**F21DL:**

**Data Mining & Machine Learning**

*Coursework 2*

*Students:*

Quentin DUCASSE

Sylvain TOUANEN

*Lecturers:*

Diana BENTAL

Ekaterina KOMENDANTSKAYA



Table of Contents

[Introduction 4](#_Toc26174963)

[Data Pre-Processing 5](#_Toc26174964)

[Decision Trees 7](#_Toc26174965)

[Neural Networks 8](#_Toc26174966)

[Logistic Classifier 8](#_Toc26174967)

[Experiments on Neural Networks 9](#_Toc26174968)

[Experiments on Train/Test Datasets 14](#_Toc26174969)

[Research Question 16](#_Toc26174970)

[Conclusion and Contribution 17](#_Toc26174971)

[References 18](#_Toc26174972)

# 

# Introduction

In the scope of the F21DL-Data Science and Machine Learning course, the second coursework comes as the sequel of the first one and uses the same dataset while performing new classifiers and algorithms.

The first part of the coursework, and therefore the first one of this report is centred around data *pre-processing*, how we managed files, how we processed, transformed and selected the data available. This part is mostly the result of the first coursework extended to a new use with, for example, the creation of training and testing datasets. The second part is focused on *Decision Trees*,both its accuracy, the influence of its different parameters and how it reacts to the change in size of training and testing datasets. The third part revolves around *Neural Networks*, the influence of its different parameters and how it reacts to the change in size of training and testing datasets. Finally, a research question is asked and a solution to it is provided in the final part.

This coursework has been the occasion to train the machine learning skills we obtained throughout the course, either during lectures or labs. As the subject was completely different from the ones in the labs, because it is related to computer vision, we were taking the coursework as a new challenge to prove our understanding of the course. Our initial objective for the first coursework was to use Python in order to complete the coursework as it is looking like the most used language in industry. We managed to use it and performed well. However, as Weka provides accessible tools for visualisation and parameter tuning, we chose this software as the main one for this second coursework.

The project is hosted on GitHub under [*https://github.com/QDucasse/dm\_cw*](https://github.com/QDucasse/dm_cw1)*2* along with the installation instructions and milestones of the project. In order for it to work, the actual datasets need to be downloaded at <https://www.dropbox.com/s/habmey8caojkawv/data_cw2.zip?dl=0> and unzipped inside the root project. The project was run under Weka 3.8.3 and Python 3.7.4.

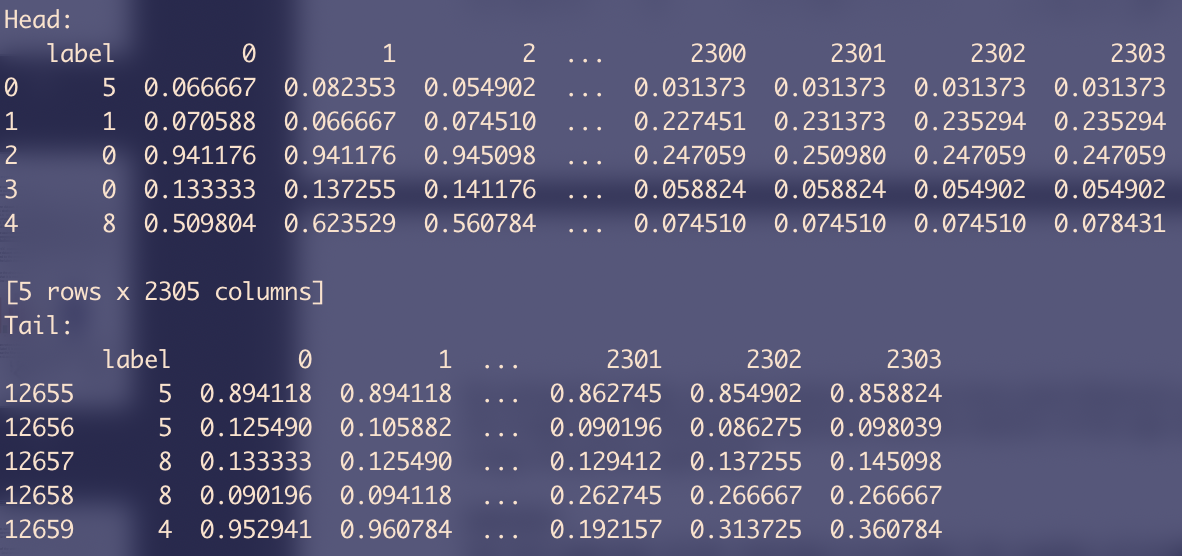
# Data Pre-Processing

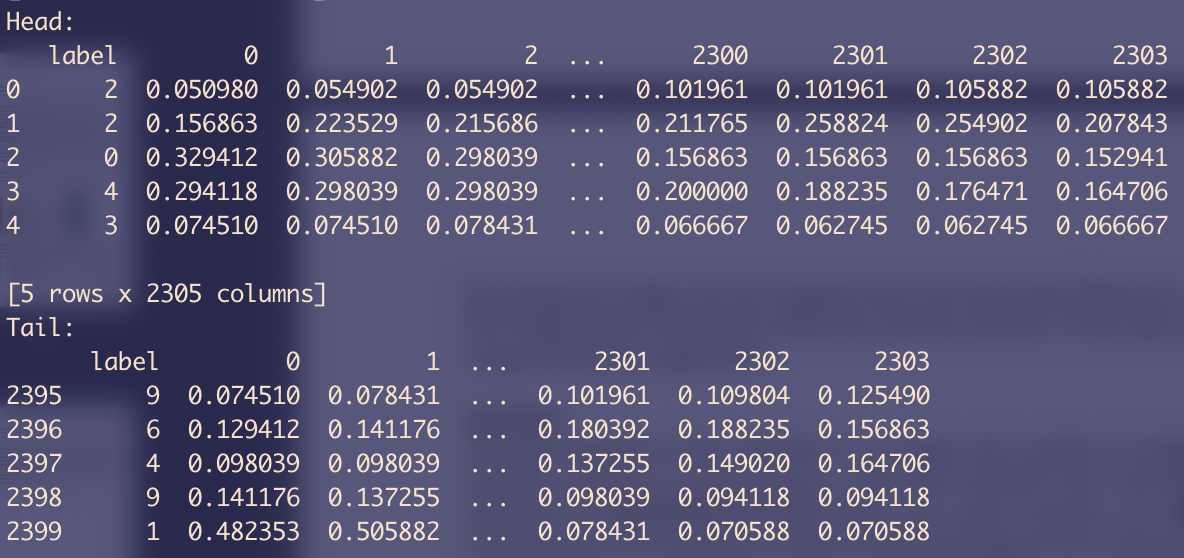
The first part of this coursework consists of a reuse of the results from the coursework 1. As a result, the actions done are summed up here. The two reused features are the *Attribute Selection* as well as the *Instance Selection.*

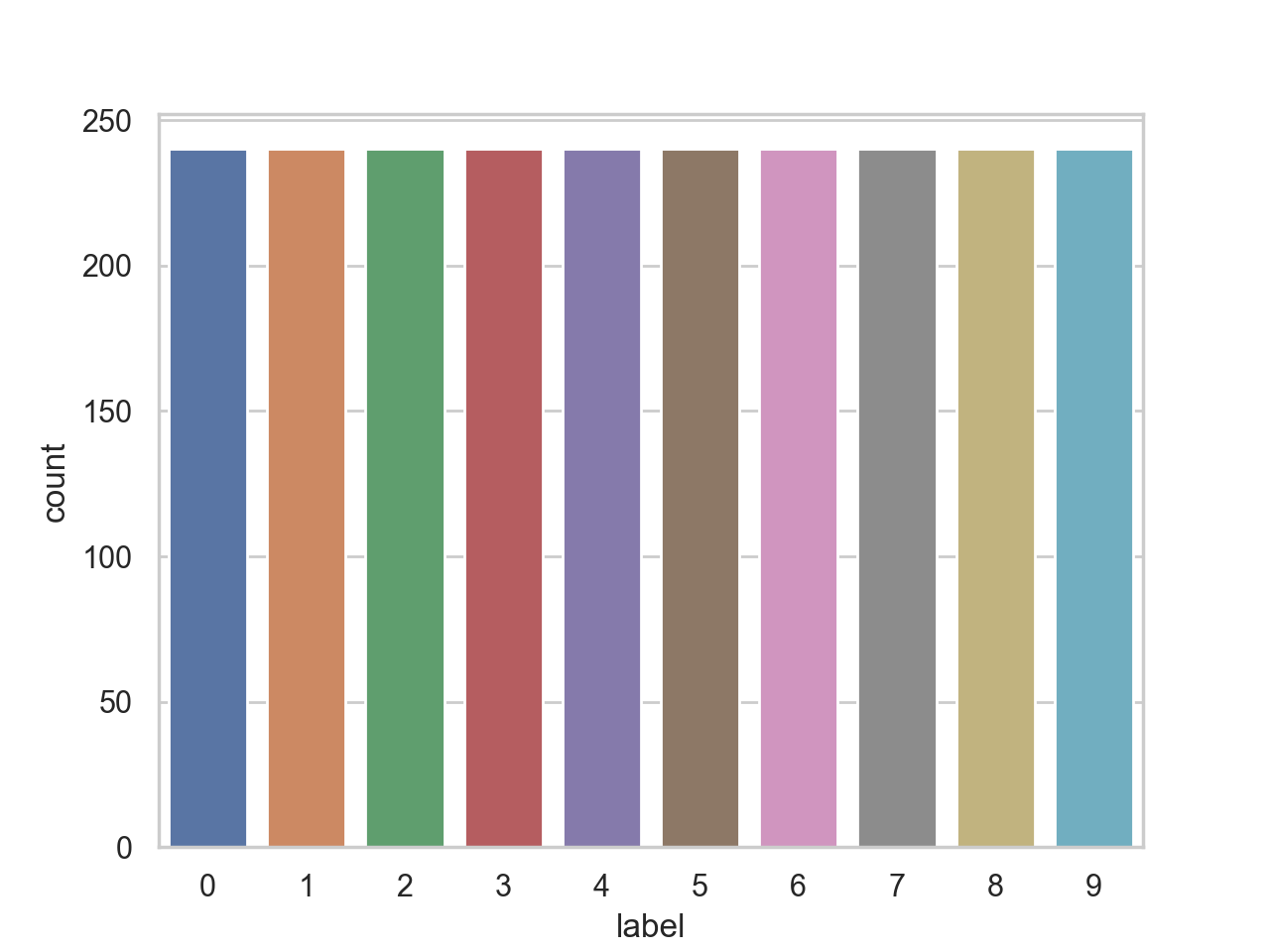
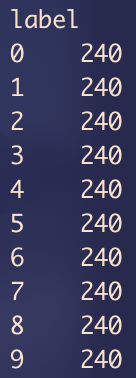
***Instance Selection***

The first objective of the selection is to reduce the number of instances [2]. To do so, we created the function select\_instances() that will compute the minimal number of instances a class attribute is represented by and select this same number out of all the other class attributes representants. The selection is either made by taking the first instances representative of the different class attributes or by picking them randomly. This option can be triggered by specifying the function parameter rd=True or False.

Moreover, the instances are put in random order each time a new computation is done [1]. This is made possible by using the function sample(frac=1) applied to the correct *pandas* data frame. The final datasets are the following (obtained by using print\_head\_tail()).







***Attribute Selection***

The best correlating attributes that were obtained for each label (by using a Boolean mask) are used as the only attributes by choosing either the two, five or ten best attributes for each label (which means 20, 50 or 100 attributes in total).

We can visualise those attributes on the following images:

*best 2 attributes best 5 attributes best 10 attributes*

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

***Arff Conversion***

The conversion of the dataset to .arff files in order to load the dataset in Weka is made through the arff\_converter.py file. As Weka cannot handle large files, the selection is done at generation here following the same guidelines the *pandas* data frame would handle. This converter was used to generate the different datasets as well as the split in train/test. The files are then stored using the following codes:

* sm: Small dataset, after selecting instances (2400 instances usually, 240 per label).
* rd: Randomised dataset.
* ba2: Best 2 (also 5 or 10) attributes dataset.
* num: Numeric labels (default are nominals: label1, label2, etc.).
* train\_70: Training dataset with the ratio.
* test\_30: Testing dataset with the ratio.

Those datasets were used within the explorer but not reused within the experimenter as it allows a percentage split to be used in an automated way. Therefore, if the different files were used within the explorer, only the sm\_rd\_ba2.arff was used in the automation.

# Decision Trees

GLOBAL PART STEPS:

Three classifiers:

* J48
* User Classifier
* Random Forest

***Tree parameters***

Binary splits or Multiple branching, Pruning, Confidence threshold for Pruning, Minimal number of Instances per leaf.

***Training data sets, 10-fold cross-validation***

Accuracy.

Confusion matrix.

Detailed accuracy: TP Rate, FP Rate, Precision, Recall, F Measure, ROC Area.

Tree Parameters.

Visualisation.

***Training and testing datasets (90 training, 10 testing)***

Accuracy.

Confusion matrix.

Detailed accuracy: TP Rate, FP Rate, Precision, Recall, F Measure, ROC Area.

Tree Parameters.

Visualisation.

***Training and testing datasets (70 training, 30 testing)***

Accuracy.

Confusion matrix.

Detailed accuracy: TP Rate, FP Rate, Precision, Recall, F Measure, ROC Area.

Tree Parameters.

Visualisation.

***Training and testing datasets (30 training, 70 testing)***

Accuracy.

Confusion matrix.

Detailed accuracy: TP Rate, FP Rate, Precision, Recall, F Measure, ROC Area.

Tree Parameters.

Visualisation.

# Neural Networks

*Neural Networks* are a special type of classifiers that need an extensive period of training to then perform on data. It is also one of the easiest classifiers to overfit on data due to this period of training. The experiments with *Neural Networks* will be made on the small dataset with the best two attributes for each label.

This part is split in three different parts. The first one presents the reference classifiers for the next experiments, the *Logistic Classifier.* The next part will present the results of using a 10 cross validation experiment on the dataset with a *Multilayer Perceptron* classifier. The parameters will be tuned on this experiment. The next part revolves around the use of a train/test split and its effect on accuracy. The parameters’ impact will be measured around the previous hypothesis made on the 10 cross validation experiment.

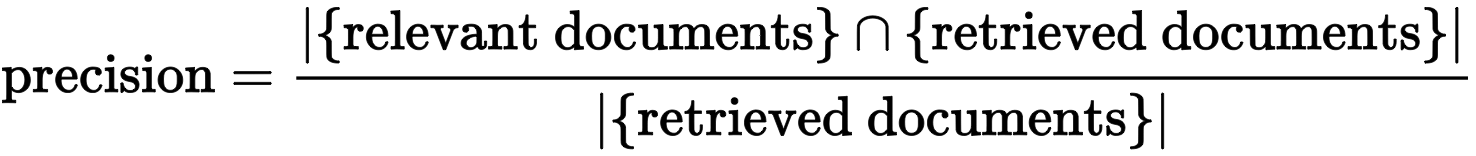
## Logistic Classifier

The *Logistic* classifier is run on the dataset using Weka and the four following parameters: 10 cross validation, percentage split 90%, percentage split 70% and percentage split 30%. The results are shown in the table beneath as well as in the screenshots showing the output:

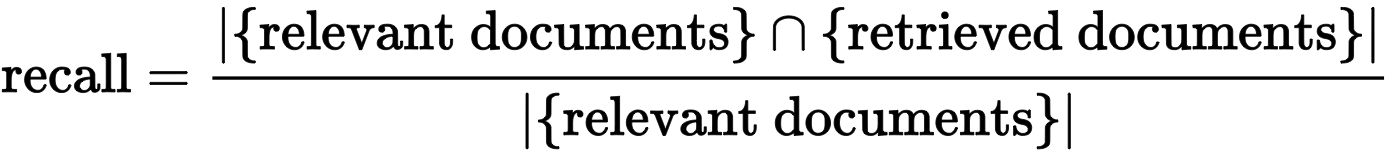
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 10 folds  cross-validation | Percentage split 90% | Percentage split 70% | Percentage split 30% |
| *TP Rate* | 0.821 | 0.817 | 0.843 | 0.764 |
| *FP Rate* | 0.020 | 0.020 | 0.017 | 0.026 |
| *Precision* | 0.821 | 0.825 | 0.842 | 0.761 |
| *Recall* | 0.821 | 0.817 | 0.843 | 0.764 |
| *F Measure* | 0.821 | 0.814 | 0.842 | 0.761 |
| *ROC Area* | 0.972 | 0.975 | 0.874 | 0.953 |
| *Accuracy* | 82.125% | 81.666% | 84.305% | 76.428% |

The different metrics taken in consideration and displayed in the table are the following:

* ***True Positive Rate:*** Rate of the test instances correctly classified as positive.
* ***FP Rate:*** Rate of the test instances incorrectly classified as positive.
* ***F Measure:*** Harmonic mean of the precision and recall.
* ***ROC Area:*** Area under the ROC curve (integration).
* ***Precision:*** Fraction of retrieved documents that are relevant.



* ***Recall:*** Fraction of the relevant documents that are successfully retrieved.



## Experiments on Neural Networks

The same four different methods are used (10 folds cross-validation, percentage split: 90, percentage split: 70 and percentage split: 30). Using the default parameters of the *Multilayer Perceptron*, we will tune the different parameters individually and see their impact on the accuracy and other metrics. The neural network parameters we are going to tune are the following:

* ***Learning rate:*** Hyperparameter that controls the impact on the change of the model once weights are updated
* ***Momentum:*** Value used to avoid the back-propagation algorithm to be stuck on a local minimum.
* ***Validation Threshold:*** Difference between the actual and expected output.
* ***Epochs:*** Number of training steps for the neural network.

The following tuning is made using a 10-folds cross-validation method.

***Learning rate***

The learning rate is tuned from 0.1 to 1.0 (0.1, 0.2, …, 1.0). Its impact is shown on the graphs beneath:

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA screenshot of a social media post

Description automatically generated

Note that only the *Accuracy, False Positive Rate* and *ROC Area* are displayed as the three other metrics have a graph extremely similar to the accuracy one (*True Positive Rate, Precision* and *Recall*).

Conclusion: The learning rate seems to have a negative impact on the different metrics and the performance overall. The default value provided by Weka is 0.3 and we can see it seems to be the best for the accuracy, FP rate and close to the best for the ROC Area (losing 0.04 to the best one).

***Momentum***

The momentum is tuned from 0.1 to 1.0 (0.1, 0.2, …, 1.0). Its impact is shown on the graphs beneath:

***A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated***

***A screenshot of a cell phone

Description automatically generated***

Note that only the *Accuracy, False Positive Rate* and *ROC Area* are displayed as the three other metrics have a graph extremely similar to the accuracy one (*True Positive Rate, Precision* and *Recall*).

Conclusion: The momentum seems to have a negative impact on the different metrics and the performance overall. The default value provided by Weka is 0.2 and we can see it seems to be the best for the accuracy, FP rate and close to the best for the ROC Area (losing 0.04 to the best one).

***Validation Threshold***

The validation threshold is tuned from 10 to 100 (10,20, …, 100). Its impact is shown on the graphs beneath:

***A screenshot of a cell phone

Description automatically generatedA screenshot of a social media post

Description automatically generated***

***A screenshot of a cell phone

Description automatically generated***

Note that only the *Accuracy, False Positive Rate* and *ROC Area* are displayed as the three other metrics have a graph extremely similar to the accuracy one (*True Positive Rate, Precision* and *Recall*).

Conclusion: The variation of the validation threshold did not have any impact on the metrics even though we tried to amplify it by 10 times.

***Epochs***

The number of epochs is tuned from 100 to 1000 (100,200, …, 1000). Its impact is shown on the graphs beneath:

***A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated***

***A screenshot of a social media post

Description automatically generated***

Note that only the *Accuracy, False Positive Rate* and *ROC Area* are displayed as the three other metrics have a graph extremely similar to the accuracy one (*True Positive Rate, Precision* and *Recall*).

Conclusion: The number of epochs corresponds directly to the problematic of how to find the sweet spot where the neural network performs well but does not overfit on the training data. This sweet spot seems to be at 500 epochs (the default value again for Weka) where the *Accuracy* is the best, the *False Positive Rate* is already at the lowest and the *ROC Area* is acceptable as it is only 0.002 away from the best.

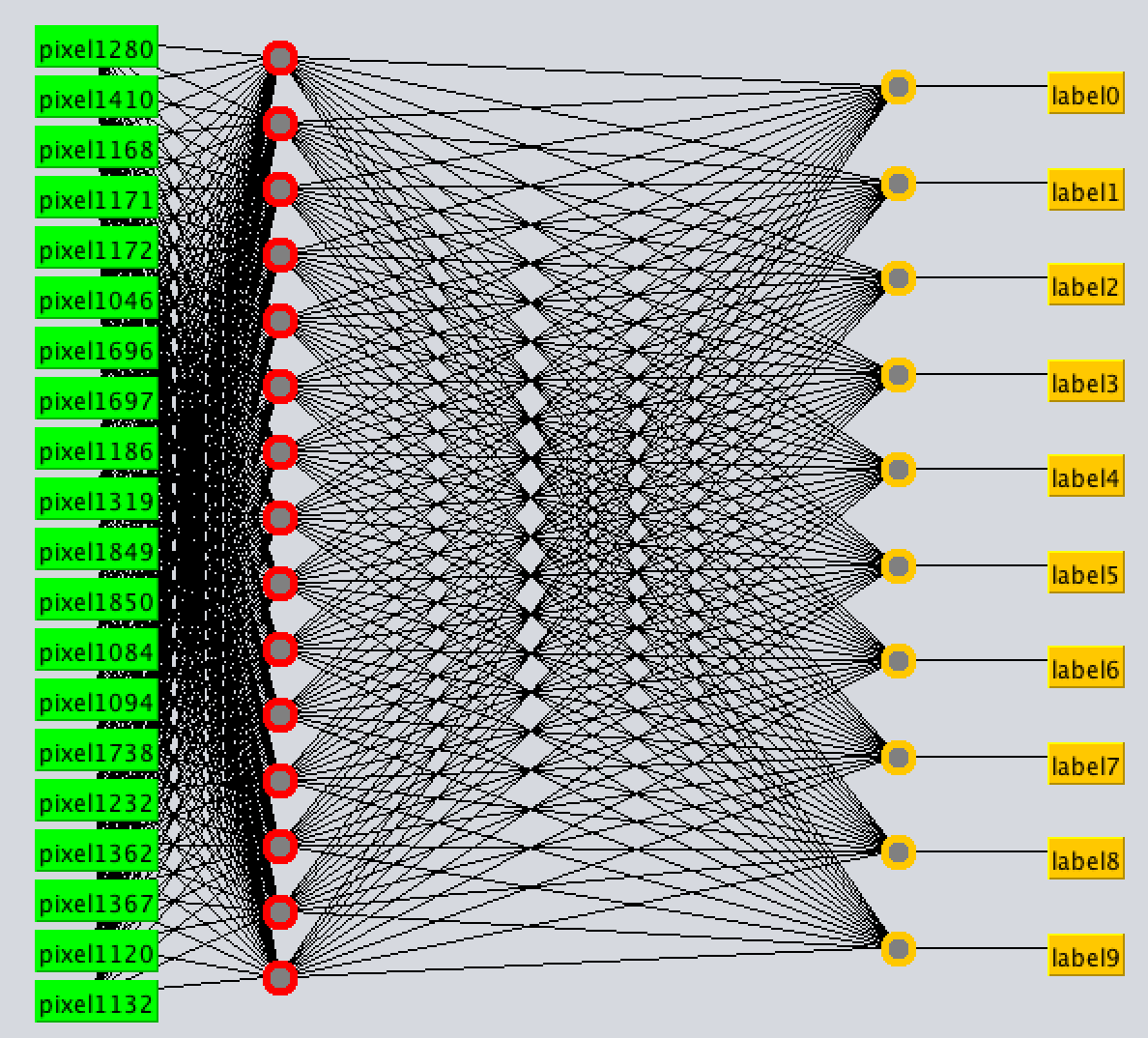
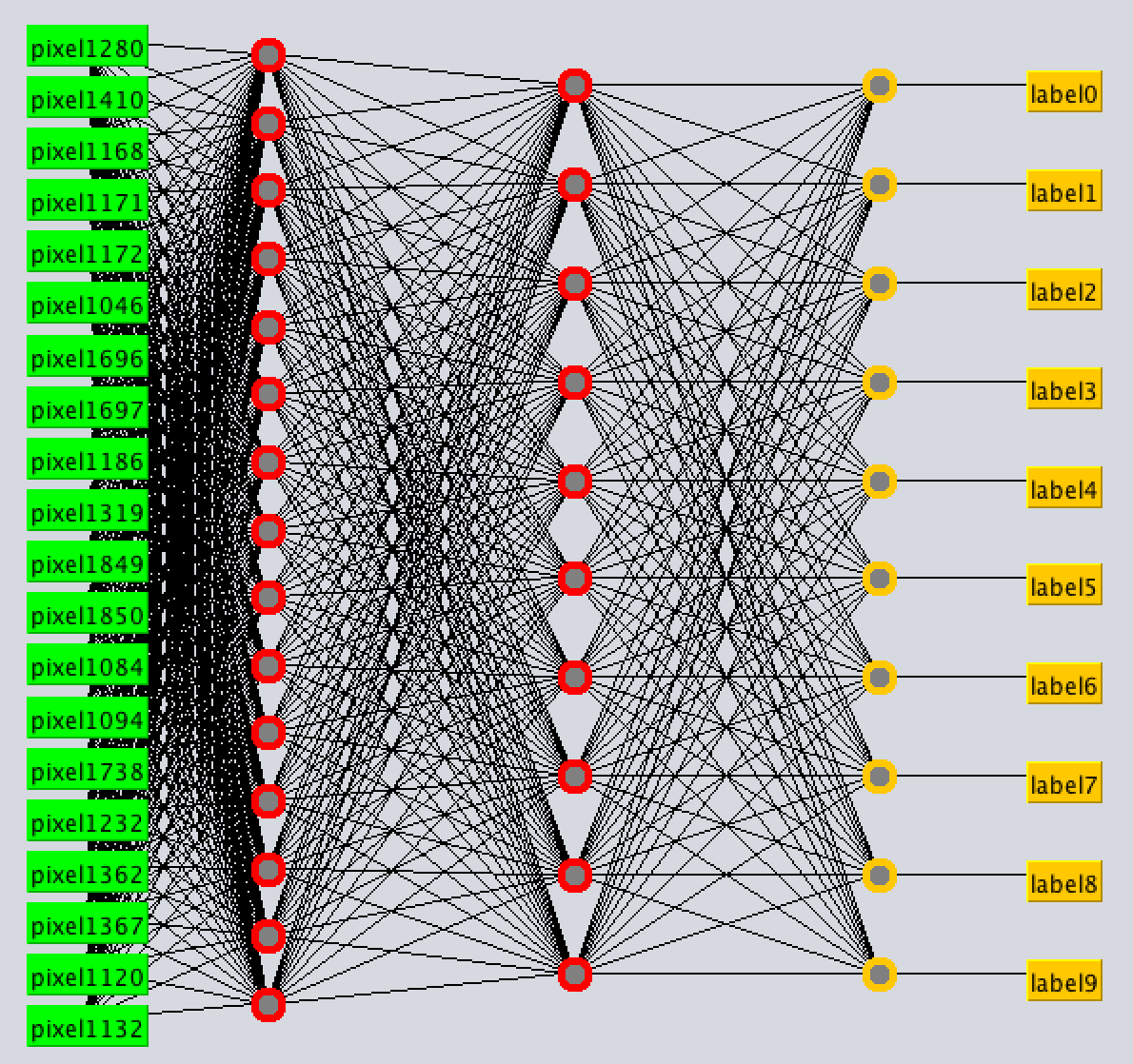
***Nodes and Layers***

The default number of nodes for the hidden layers in Weka is a which corresponds to the number of attributes added to the number of classes, halved overall. In our case and with the corresponding dataset, this means the first layer will have nodes. We will try to tune those numbers by testing different configurations on a maximum of 3 layers.

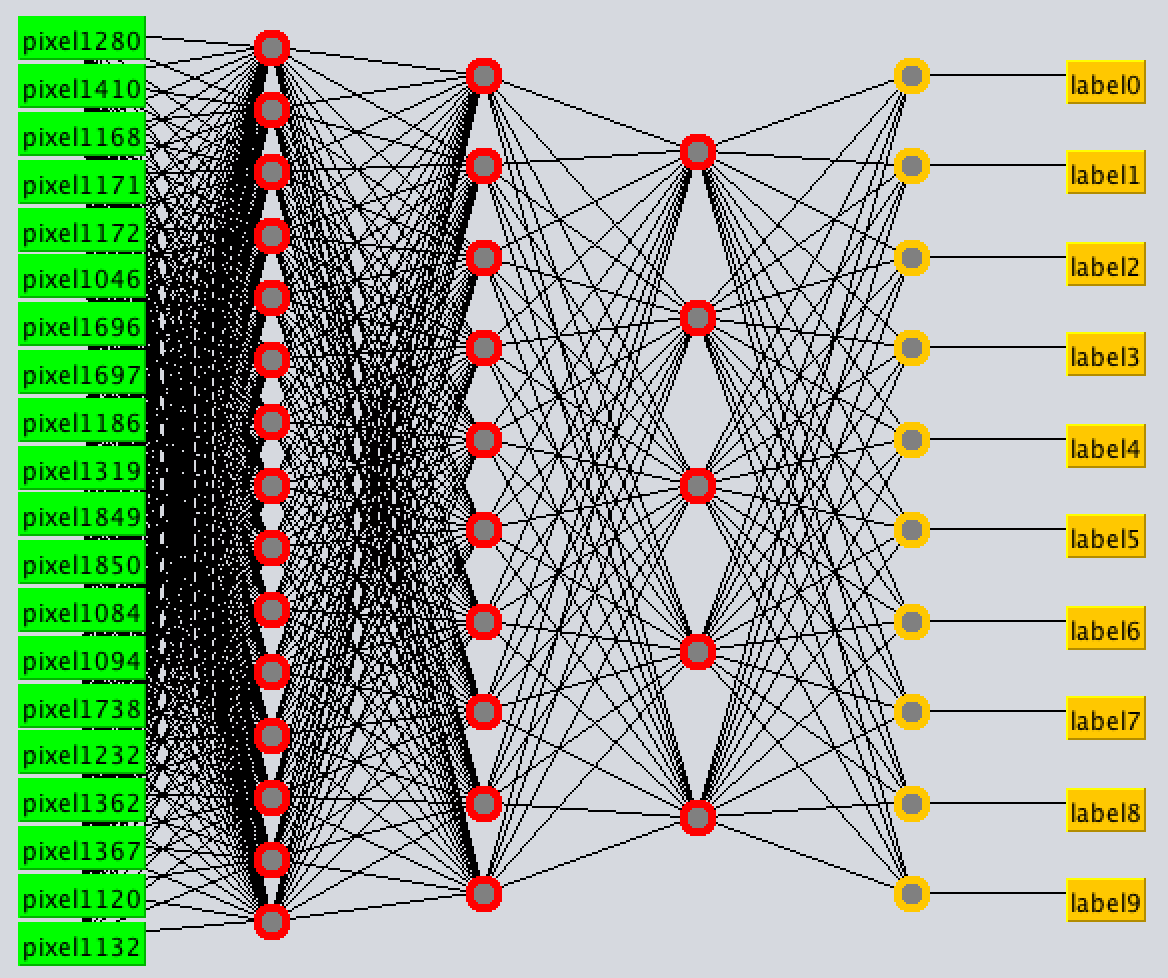
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | a | a,10 | a,10,5 |
| *TP Rate* | 0.776 | 0.833 | 0.803 | 0.764 |
| *FP Rate* | 0.025 | 0.019 | 0.022 | 0.026 |
| *Precision* | 0.777 | 0.833 | 0.806 | 0.770 |
| *Recall* | 0.776 | 0.833 | 0.803 | 0.764 |
| *F Measure* | 0.776 | 0.832 | 0.804 | 0.764 |
| *ROC Area* | 0.955 | 0.966 | 0.958 | 0.936 |
| *Accuracy* | 77.62% | 83.25% | 80.33% | 76.41% |

Visualisation of the different networks:

*a a,10*

*** ***

*a,10,5*



The way to choose the number of nodes and layers is discussed more in depth in the *Research Question*.

***Conclusion on parameters tuning***

The parameters used in this coursework work tightly together as the learning rate and momentum allow the neural network to train faster with the certitude it will not get stuck in local minima. Using a large momentum makes you use a tinier learning rate and vice versa. Then, the number of epochs is important in order to find the sweet spot for the training, a balance between the speed and performance. If a neural network can train and get cross validation results that are outstanding, nothing can assert this dataset was representative and that using it on a real-world dataset will make it stand out. The number of epochs and the validation threshold are here to prevent overfitting and stop the training whenever the neural network performs “*good enough*”. Here, we found that the default values for each parameter are often optimal and allow the other metrics to be at their maximal potential (especially ROC Area and FP Rate in addition to the accuracy as precision, recall and TP Rate are used to compute those). Then, the number of layers and nodes has not been tested extensively due to the fact that this is mostly the subject of our research question. However, in most of the cases, the use of a correct number of nodes in a single hidden layer outperforms several neural networks with numerous layers.

## Experiments on Train/Test Datasets

We can now use the classifier on different datasets and use the percentage split or the dissociation train/test on our previously tuned classifier. As discussed in the previous part, the following values will be used for the different parameters:

* ***Learning rate:*** 0.3
* ***Momentum:*** 0.2
* ***Validation threshold:*** 20
* ***Epochs:*** 500
* ***Nodes/Layers:*** a(15) in only one hidden layer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 10 folds  cross-validation | Percentage split 90% | Percentage split 70% | Percentage split 30% |
| *TP Rate* | 0.833 | 0.846 | 0.826 | 0.779 |
| *FP Rate* | 0.019 | 0.016 | 0.018 | 0.024 |
| *Precision* | 0.833 | 0.848 | 0.833 | 0.785 |
| *Recall* | 0.833 | 0.846 | 0.826 | 0.779 |
| *F Measure* | 0.832 | 0.845 | 0.827 | 0.778 |
| *ROC Area* | 0.966 | 0.974 | 0.969 | 0.955 |
| *Accuracy* | 83.25% | 84.58% | 82.63% | 77.85% |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Handcrafted  Train/test (90%) | Handcrafted  Train/test (70%) | Handcrafted  Train/test (30%) |
| *TP Rate* | 0.813 | 0.814 | 0.774 |
| *FP Rate* | 0.021 | 0.020 | 0.025 |
| *Precision* | 0.814 | 0.833 | 0.787 |
| *Recall* | 0.813 | 0.814 | 0.774 |
| *F Measure* | 0.809 | 0.814 | 0.765 |
| *ROC Area* | 0.966 | 0.959 | 0.953 |
| *Accuracy* | 81.25% | 81.38% | 77.38% |

The tables above present the different configurations under which the classifier was run and the metrics obtained. One of our hypotheses was that the train/test should perform better on a 70/30 ratio than on a 90/10 due to overfitting and better than on a 30/70 due to the training not being effective enough. This hypothesis is not really confirmed as the classifiers have close performances on both the 90/10 and 70/30. The effect of overfitting should be reduced due to the correct choice and tuning of the different parameters discussed earlier.

The different tests made earlier have also been made on the dataset with the different train/test methods by using the *Experimenter* GUI in Weka. The *Experimenter* allows automation of different tasks. This allows to get the best classifier for the different configurations. The automation can be made over several datasets and using a wide number of configurations. We can then reiterate the tuning made earlier on the different datasets (using 2, 5 or 10 best attributes) and varying the neural network parameters.

The first window makes us choose the details of the experiment. Here, we are going to use three different datasets (best 2, 5 and 10 attributes) along with the 20 different configurations corresponding to the earlier parameter tuning.

A screenshot of a computer

Description automatically generated

The third tab provides a view of the results and different methods to rank the classifiers. Here, the classifiers are being clashed between them in order to find the best. In our case, the best one corresponds to the one with the default parameters and a higher number of epochs, 900. The results are all stored in the folder of the project, along with the settings used for the *Experimenter.*

A screenshot of a social media post

Description automatically generated

# Research Question

The research question we chose to ask ourselves was the following, as it was the one point that did not come out clear out of our experiments:

*“How to know the number of nodes and layers we need in our Neural Network?”*

As this question is a wide research subject, we will provide references to article showing actual techniques. The initial idea is that following a small set of rules, one can set up a competent network architecture by both setting the number and the type of both neuronal layers and number of neurons of each layer. The network resulting of this process will be competent, but probably not optimal. However, the real “*tuning”* of the network happens once it is initialised. It is possible to iteratively tune the configuration by either pruning the nodes based on values of the weight vectors (removing small ones and therefore removing redundancy).

Every neural network has three types of layers: input, hidden and output. The input layer is entirely known as long as the shape of the data is known as it should be equal to the number of features (or plus one if the designer wants to add a bias term). The output layer should have one node per class and therefore be known as long as the shape of the data is known as well. However, when talking about hidden layers, several rules-of-thumb exist.

First of all, if the data is linearly separable, there is no need for hidden layers at all, even though there is probably no need for a neural network anyway in that case. Next, *one hidden layer is sufficient in most cases.* This means that in the vast majority of the problems, a single layer is sufficient [1]. Several attempts [2] were made to find a fully automated way for this issue to be solved but the conclusion shows that “*a second layer is desirable even though it relies heavily on the topology chosen. […] the Kanellopoulos-Wilkinson [3] rule gives the best results in terms of accuracy while pruning methods are mostly efficient at simplifying the neural networks and not focusing on optimising accuracy.”*

Finally, the number of nodes is often chosen as the default method in Weka, an empirically derived rule-of-thumb is “*the optimal size of the hidden layer is between the size of the input and size of the output layers”.*

A decent and competent neural network can therefore be created by instantiating it using just two rules:

1. *Number of hidden layers: 1*
2. *Number of neurons: mean(input layer neurons, output layer neurons)*

# Conclusion and Contribution

This coursework felt extremely interesting as it both challenged us to the use of a new language used industry-wise, Python and on the other side let us explore Weka. It was the occasion for us to challenge ourselves to a state-to-the-art issue that can and will be extremely important regarding driverless cars.

The project is hosted on GitHub under <https://github.com/QDucasse/dm_cw2> along with the installation instructions and milestones of the project. In order for it to work, the actual datasets are gathered under a ‘data’ folder but most of it can be generated from the base datasets.

The generated datasets can be found online as they are stored on Dropbox and need to be downloaded at <https://www.dropbox.com/s/habmey8caojkawv/data_cw2.zip?dl=0> then unzipped inside the root project. The project was run under Weka 3.8.3 and Python 3.7.4.

Contribution of the different members:

Decision Trees: Sylvain TOUANEN **100%**

Neural Networks: Quentin DUCASSE **100%**

# References

[1] comp.ai.neural-nets FAQ, Part 1 <http://www.faqs.org/faqs/ai-faq/neural-nets/part1/preamble.html> [visited online the 02/12/2019]

[2] D. Stathakis, “*How many hidden layers and nodes?”*, 2008

[3] I. Kanellopoulos &G. G. Wilkinson, *“Strategies and best practice for neural network image classification”*, 1997