcn: 'netsum'
   netInputParam: (none)
       positions: []
           range: [10x2 double]
            size: 10
     topologyFcn: (none)
     transferFcn: 'logsig'
   transferParam: (none)
        userdata: (your custom info)
```

```
>> net.trainParam.lr

ans =

    0.1000

>> net.trainParam.mc

ans =

    0.4000
```

```
>> net.IW{1}

ans =

   -0.3104    1.7221    0.2135   -1.4349   -1.4481   -0.3439   -0.0032   -0.8530   -1.3500   -1.5241
   -1.2650    1.7833   -0.8007   -0.2287   -1.3837    0.3389   -0.3206   -1.8545    0.3700   -1.0760
    2.3188   -0.3436    0.4621    0.6619   -1.4765   -0.0756   -0.7527   -0.9126   -0.1999   -1.5233
    0.9024   -0.9694    0.4333    0.5961    0.7822   -1.6276   -2.3016   -1.1366    0.6141   -0.4407
   -0.9341   -1.4421   -2.6607    0.9162   -1.6804    0.2906   -0.2613   -1.3913   -0.1822    0.9560
   -1.7003   -1.4527   -0.2619   -1.7365   -1.6198    1.8475   -0.6556   -0.8263   -1.4606   -0.0026
    1.8941    1.6014    0.1802   -0.6599    2.1924    1.0577    1.4443    0.7761   -0.3188   -1.9021
    0.2971    0.5534    1.7409   -0.6064   -0.5523    0.2840    2.5428    0.2262   -0.4738    0.9754
    0.7005    2.1781    0.5099   -1.1114   -0.0715    1.9720    0.3045    1.1666    0.3545   -1.3362
   -1.2650    0.9797    0.1759    0.8910   -2.3932   -1.4178   -0.8519    0.7648    0.1207    0.9560
```





Local minimum


```
>> net.layers{1}

ans =

    Neural Network Layer

            name: 'Hidden 1'
      dimensions: 5
   distanceFcn: (none)
 distanceParam: (none)
       distances: []
          initFcn: 'initnw'
    netInputFcn: 'netsum'
  netInputParam: (none)
       positions: []
           range: [5x2 double]
            size: 5
    topologyFcn: (none)
    transferFcn: 'logsig'
  transferParam: (none)
        userdata: (your custom info)
```

```
>> net.layers{2}

ans =

    Neural Network Layer

            name: 'Hidden 2'
      dimensions: 5
   distanceFcn: (none)
 distanceParam: (none)
       distances: []
          initFcn: 'initnw'
    netInputFcn: 'netsum'
  netInputParam: (none)
       positions: []
           range: [5x2 double]
            size: 5
    topologyFcn: (none)
    transferFcn: 'tansig'
  transferParam: (none)
        userdata: (your custom info)
```

```
>> net.layers{3}

ans =

    Neural Network Layer

            name: 'Output'
      dimensions: 1
   distanceFcn: (none)
 distanceParam: (none)
       distances: []
          initFcn: 'initnw'
    netInputFcn: 'netsum'
  netInputParam: (none)
       positions: []
           range: [1x2 double]
            size: 1
    topologyFcn: (none)
    transferFcn: 'logsig'
  transferParam: (none)
        userdata: (your custom info)
```

```
>> net.b

ans =

  3×1 cell array

    {5×1 double}
    {5×1 double}
    {[  0.4866]}
```

```
>> net.b{1}

ans =

    3.9668
   -3.0018
   -0.9299
   -2.9993
    3.4718
```

```
>> net.b{2}

ans =

    6.1417
    0.4905
    0.2736
    0.3082
   -0.8668
```

Two bias layers because 2 hidden layers

```
>> net.IW

ans =

  3×1 cell array

    [5×10 double]
    [0×0  double]
    [0×0  double]

>> net.IW{1}

ans =

   -1.0272   -1.4514    0.4568   -2.2016   -0.8042    0.2431   -0.5284    0.6858   -0.0937   -1.7935
    0.9852    0.6584    0.9082    0.9870    0.2596    1.3202   -0.9220   -0.6812    1.3153    0.6325
   -0.1044   -1.4237   -3.3275   -0.8438   -1.5917   -1.3036   -0.5218   -2.9903    0.0076    0.4743
   -1.6052   -2.0470   -0.8869   -0.6594   -0.9198    0.5973   -1.0293   -2.5895   -0.6856    0.5656
    0.2987   -0.3314   -0.7191   -1.4370    0.0095    1.5257   -1.6938    1.3150   -1.0193   -0.3716
```

```
>> net.layerWeights

ans =

  3×3 cell array

    {0×0 double    }    {0×0 double    }    {0×0 double}
    {1×1 nnetWeight}    {0×0 double    }    {0×0 double}
    {0×0 double    }    {1×1 nnetWeight}    {0×0 double}
```

2 Weight Layers on hidden

```
trainscg = Average Percentage Total Correct Classification   : 96.789690% | MSE : 0.027962%
trainscg = Average Percentage Total Incorrect Classification : 3.210310%
trainrp = Average Percentage Total Correct Classification   : 94.586995% | MSE : 0.040048%
trainrp = Average Percentage Total Incorrect Classification : 5.413005%
traingdx = Average Percentage Total Correct Classification   : 94.376098% | MSE : 0.040119%
traingdx = Average Percentage Total Incorrect Classification : 5.623902%
>>
```

```
>> net.layers{2}

ans =

    Neural Network Layer

            name: 'Output'
      dimensions: 1
     distanceFcn: (none)
   distanceParam: (none)
       distances: []
          initFcn: 'initnw'
     netInputFcn: 'netsum'
   netInputParam: (none)
       positions: []
           range: [1x2 double]
            size: 1
     topologyFcn: (none)
     transferFcn: 'logsig'
   transferParam: (none)
        userdata: (your custom info)

>>
```

Figure 20 Layer transfer function

```
>> net.biases{1}

ans =

    Neural Network Bias

          initFcn: (none)
            learn: true
          learnFcn: 'learngdm'
        learnParam: .lr, .mc
             size: 10
         userdata: (your custom info)
```

Figure 21 biases, learnFcn, LearnParam for traingdx

```
>> net.layerWeights{2}

ans =

    Neural Network Weight

            delays: 0
          initFcn: (none)
     initSettings: .range
            learn: true
         learnFcn: 'learngdm'
       learnParam: .lr, .mc
             size: [1 10]
        weightFcn: 'dotprod'
      weightParam: (none)
         userdata: (your custom info)
```

Figure 22 Layer Weights function

```
functions:

      adaptFcn: 'adaptwb'
    adaptParam: (none)
      derivFcn: 'defaultderiv'
     divideFcn: 'dividerand'
   divideParam: .trainRatio, .valRatio, .testRatio
    divideMode: 'sample'
       initFcn: 'initlay'
    performFcn: 'mse'
  performParam: .regularization, .normalization
      plotFcns: {'plotperform', 'plottrainstate', 'ploterrhist',
                 'plotconfusion', 'plotroc'}
    plotParams: {1x5 cell array of 5 params}
      trainFcn: 'traingdx'
    trainParam: .showWindow, .showCommandLine, .show, .epochs,
                .time, .goal, .min_grad, .max_fail, .lr, .lr_inc,
                .lr_dec, .max_perf_inc, .mc

weight and bias values:

              IW: {2x1 cell} containing 1 input weight matrix
              LW: {2x2 cell} containing 1 layer weight matrix
               b: {2x1 cell} containing 2 bias vectors

methods:
```

Figure 23 net object functions

2.2.2.4. Code

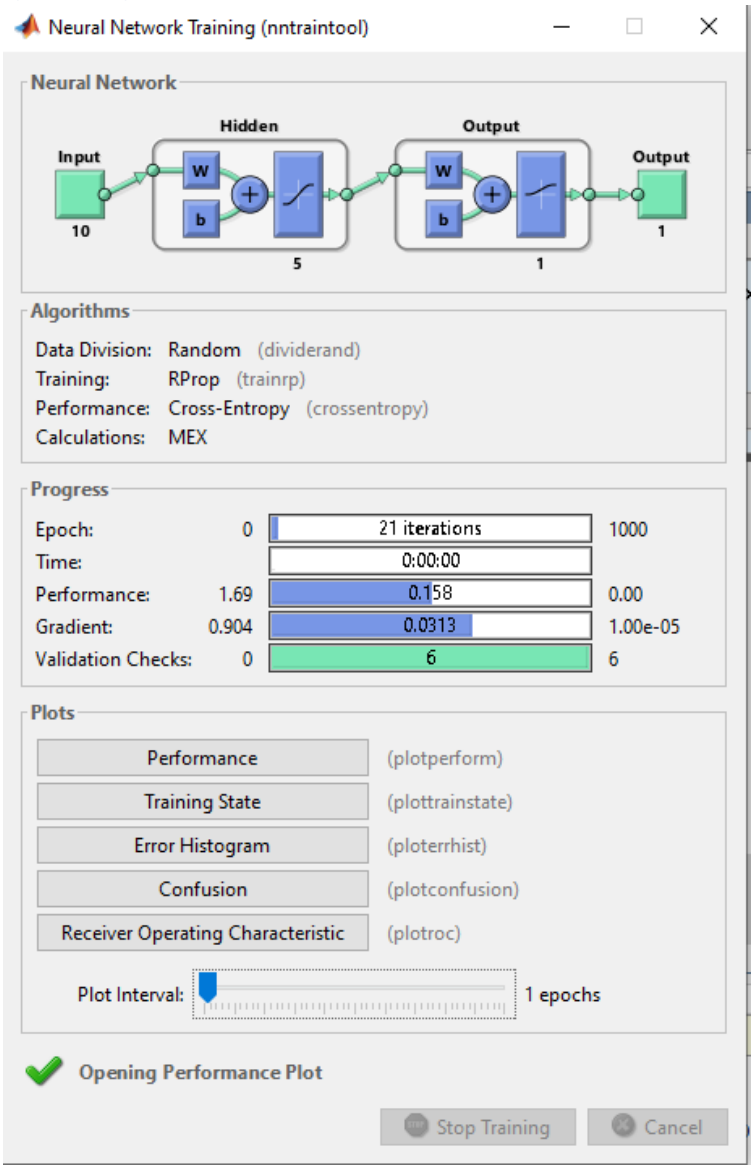
```
%% Prepare Neural Network

% every iteration new training function tested numerous times and getting
average results of the mse
for test_various_training_functions_average_results = 1:3
    hidden_layers_neurons = [10];
    if (test_various_training_functions_average_results == 1)
        training_function = 'trainscg';
    end
    if (test_various_training_functions_average_results == 2)
        training_function = 'trainrp';
    end
    if (test_various_training_functions_average_results == 3)
        training_function = 'traingdx';
    end

    net = patternnet(hidden_layers_neurons);
    %net.divideFcn = 'dividerand';
    net.trainFcn = training_function;
    %net.trainParam.goal = 1e-2;
    net.trainParam.epochs = 1500;
    net.trainParam.show = 1;
    net.performFcn = 'mse'; % or crossentropy

    net.divideParam.trainRatio = 70/100;
    net.divideParam.valRatio = 15/100;
    net.divideParam.testRatio = 15/100;

    net.trainParam.max_fail = 10; % when validation should kick in by
stopping the process of the neural network

    % traingdx only
    if (strcmp(training_function, 'traingdx'))
        % In setting a learning rate, there is a trade-off between the rate
of convergence and overshooting.
        net.trainParam.lr = 0.1; % Corresponds how fast will move (step size
at each iteration) towards goal (minimum of loss function)

        net.trainParam.mc = 0.4; % Corresponds how strong (control adaptation
parameter) will be the influence of the direction (training gain) that is
heading towards (affects error convergence)

    end

    net = init(net);

    % transpose only on first run the matrix cells
    if (test_various_training_functions_average_results == 1)
         x = x(1:569,1:10); % Feature selection only the first 10 columns of
569 samples
         x = x';
         t=t';
    end

    [net,tr] = train(net, x, t); %train
```

Figure 24 Part 1 of script

```
%% Confusion Matrix

    outputs= zeros(length(x(:,:)),length(x(1,:))); % fill with 569x569
    pcorrect=0;
    perror=0;

    % Average re-train confusion
    mse=0.0;
    for i=1:15 % 15 times re-train
        [net,tr] = train(net, x, t); % re-train
        outputs = net(x);
        Nepochs = tr.epoch(end); % Total epochs iteration of net train
        N = length(x); % Proccessed input Data Length
        for i_epoch = 1:Nepochs % iterations run (total number of epochs
trained)
            for i_mse = 1:N % do addititive summary 569 times row (1 epoch)
then divide it with 569.. do that for Nepochs
                mse(1) = mse(1) + mean((t(i_mse)-outputs(i_mse)).^2); % add
in first col the previous mse + next the row-by-row actual target from Data -
Data from output you got instead
            end
            mse(1) = mse(1)/N;
        end
        [c,cm,ind,per] = confusion(t,outputs);
        pcorrect = pcorrect + (100*(1-c)); % numeric representation of
Correct Classification
        perror = perror + (100*c); % numeric representation of Incorrect
Classification
        if (i==15)
            pcorrect = pcorrect / 15;
            perror = perror / 15;
            fprintf('%s = Average Percentage Total Correct Classification   :
%f%% | MSE : %f%%\n', training_function, pcorrect, mse(1));
            fprintf('%s = Average Percentage Total Incorrect Classification :
%f%%\n', training_function, perror);
        end
    end
end
```

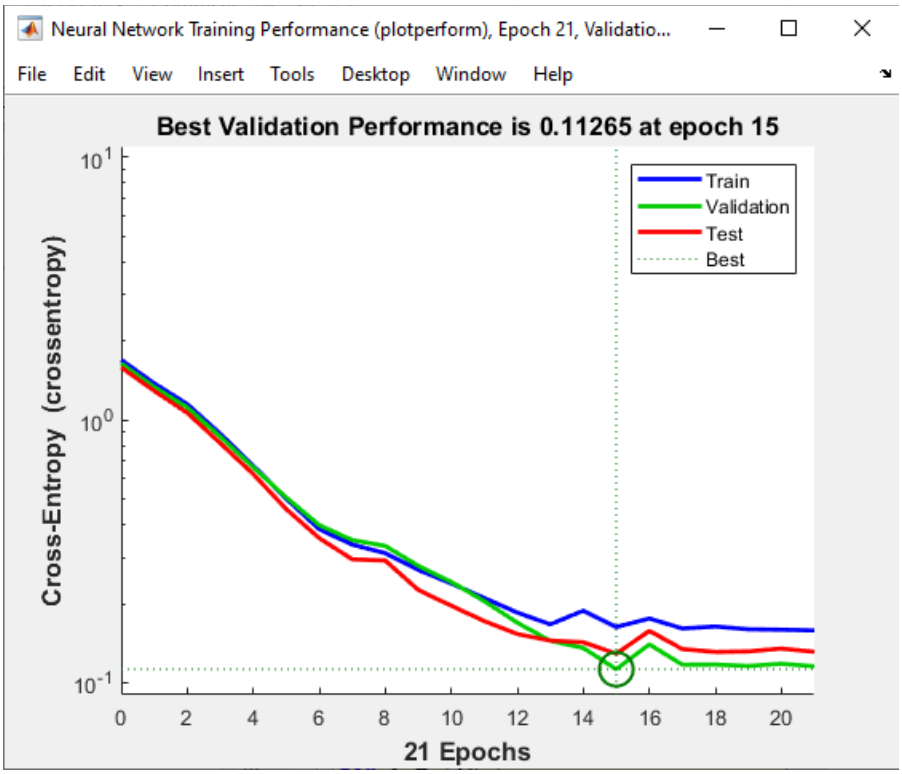
Figure 25 Part 2 of script

2.2.2.5. Plot figures diagrams



In the figure above we see a network with a hidden layer of 5 neurons with sigmoid transfer function and 10 feature / samples / rows input with 1 output (1 or 0, Malignant or Benign) by dividing the data randomly into 3 categories (train-test- valid) use of dividend.

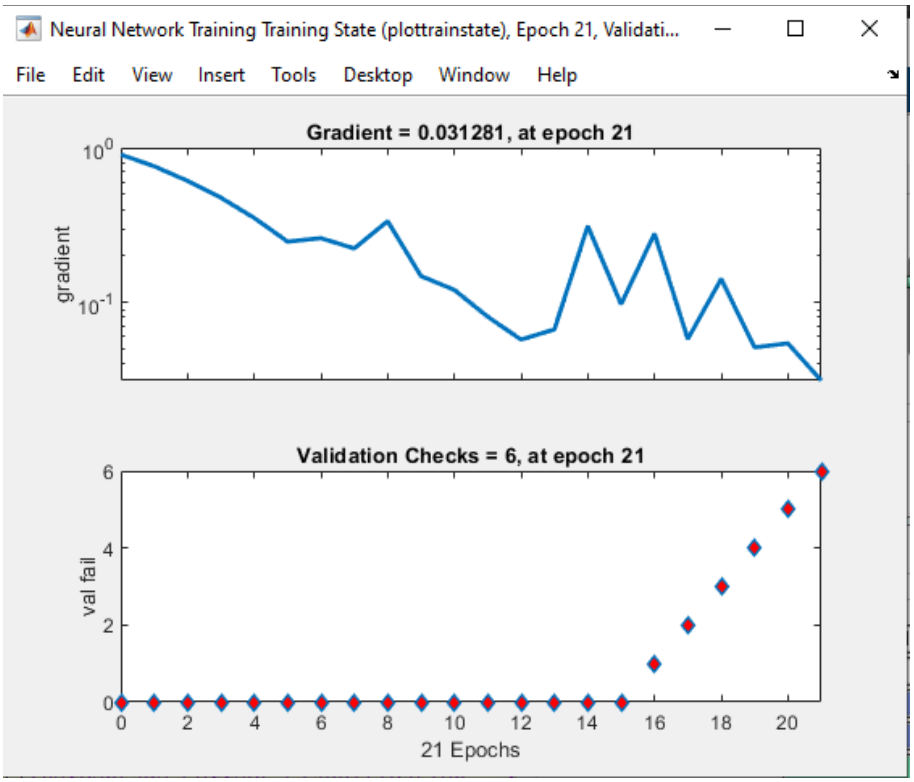
The trainrp algorithm has been used for training and function loss cross-entropy (instead of mse in this example because of the multiple tests). It took 21 repetitions until the validation check of 6 tries to stop the training.



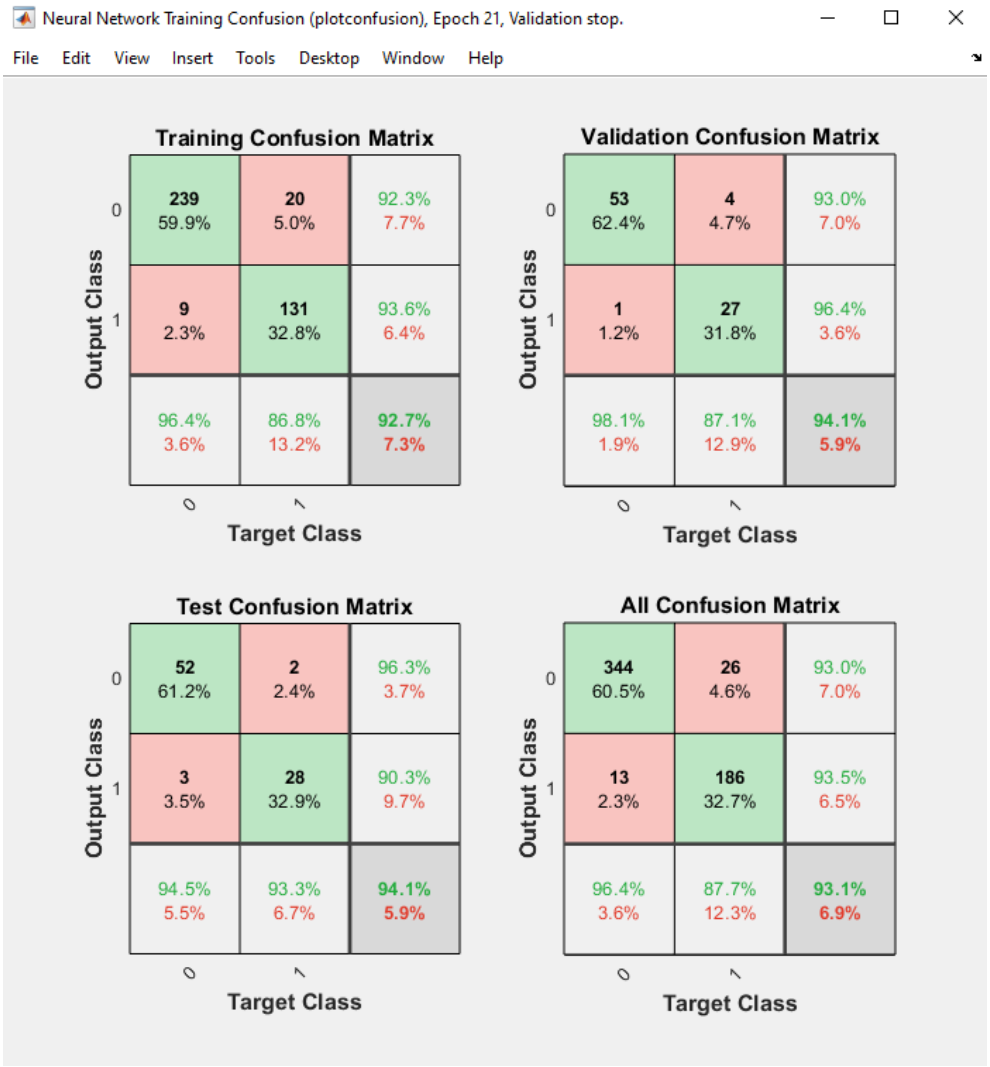
In the above Plot we see a good course of the 3 categories of training data and neural network verification. In general, it would be good for all 3 of these curves to go stuck with no sharp fluctuations but smoothly (harmoniously) as close as possible to the smallest error rate.

We also conclude that the validation after 6 attempts since it found the minimum between 14-16 stopped the training (best validation performance) of the network as the prices stuck to a local minimum but this reason of the small error is almost the universal minimum.

I do not observe overfitting phenomena (keeping the concept of generalization at a positive level) because the red line and the green line should have a sharp sudden upward course.



We see the best validation performance between 14-16 trying 6 times to keep the training in operation due to reflection but in this case we have the so-called convergence (convergence) ie the network will not improve further so it stops training.



The most important diagram (plot) especially the part of Test confusion matrix.

It shows us that out of the 2 categories (1-malignant, 0-benign) in category 0 out of 54 data for test he successfully achieved 52 and classified 2 in category 1 instead of 0.

Similar to category 1 3 of them ranked in category 0 which is wrong and 28 in 1 with more to the right to see the percentage percentage achieved 90.3%

The most important value is the overall result (mainly of the test data) which gives us a score of 94.1% success with 5.9% failure.

Conclusion? It went well! as in each re-train as mentioned above these values change either above or below with a higher score than observed at 97.2% in the test data and a minimum of 89%. train but give the result for a higher score (in the picture the training for x number epoch had run 1 time) and adjust the system so that it is not polarized / biased / comparing the results with the previous ones without remaining stationary in the most recent results.

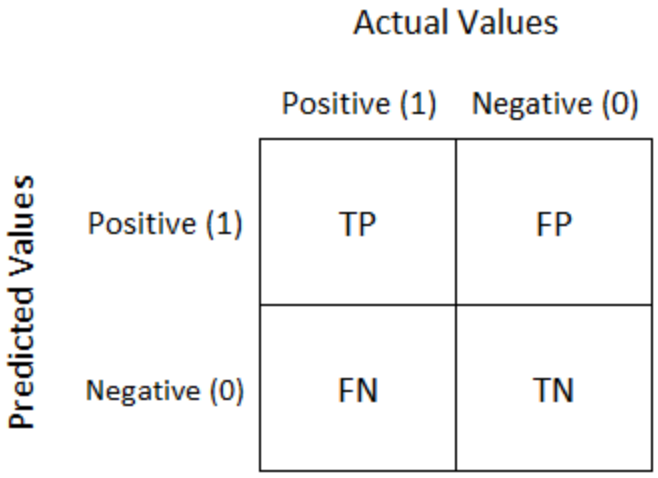


Figure 26 Confusion Matrix True/False Positives, False/True Negatives

True / False Positive = When true (correct) in category positive (1) and false categorization in error but from positive again.

False / True Negative = When it is true (correct) in the category negative (0) and false categorization in the wrong but from the negatives again.

That is, we are interested in the cross (diagonal) how much TP & TN succeeded correctly the rest goes to FP & FN which are invalid.

This has a lot of impact and not only on logistic regression[16] where we have a lot of confusion matrix if we get a lot of high-low thresholds to prevent this from happening, we come to ROC & AUC (Receiver Operator Characteristic & the Area Under the Curve).

[16] In Logistic Regression it has points, you place thresholds for statistics, categorize them in different y-axis values, and you draw conclusions from many confusion matrices (non-efficient) or you use ROC (sum of matrices) for better visual all at one place (efficient).

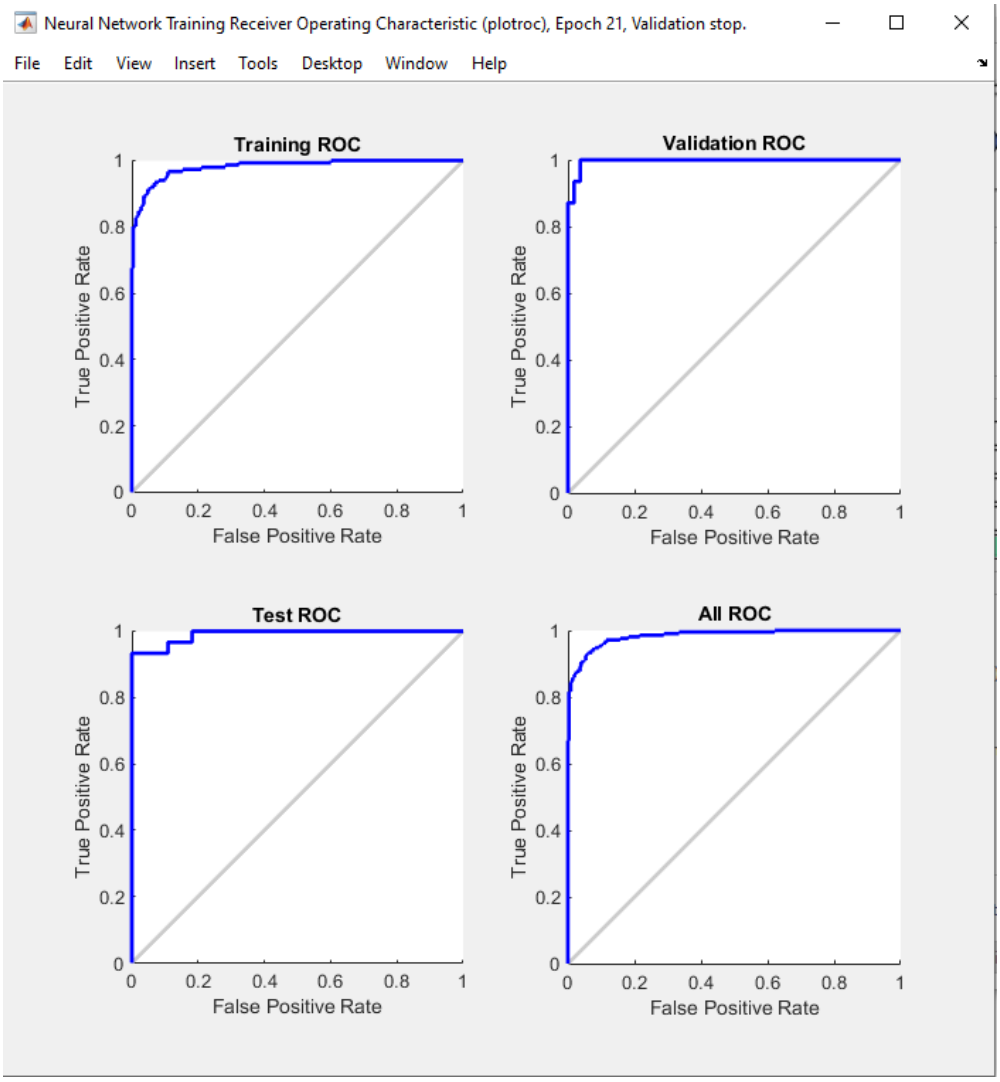


Figure 27 ROC (Receiver Operator Characteristic) summarizes all of the confusion matrices that each threshold produced [15]

The Curve is read from right to left showing the farther to the left there are points the more correct results it brought to that threshold (taking the 1.1 point in the level as equal categorization of right and wrong i.e., 10 error 10 correct and decreasing the error as it goes left)

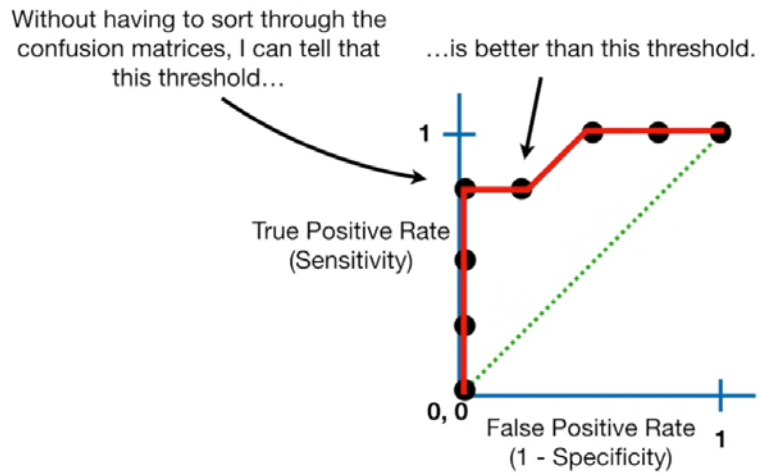


Figure 28 ROC makes it easy to identify the best threshold for making a decision

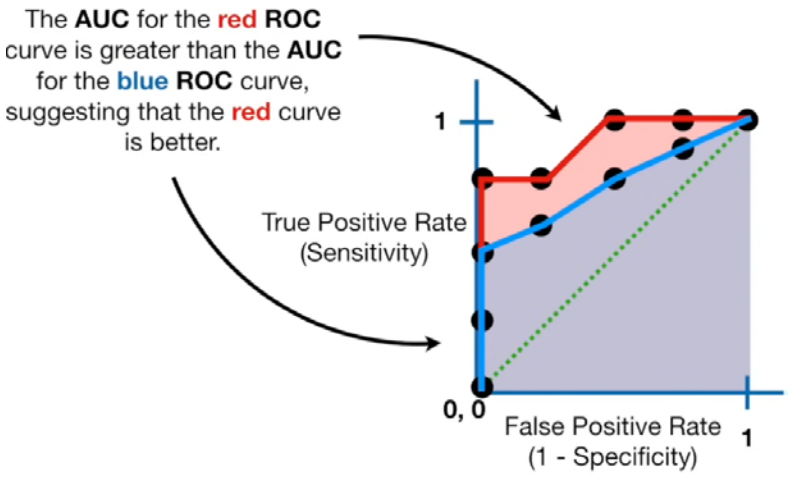
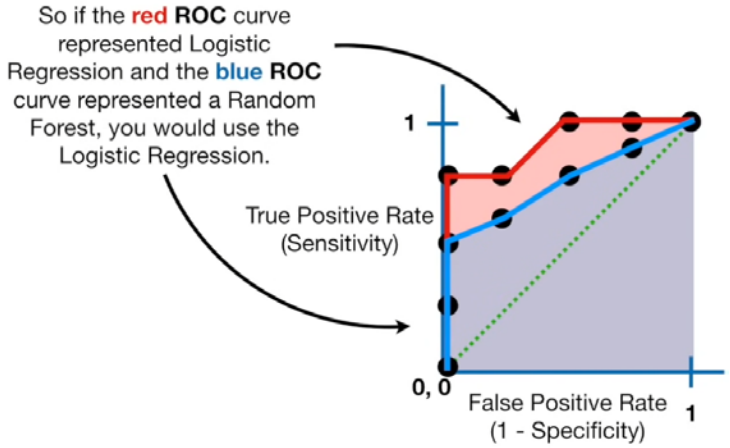


Figure 29 AUC helps to decide which categorization method is better



```
trainscg = Average Percentage Total Correct Classification   : 96.789690% | MSE : 0.027962%
trainscg = Average Percentage Total Incorrect Classification : 3.210310%
trainrp = Average Percentage Total Correct Classification   : 94.586995% | MSE : 0.040048%
trainrp = Average Percentage Total Incorrect Classification : 5.413005%
traingdx = Average Percentage Total Correct Classification   : 94.376098% | MSE : 0.040119%
traingdx = Average Percentage Total Incorrect Classification : 5.623902%
>>
```

In the image above we see the average after 15 repetitions with a variety of tests (through the script Figure 24 Part 1 of script Figure 25 Part 2 of script) as well as the calculation of the loss function.

Generalization

Generalization is the ability of the neural network to avoid so-called overfitting [3] which occurs by incorrectly configuring its initializable parameters. That is, his training framework is better than ever, reaching the minimum point of error immediately, but in reality, what is happening is that he loses the ability to achieve new categorization / prediction of data into new unknown data because he memorizes patterns to **noise** and not to **signal**. That is, it focuses so much on training data that it is obviously largely biased (all data is because we always collect it from a percentage of availability and not from all over the planet or universe depending on the problem) so what the system achieves is yes it follows biased only that it has learned without causing abstraction layers to the general pattern of the signal (i.e., the common pattern extract through / in the data (**signal**) and not the data itself as pattern (**noise**)). To avoid this we always need data filtering i.e., feature selection engineering heuristics [16](see page 1) so that neither the algorithm (but also the choice of the algorithm itself) has to process many features that do not make sense in the existing problem as output but also the parameters of the algorithm itself are properly structured so as not to lead to bias or weakness categorizations due to insufficient computing resources or input dataset (underfit[17]) and test, test, test for separation by cross-validation (even without validation data also however we will not have early stopping like that). In generalization / regularization the curves flow smoothly towards the minimum error rate (train + test + valid) which means that the hidden layers and neurons per layer are properly adjusted for each problem.

[17] Occurs in a simple model with few features the wrong configuration of the algorithm (regularized too much without learning from the dataset). Ordinary learners tend to have less variance in their predictions but more bias towards wrong results. On the other hand, advanced learners tend to have more variation in their predictions (the so-called The Bias-Variance Tradeoff on page 36). Usually, we can reduce the error by bias, but we can increase the error by variance as a result, or vice versa. This hedge between very simple (high bias) versus overly complex (high sensitivity / variance) is a key concept in statistics and machine learning and affects all supervised learning algorithms. Both bias and variance are forms of prediction error in machine learning. [19] on page 38

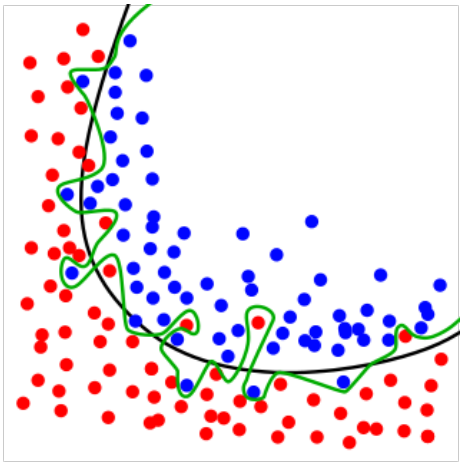


Figure 30 While the black line fits the data well, the green line is overfit. So in statistics Goodness of fit is good but in ML is not.

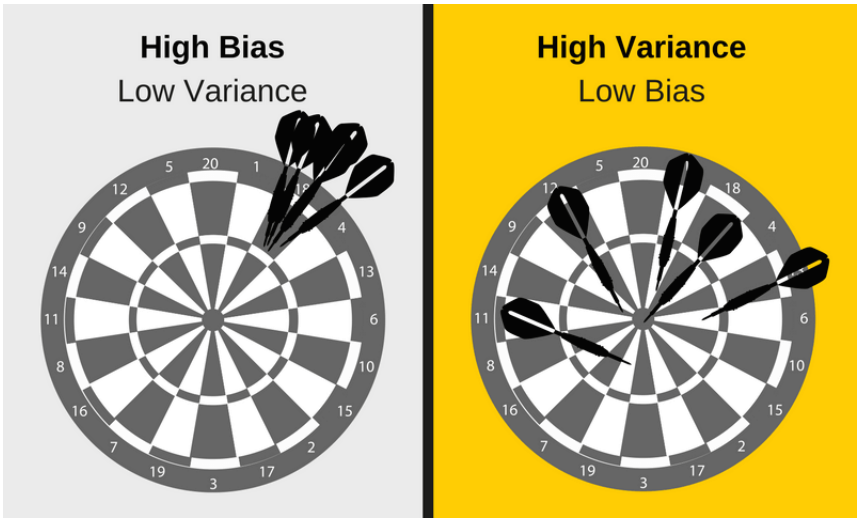


Figure 31 Bias vs. Variance (source: EDS) e.g., High bias from poor input

2.3. Conclusion (Michail Markou)[i]

For the training of a neural network model, we follow a common standard best practice workflow but there are no best / good practices for its construction, only the effort and observation of the designer for the results brings because there is no ideal based on the respective problem (trial and error). From the collection of raw data and their grouping / processing to their separation into grid feed and verification data, the whole process contributes to the proper design so that the neural network becomes consumer ready / production ready, meaning that the predictions are useful and follow specific policies. in the ideological & social part as well as in the technical and of course and more importantly the avoidance of a biased system. An intelligent artisan model enables us to study the world through big data that we are unable to process or find patterns in problems through the myriad examples so that the model unlike us is able to analyze and apply knowledge recognizing future problems from patterns arising from unknown data. A simulation of the human brain with stochastic tendencies.[18]

[18] With the right use and the right collection of data without carrying prejudices and the problems of the real world in the part that does not concern science and its evolution such as the problem of economics, you owe a profit for

In our example we see that it has been applied as an automated data entry process is used (as well as a large amount of data from them for a correct classification) with a common structure recognizing the useful from the unnecessary data (data cleansing) at the level of pre-processing. As then follows its design through a variety of algorithm options and parameters (tuning) to become an aggregate and find an average for which algorithm / technique has the best result in the present problem. Due to the nature of supervising machine learning a "supervisor" "supervisor" "teacher" you might say must be present throughout the workflow / pipeline process not only as an observer but also as a facilitator to evaluate and correct the results.

Finally, what we are interested in when we apply machine learning application is the ability to generalize and not just to associate an input pattern x with a training pattern t this can be done in a database, i.e., for this model x the target t is that. The peculiarity of machine learning is that for standards x that it has not seen before during the training it is able to classify them to decide which class t they belong to (for this reason we use the cross-validation method by dividing the data into train & test 80:20) all the training process is judged on the amount well recognizes unknown data finding the patterns through them i.e., test data is essentially the part that will judge a production ready machine learning model network.

Bibliography

[1] "SVM in Matlab: Meaning of Parameter 'box constraint' in function fitcsvm," [Online]. Available: https://stackoverflow.com/questions/31161075/svm-in-matlab-meaning-of-parameter-box-constraint-in-function-fitcsvm/31171332. [Accessed 12 12 2021].

[2] "What "Kernel Scale" in svm really is?," [Online]. Available: https://www.mathworks.com/matlabcentral/answers/516738-what-kernel-scale-in-svm-really-is. [Accessed 12 12 2021].

[3] "Overfitting in Machine Learning: What It Is and How to Prevent It," [Online]. Available: https://elitedatascience.com/overfitting-in-machine-learning. [Accessed 12 12 2021].

[4] "how can i changhe the transfer function of output layer of neural network?," mathworks, [Online]. Available: https://www.mathworks.com/matlabcentral/answers/84931-how-can-i-changhe-the-transfer-function-of-output-layer-of-neural-network. [Accessed 12 12 2021].

[5] "What is a Learning Rate in a Neural Network?," [Online]. Available: https://www.machinecurve.com/index.php/2019/11/06/what-is-a-learning-rate-in-a-neural-network/. [Accessed 12 12 2021].

what will be considered reliable in the system and the data have taken place and are properly utilized in research and development of innovative models. This will lead us to a better society.

[6] "Knowing the Weights in Matlab," [Online]. Available: https://www.mathworks.com/matlabcentral/answers/11815-knowing-the-weights-in-matlab. [Accessed 23 12 2021].

[7] "What is a Hidden Layer?," [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning#:~:text=In%20neural%20networks%2C%20a%20hidden%20layer%20is%20located,transformations%20of%20the%20inputs%20entered%20into%20the%20network.. [Accessed 12 12 2021].

[8] "Neuron," [Online]. Available: http://www.biologyreference.com/Mo-Nu/Neuron.html#:~:text=The%20neuron%20%28nerve%20cell%29%20is%20the%20fundamental%20unit,to%20rapidly%20send%20signals%20across%20physiologically%20long%20distances.. [Accessed 12 12 2021].

[9] "Neural network: activation function vs transfer function," [Online]. Available: https://intellipaat.com/community/16651/neural-network-activation-function-vs-transfer-function#:~:text=In%20machine%20learning%2C%20the%20sums%20of%20each%20node,function%22%20is%20more%20commonly%20used%20in%20signal%20processing.. [Accessed 12 12 2021].

[10] "Scaled conjugate gradient backpropagation," [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/trainscg.html. [Accessed 12 12 2021].

[11] "Resilient backpropagation," [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/trainrp.html?searchHighlight=trainrp&s_tid=srchtitle_trainrp_1. [Accessed 12 12 2021].

[12] "Gradient descent with momentum and adaptive learning rate backpropagation," [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/traingdx.html?searchHighlight=traingdx&s_tid=srchtitle_traingdx_1. [Accessed 12 12 2021].

[13] "5 algorithms to train a neural network," neuraldesigner, [Online]. Available: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network#ConjugateGradient. [Accessed 12 12 2021].

[14] "What is an Epoch?," [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/epoch. [Accessed 12 12 2021].

[15] "ROC and AUC, Clearly Explained! | Logistic Regression," [Online]. Available: https://www.youtube.com/watch?v=4jRBRDbJemM. [Accessed 12 12 2021].

[16] "Dimensionality Reduction Algorithms: Strengths and Weaknesses," [Online]. Available: https://elitedatascience.com/dimensionality-reduction-algorithms#feature-selection. [Accessed 12 12 2021].

[17] "What is meaning of mu in artificial neural network (NNTOOL) MATLAB?," researchgate, [Online]. Available: https://www.researchgate.net/post/What-is-meaning-of-mu-in-artificial-neural-network-NNTOOL-MATLAB. [Accessed 12 12 2021].

[18] "Computer Scientist Explains Machine Learning in 5 Levels of Difficulty | WIRED," [Online]. Available: https://www.youtube.com/watch?v=5q87K1WaoFI&list=PLiTVVRdEvpm6UuNLky1I58fy_eavP5t92&index=41. [Accessed 12 12 2021].

[19] "WTF is the Bias-Variance Tradeoff? (Infographic)," [Online]. Available: https://elitedatascience.com/bias-variance-tradeoff. [Accessed 12 12 2021].

Appendix

Glossary

Term	Definition
Convergence	Completed the change of calculation in the network that is disturbed by the normal flow (recalculation). A state where the model is aware of loss that has been settled within an error range around the final value and won't improve further. (a known state of information facts)
bias	Prejudice in a (sub) set of results without taking into account the superset (not open-minded) (or other results before). So-called one-way polarization (monopoly)
Regularization (feature selection heuristic)	Regularization refers to a broad range of techniques for artificially forcing your model to be simpler. The method will depend on the type of learner you're using. For example, you could prune a decision tree, use dropout on a neural network, or add a penalty parameter to the cost function in regression. Oftentimes, the regularization method is a hyperparameter as well, which means it can be tuned through cross-validation.

	We have a more detailed discussion here on algorithms and regularization methods.
Ensembling (feature selection heuristic)	Ensembles are machine learning methods for combining predictions from multiple separate models.
Principal Component Analysis (PCA) (feature selection heuristic)	PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.
Linear Discriminant Analysis (LDA) (feature selection heuristic) [16]	creates linear combinations of your original features. However, unlike PCA, LDA doesn't maximize explained variance. Instead, it maximizes the *separability* between classes. Therefore, LDA is a supervised method that can only be used with labeled data.

Assets

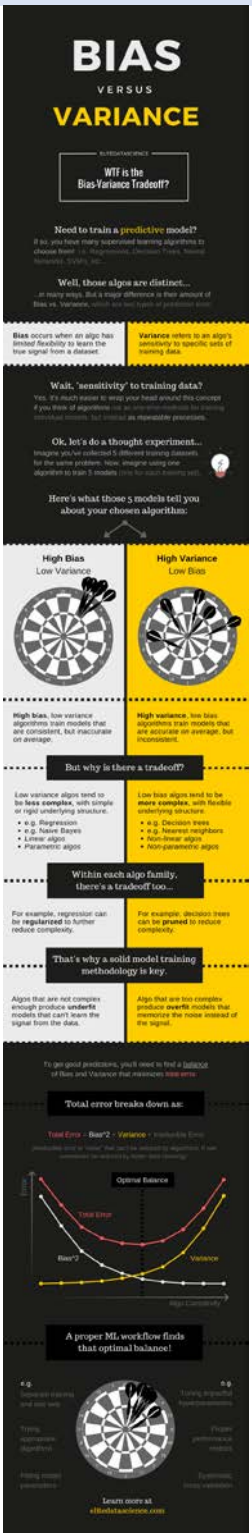


Figure 32 https://elitedatascience.com/bias-variance-tradeoff

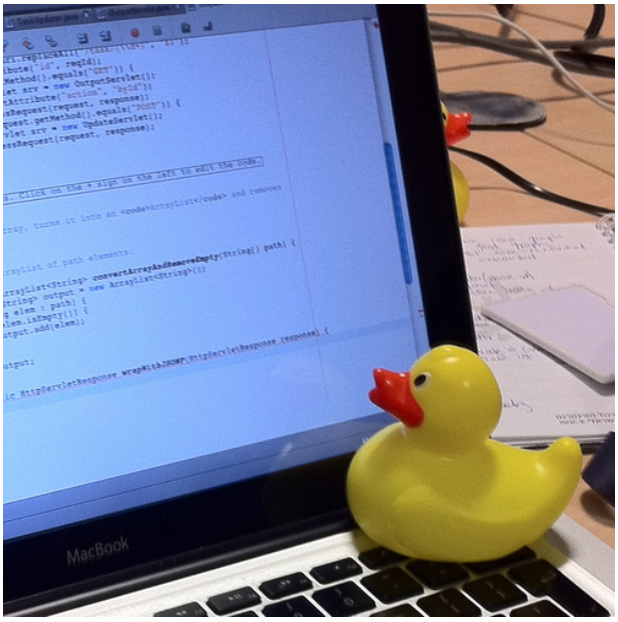


Figure 33 Rubber duck debugging

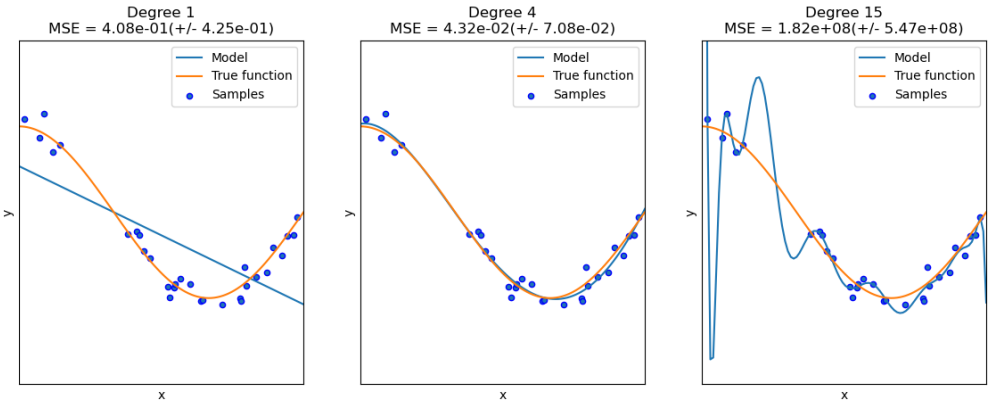


Figure 34 Left underfit, middle fit, right overfit

[i] Each Student should override this section for his own opinion.