Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity 2019-2020

Name:Trif Gheorghe Andrei
Group:30235
Email:trifandrei@yahoo.com

Assoc. Prof. dr. eng. Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Chapter 1

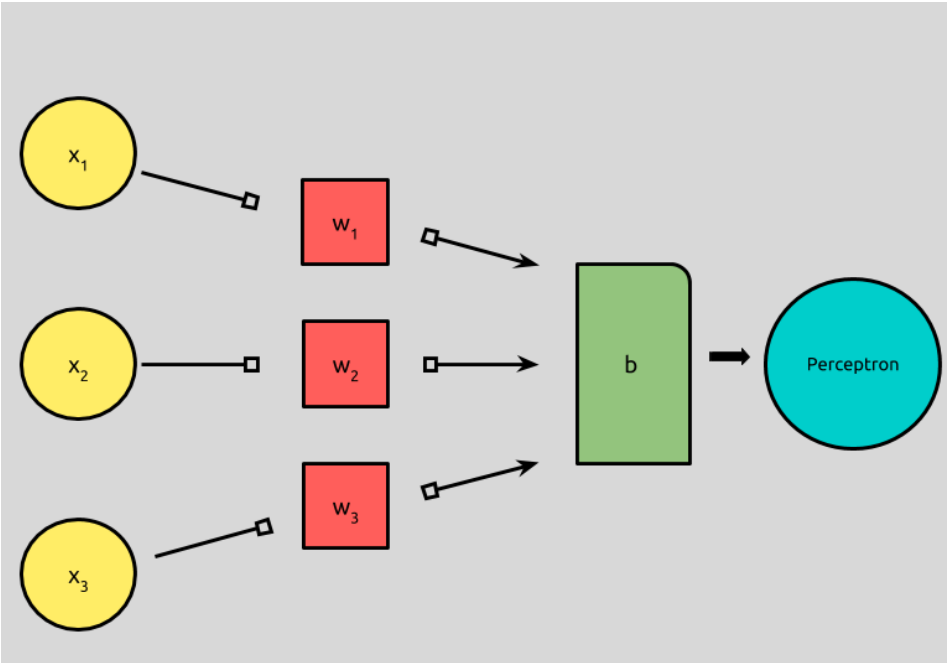
Neuronal Networks

1.1 Problem definiton

This project is about understanding and deepening the different types of neural networks.In this documentation are presented the different networks that I have chosen in this project.I will take each network in turn and explain the main advantages and how it works.Let's begin!!

1.2 1.Perceptron

Perceptrons are a type of artificial neuron .It appears that they were invented in 1957 by Frank Rosenblatt.The percepton ideea is that he takes in an input vector, x, multiplies it by a corresponding weight vector w, and then adds it to a bias, b. It then uses an activation function, (Sigmoid,ReLu) to determine our result.A perceptron looks like this:



1.3 Decision Tree

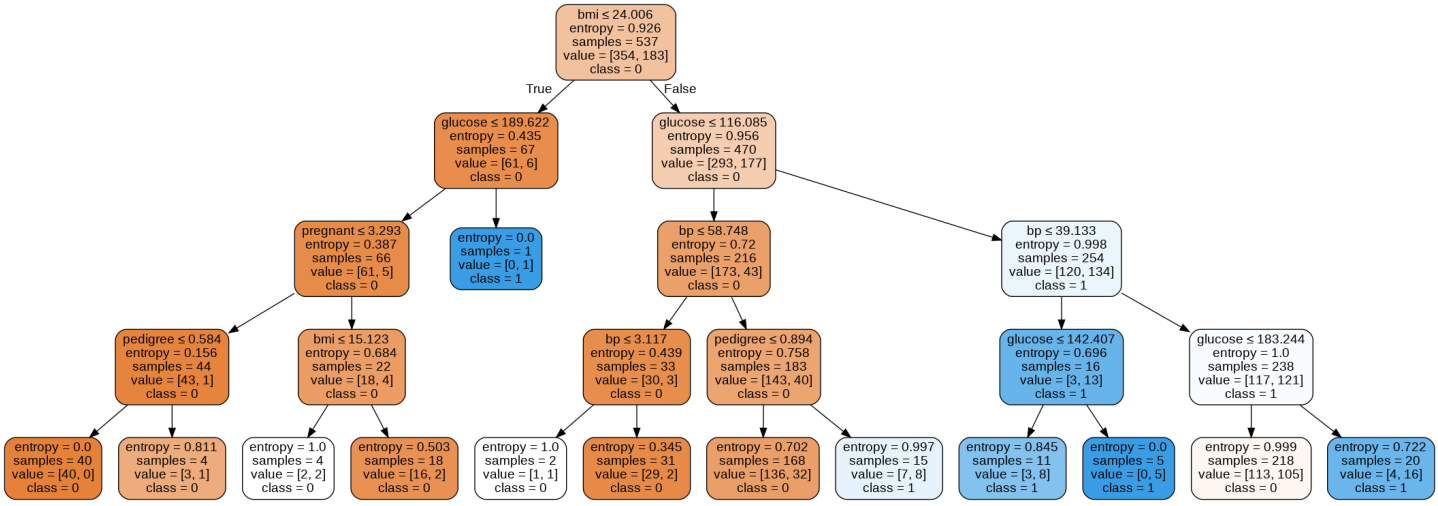
A Decision Tree Algorithm is a flowchart-like tree structure where an internal node represents feature, the branch represents a decision rule, and each leaf node represents the outcome.The basic idea behind any decision tree algorithm is as follows:

1.Select the best attribute using Attribute Selection Measures(ASM) to split the records.

2.Make that attribute a decision node and breaks the dataset into smaller subsets.

3.Starts tree building by repeating this process recursively for each child until one of the condition will match:all the tuples belong to the same attribute value or there are no more remaining attributes or there are no more instances.

For this project i run the daiabetes dataset and the algorithm return me for am max-depth of 4 the next flowchart:



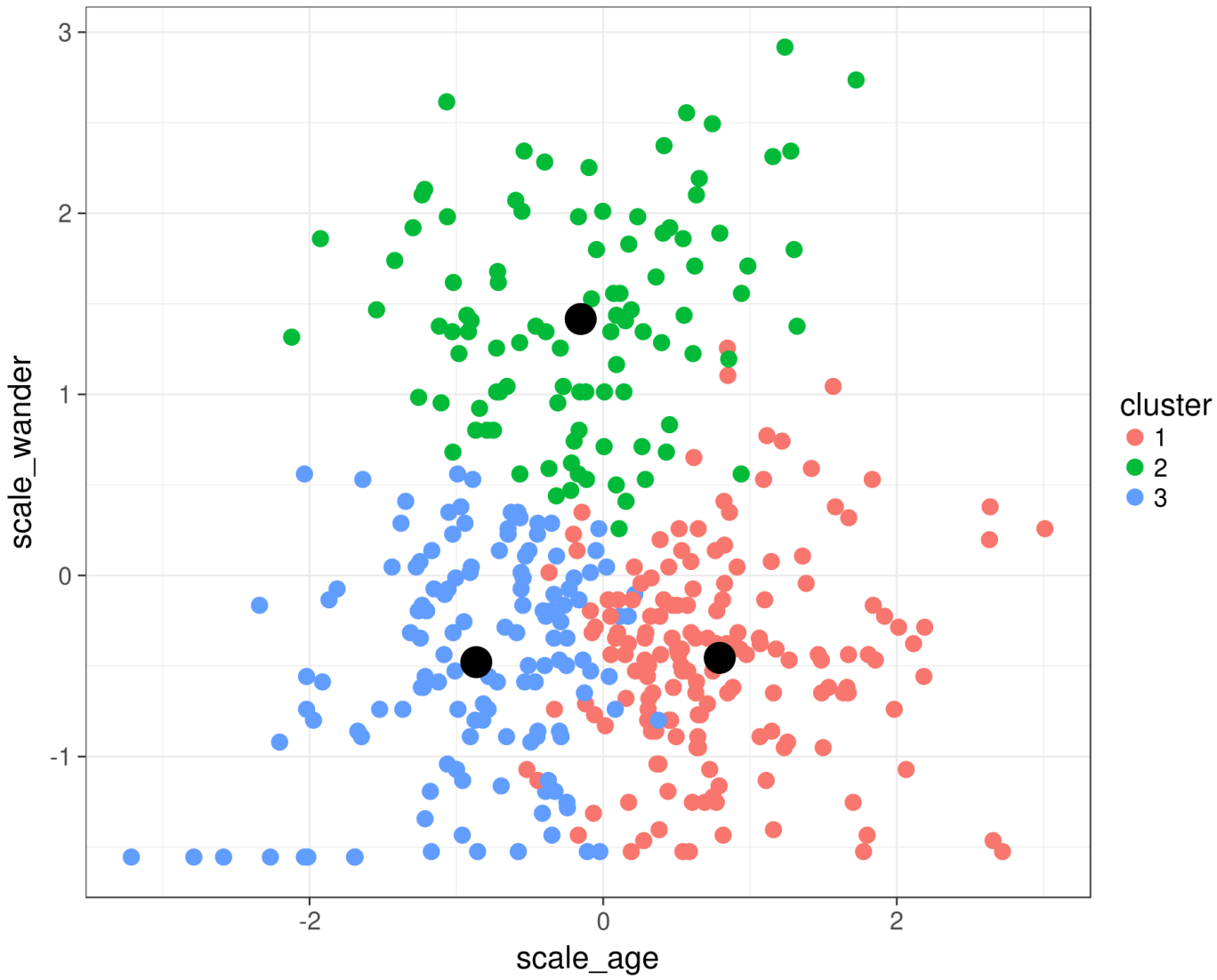
Some advantages :

Decision trees are easy to interpret and visualize.It can easily capture Non-linear patterns.It requires fewer data preprocessing from the user, for example, there is no need to normalize columns.It can be used for feature engineering such as predicting missing values, suitable for variable selection.

1.4 K-Means Clustering

K-Means Clustering is an unsupervised learning algorithm who have no variable to predict tied to the data.He use the clustering operation to group the objects who have the same characteristics.K-is the number of clusters who is equal with the numbers of centroids.

So the basic idea behind K-Means Clustering is K-Means encapsulates objects according to centroids after such clustering the algorithm tries to improve the centroids until it can no longer be improved.If in two "runs / clustering" the clusters are identical then the algorithm stops

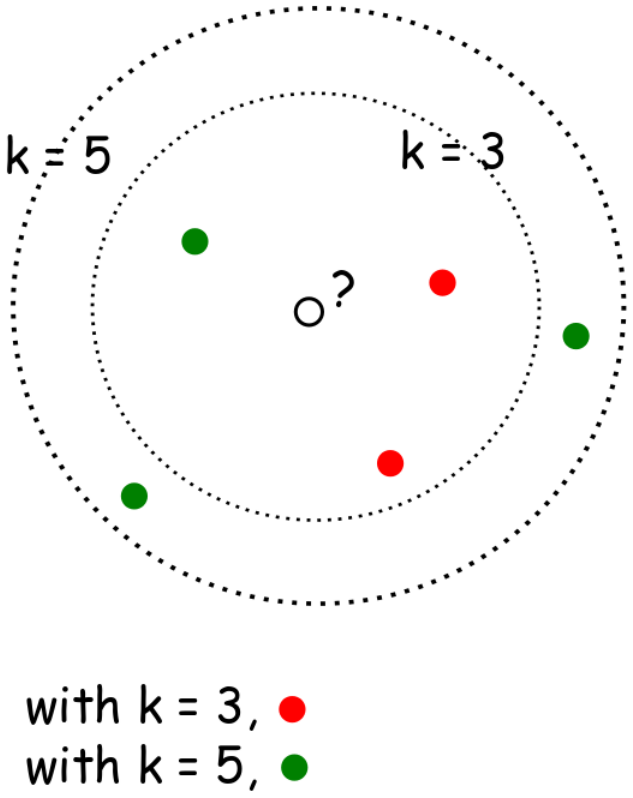


1.5 K-Nearest Neighbor(KNN)

KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution.Lazy algorithm means it does not need any training data points for model generation. All training data used in the testing phase. This makes training faster and testing phase slower and costlier.

In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbor algorithm.

KNN has the following basic steps: calculate distance, find closest neighbors, vote for labels.



Some advantages: The training phase of K-nearest neighbor classification is much faster compared to other classification algorithms. There is no need to train a model for generalization.That is why KNN is known as the simple and instance-based learning algorithm.

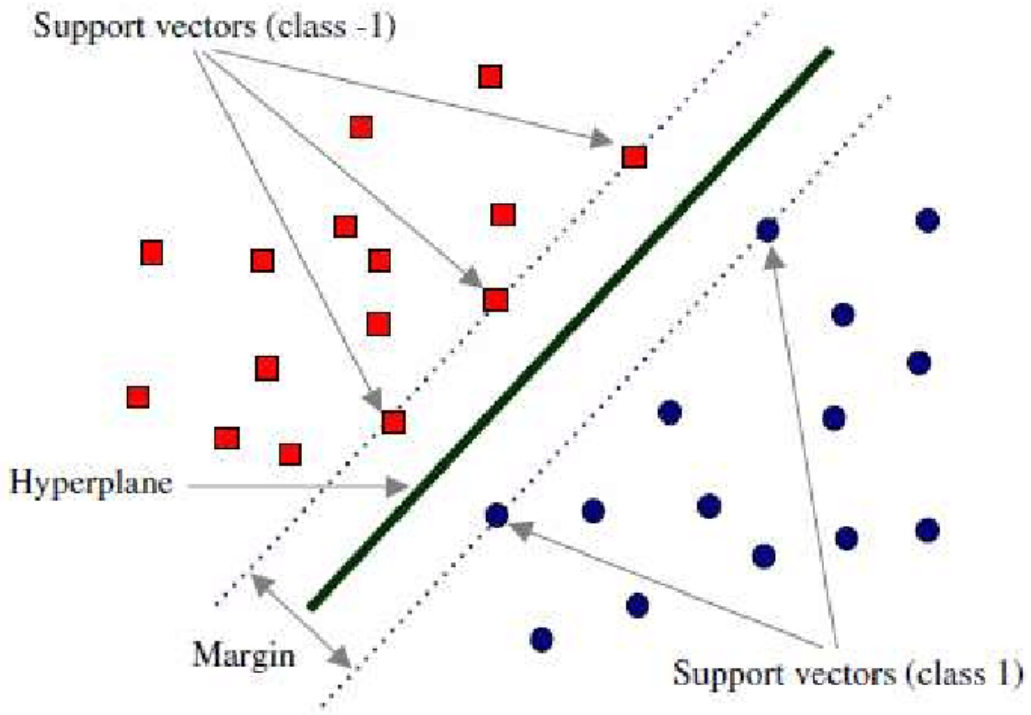
1.6 Support Vector Machines

Support Vector Machines is considered to be a classification approach, it but can be employed in both types of classification and regression problems. It can easily handle multiple continuous and categorical variables. SVM constructs a hyperplane in multidimensional space to separate different classes. SVM generates optimal hyperplane in an iterative manner, which is used to minimize an error. The core idea of SVM is to find a maximum marginal hyperplane(MMH) that best divides the dataset into classes.SVM is a binary classifier that uses supervised learning.

The main objective is to segregate the given dataset in the best possible way. The distance between the either nearest points is known as the margin. The objective is to select a hyperplane with the maximum possible margin between support vectors in the given dataset. SVM searches for the maximum marginal hyperplane:

1.Generate hyperplanes which segregates the classes in the best way. Left-hand side figure showing three hyperplanes black, blue and orange. Here, the blue and orange have higher classification error, but the black is separating the two classes correctly.

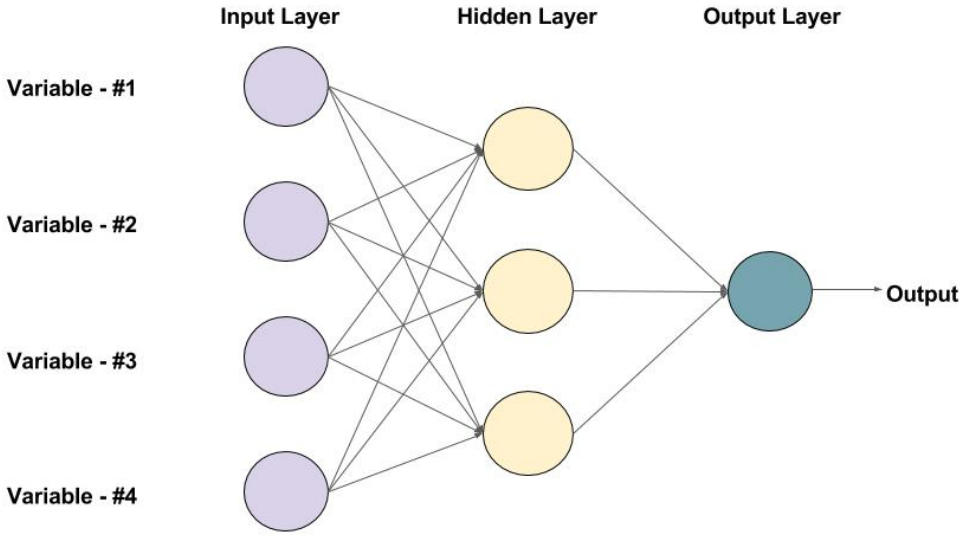
2.Select the right hyperplane with the maximum segregation from the either nearest data points as shown in the right-hand side figure.



1.7 Multilayer Perceptron(MLP)

A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to refer to any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptrons.The MLP consists of three or more layers (an input and an output layer with one or more hidden layers) of nonlinearly-activating nodes. Since MLPs are fully connected, each node in one layer connects with a certain weight to every node in the following layer.

The term "multilayer perceptron" does not refer to a single perceptron that has multiple layers. Rather, it contains many perceptrons that are organized into layers.The MLP consists of three or more layers (an input and an output layer with one or more hidden layers) of nonlinearly-activating nodes.



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

1.7.1 Overfitting

Overfitting is the case where the overall cost is really small, but the generalization of the model is unreliable. This is due to the model learning "too much" from the training data set.

1.7.2 Underfitting

Underfitting is the case where the model has " not learned enough" from the training data, resulting in low generalization and unreliable predictions.

Appendix A

Here is the link to github repository were all Neuronal Networks are :

https://github.com/trifandrei/Si/tree/NN-PROJECT-FINA Here is the most important piece of code of the MLP network.

```
trainDataset=Dataset(X_train, y_train)
trainLoader=DataLoader(dataset=trainDataset,
                       batch_size=64,
                       shuffle=True,
                       num_workers=64)
validationDataset=Dataset(X_test, y_test)
validationLoader=DataLoader(dataset=validationDataset,
                       batch_size=32,
                       shuffle=True,
                       num_workers=1)

class HeartDiseaseNN(nn.Module):
def __init__(self):
    super(HeartDiseaseNN, self).__init__()


    self.sequential= nn.Sequential(
        nn.Linear(8,100),
        nn.ReLU(),
        nn.Linear(100, 60),
        nn.ReLU(),
        nn.Linear(60, 2)
    )


def forward(self, x):
    return self.sequential(x)
net = HeartDiseaseNN()

optimizer = optim.SGD(net.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer,
```

```
def train(epoch):

net.train()
losses=[]
for batch_idx, data in enumerate(trainLoader, 0):
  inputs, labels =data

  outputs = net(inputs)

  loss = criterion(outputs, labels)

  losses.append(loss.item())

  optimizer.zero_grad()
  loss.backward()
  optimizer.step()
  print(f"[Train Epoch: {epoch}, Batch: {batch_idx+1}, Loss:
mean_loss=sum(losses)/len(losses)
scheduler.step(mean_loss)
train_losses.append(mean_loss)
print(f"[TRAIN] Epoch: {epoch} Loss:{mean_loss}")
def validation():

net.eval()

test_loss=[]
correct = 0

with torch.no_grad():
    for batch_idx, data in enumerate(validationLoader, 0):
      inputs, labels = data

      output=net(inputs)

      loss= criterion(output, labels)
      test_loss.append(loss.item())


      pred = output.data.max(1, keepdim=True)[1]
```

```
            correct += pred.eq(labels.data.view_as(pred)).sum()
            current_correct=pred.eq(labels.data.view_as(pred)).sum
            print("=============")
            print(f"[Validation set] Batch index: {batch_idx+1} Ba
            print("=============")
        mean_loss=sum(test_loss)/len(test_loss)
        test_losses.append(mean_loss)
        accuracy = 100. * correct/len(validationLoader.dataset)
        print(f"[Validation set] Loss: {mean_loss}, Accuracy: {a

        accuracies.append(accuracy)

#Here is the code for a simple perceptron.
import numpy as np
import torch
from torch import nn
class Perceptron(object):

    def __init__(self, no_of_inputs, threshold=100, learning_rat
        self.threshold = threshold
        self.learning_rate = learning_rate
        self.weights = np.zeros(no_of_inputs + 1)

    def predict(self, inputs):
        summation = np.dot(inputs, self.weights[1:]) + self.weig
        if summation > 0:
          activation = 1
        else:
          activation = 0
        return activation

    def train(self, training_inputs, labels):
        for _ in range(self.threshold):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights[1:] += self.learning_rate * (label
                self.weights[0] += self.learning_rate * (label -
```