Classification with Weka and Neural Networks

In this exercise, we will explore the use of Weka and Neural Networks for classification. Your task is:

- Select one of the following datasets and read the associated papers:
  1. The OCR dataset discussed during the course
  2. The following dataset on SUSY Monte Carlo simulations: http://archive.ics.uci.edu/ml/datasets/SUSY
  3. Previous agreement by the teacher, a dataset of your own or from this repository: http://archive.ics.uci.edu/ml/datasets.php
- Your goal is to find a good method to classify the objects, as done in the papers. In our terms, you have to find a classification algorithm providing a good classification for the given data. You have to do this by:
  1. Test and select the best classification algorithms in Weka for this task (use the explorer and the experimenter as explained in the sessions)
  2. Design, train and test a Neural Network so you get the best classification (use different topologies, training methods, etc.) and select the most suitable one. You can use JNNS, Python or any other software.
- For this:
  - You can consider reducing the dimensionality of the dataset (e.g. PCA)
  - You have to divide the input file into training/validation files.
- Your starting point is the arff (weka) file provided. This file can directly be used with Weka, but you will have to adapt it to be used with a Neural Network package.
- For NN, to check the results do not simply use the error value given by the software. Your result is the classification of the objects. Therefore, for a given neural network you must analyze the results and give the percent of correctly classified objects (like we did in the classroom), comparable with Weka results.

For this assignment you have to deliver a report describing all the steps you have followed and a summary of all the trials you have done and the best percentage of correct classification you have managed in each case. Include also your comments and conclusions about the task.

1 Selection of the dataset

For this assignment I am going to work with the OCR dataset discussed during the course and studied in [1]. Which has the following form:

```
    The objective is to identify each of a large number of black-and-white
    rectangular pixel displays as one of the 26 capital letters in the
    English alphabet. The character images were based on 20 different fonts
     and each letter within these 20 fonts was randomly distorted to
    produce a file of 20,000 unique stimuli. Each stimulus was converted
    into 16 primitive numerical attributes (statistical moments and edge
    counts) which were then scaled to fit into a range of integer values
    from 0 through 15. We typically train on the first 16000 items and then
     use the resulting model to predict the letter category for the
    remaining 4000.  See the article cited above for more details.

    Number of Instances: 20000

    Number of Attributes: 17 (Letter category and 16 numeric features)


@RELATION letter-recognition

@ATTRIBUTE class {A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z} %
    lettr capital letter (26 values from A to Z)
@ATTRIBUTE x-box REAL    %  horizontal position of box
@ATTRIBUTE y-box REAL   %  vertical position of box
@ATTRIBUTE width-box REAL  % width of box
@ATTRIBUTE heigth-box REAL  % height of box
@ATTRIBUTE npix REAL   % total # on pixels
@ATTRIBUTE x-var-pix REAL  % mean x of on pixels in box
@ATTRIBUTE y-var-pix REAL  % mean y of on pixels in box
@ATTRIBUTE x2bar REAL   % mean x variance
@ATTRIBUTE y2bar REAL   % mean y variance
@ATTRIBUTE xybar REAL   % mean x y correlation
@ATTRIBUTE x2ybr REAL   % mean of x * x * y
@ATTRIBUTE xy2br REAL   % mean of x * y * y
@ATTRIBUTE x-ege REAL   % mean edge count left to right
@ATTRIBUTE xegvy REAL   % correlation of x-ege with y
@ATTRIBUTE y-ege REAL   % mean edge count bottom to top
@ATTRIBUTE yegvx REAL   % correlation of y-ege with x

@DATA
T,2,8,3,5,1,8,13,0,6,6,10,8,0,8,0,8
I,5,12,3,7,2,10,5,5,4,13,3,9,2,8,4,10
D,4,11,6,8,6,10,6,2,6,10,3,7,3,7,3,9
N,7,11,6,6,3,5,9,4,6,4,4,10,6,10,2,8
G,2,1,3,1,1,8,6,6,6,6,5,9,1,7,5,10
S,4,11,5,8,3,8,8,6,9,5,6,6,0,8,9,7
B,4,2,5,4,4,8,7,6,6,7,6,6,2,8,7,10
A,1,1,3,2,1,8,2,2,2,8,2,8,1,6,2,7
J,2,2,4,4,2,10,6,2,6,12,4,8,1,6,1,7
M,11,15,13,9,7,13,2,6,2,12,1,9,8,1,1,8
X,3,9,5,7,4,8,7,3,8,5,6,8,2,8,6,7
O,6,13,4,7,4,6,7,6,3,10,7,9,5,9,5,8
G,4,9,6,7,6,7,8,6,2,6,5,11,4,8,7,8
M,6,9,8,6,9,7,8,6,5,7,5,8,8,9,8,6
R,5,9,5,7,6,6,11,7,3,7,3,9,2,7,5,11
F,6,9,5,4,3,10,6,3,5,10,5,7,3,9,6,9
O,3,4,4,3,2,8,7,7,5,7,6,8,2,8,3,8
.
.
.
```

2 Classification algorithms in Weka:

Testing the classifiers of the Weka explorer performing a 10-fold cross-validation, we obtain the following results:

Classifier	Correctly classified instances (%)	Root mean squared error (%)	Relative absolute error (%)	Precision (%)	True positive rate [Sensitivity] (%)	False positive rate [Specificity] (%)
J48	87,885	9,07	14,3017	87,9	87,9	0,5
Random tree	85,795	10,45	14,7737	85,8	85,8	0,6
Random forest	96,39	6,22	17,66	96,4	96,4	1
Hoeffding tree	64,215	13,94	42,9905	65,6	64,2	1,4
REP tree	84,235	9,78	21,718	84,4	84,2	0,6
Naives bayes	64,01	13,91	43,6368	65,5	64	1,4
Bayes net	74,31	11,87	30,8	75,9	74,3	1
Naive bayes multinomial	54,995	15,33	66,6502	55,4	55	1,8
Naive bayes (updatebale)	64,01	13,91	43,6368	65,5	64	1,4
Naive bayes multinomial (updateable)	54,995	15,33	66,6502	55,4	55	1,8
SMO	82,42	18,61	96,1766	83,1	82,4	0,7
Logistics	77,425	11,27	36,6583	77,3	77,4	0,9
Attribute selected	88,05	8,98	14,3309	88,1	88,1	0,5
Randomizable filtered	85,215	10,66	15,4919	85,3	85,2	0,6
Ibk	95,955	5,51	4,367	96	96	0,2
Kstar	95,885	4,86	6,4948	95,9	95,9	0,2
LWL	41,615	18,72	96,2395	-	41,6	2,4

Table 1: Experimental results of different Weka classifiers acting on the OCR dataset.

a better representations for this data, is the following plot of the root mean squared:

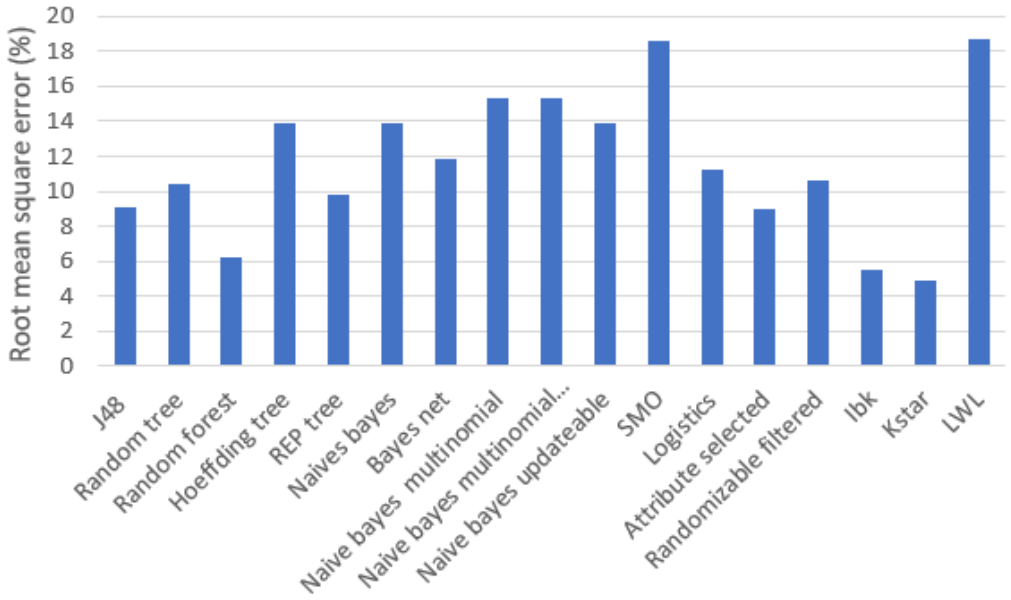

Figure 2.1: Experimental root mean square error(%) of different Weka classifiers, acting on the OCR dataset.

from where we see that Random forest, Ibk and Kstar are the classifiers that perform the best, with accuracy orders of 96% and root mean square errors around 5%.

In the following figures, I will show the classification results for these three best classifiers:

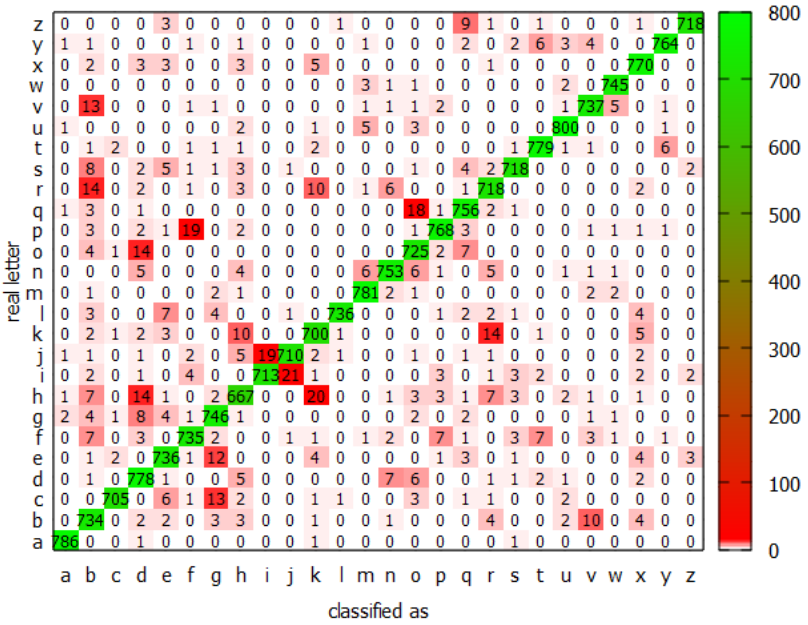

Figure 2.2: Experimental result for the classification of the letters with the Random Forest classifier

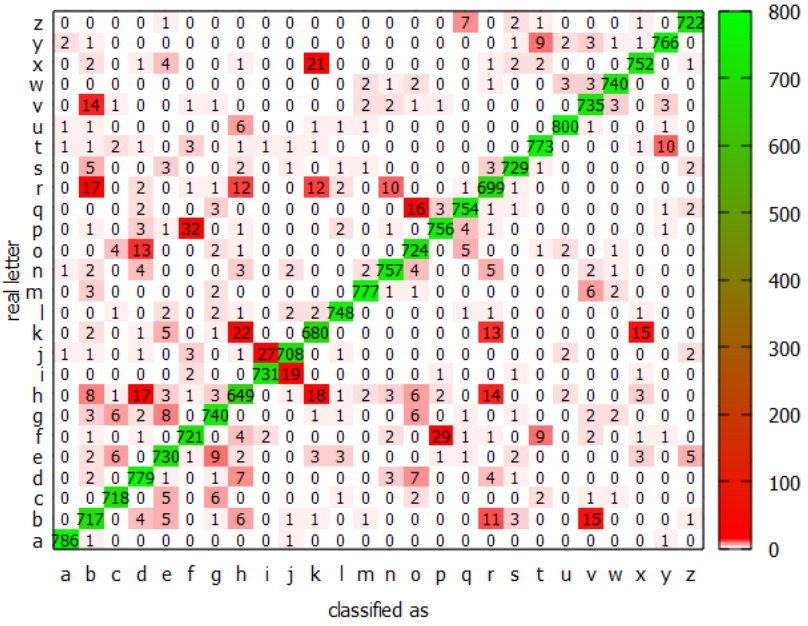

Figure 2.3: Experimental result for the classification of the letters with the Ibk classifier

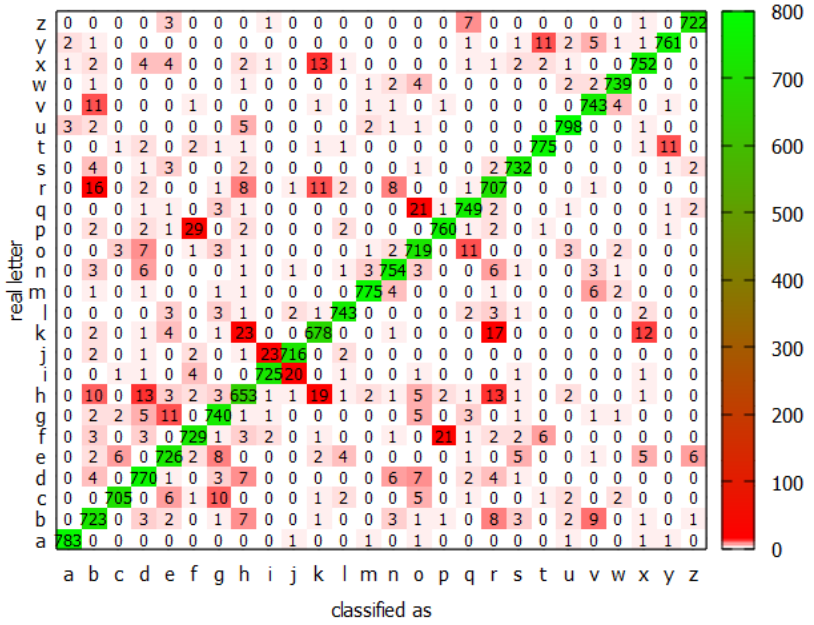

Figure 2.4: Experimental result for the classification of the letters with the KStar classifier

which we see have very similar results, where the most common mistakes are confusing the I-J, the H-K, the P-F, the B-R, the Q-O, the X-K and G-C with varying frequencies in each case but similar in general, as should be expected because the letters are pretty similar.

But we also see a common mistake, that isn't that much intuitive, which is confusing the V-B, letters that doesn't look a like at all, but must have similar variances and means for the pixels distributions I suppose, giving such a common mistake. Another option, could be that the dataset contains mistakes introduced when manually transcripting the data, given the similar sound of both letters.

Another comment to make is that the matrix of results seems pretty symmetric as one would expect, because if you confuse an a with a d it's just normal to confuse d's with a's as well. It is not fully symmetric, maybe because the algorithm tends to go more to one of the letters of the pair, or maybe because by chance most of the confusing fonts are in one of them.

So, concluding, the better Weka classifiers I have found are:

- Random forest: with an accuracy and root mean square error of 96,39% and 6,22% respectively. Which commonest mistakes are classifying P as F, H as K, J as I, I as J and Q as O.
- Ibk: with an accuracy and root mean square error of 95,95% and 5,51% respectively. Which commonest mistakes are classifying F as P, P as F, K as H, H as K, J as I, I as J, R as B, Q as O.
- Kstar: with an accuracy and root mean square error of 95,88% and 4,86% respectively. Which commonest mistakes are classifying F as P, P as F, H as K, K as H, I as J, J as I, B as R, R as B, Q as O and K as R. A bit more symmetric than the previous two.

which make a decent job, classifying correctly around 95 of each 100 letters, making mistakes in the hardest cases, as one would expect.

Finally we would also like to show the classification results for the worst classifier, Lwl:

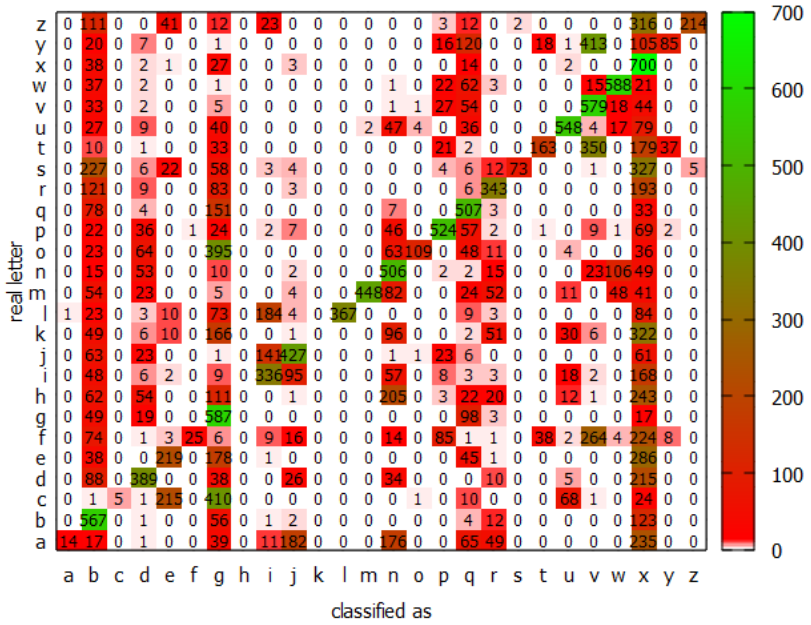

Figure 2.5: Experimental result for the classification of the letters with the Lwl classifier

from where we see that the classifier tends to give more some classification (outputs), independently of the real letter (input), as is the case for the letters B, G, Q, and X, where we see red-brown columns in the Figure 2.5. Also we see the opposite, letters that are given less as result, such as the letters, A, C, F, H, K, P, S, T and Y, where we have white columns, even in the main diagonal even.

In conclusion, it's pretty clear that this algorithm does work acceptably for a few letters, but terribly bad for the majority and shouldn't be used with this type of datasets.

3 Design, train and test of a Neural Network:

Now I am going to train several Neural Networks, and I'm going to compare them to the Ibk and Kstar algorithms from the previous sections, which were the bests.

So, first let's try building the algorithms/training the Neural Networks with 75% of the data (15000 letters) and using the 25% remaining (5000 letters) to test them, obtaining:

Classifier:	Accuracy(%):
Ibk	96,6
Kstar	95,14
NN(16-100-100-8-100-32-26)[200]	94,44
NN(16-1024-512-256-128-64-52-26)[100]	94,04
NN(16-1024-512-256-128-64-52-26)[200]	94,7

And we are going to do the same with 95% of the data (19000 letters) for training and the remaining 5% (1000 letters) for testing, obtaining these other results:

Classifier:	Accuracy(%):
Ibk	96,3
Kstar	96,1
NN(16-20-30-26)[200]	90,9
NN(16-100-52-26)[200]	94,8
NN(16-100-100-8-100-32-26)[100]	94,499
NN(16-100-100-8-100-32-26)[200]	93,599
NN(16-100-200-16-200-52-26)[100]	94,099
NN(16-100-200-16-200-52-26)[200]	95,499
NN(16-100-32-400-200-100-52-26)[200]	95,899
NN(16-200-1000-52-2000-1000-200-26)[200]	94,59
NN(16-2048-1024-512-256-128-52-26)[100]	95,399
NN(16-1024-512-256-128-64-52-26)[75]	93,3
NN(16-1024-512-256-128-64-52-26)[100]	96,2
NN(16-1024-512-256-128-64-52-26)[600]	95,49

where $\mathrm{NN}(16-(m^{(1)}+1)-...-(m^{(L)}+1)-26)[X]$ is a X epochs trained Neural Network, with this topology:

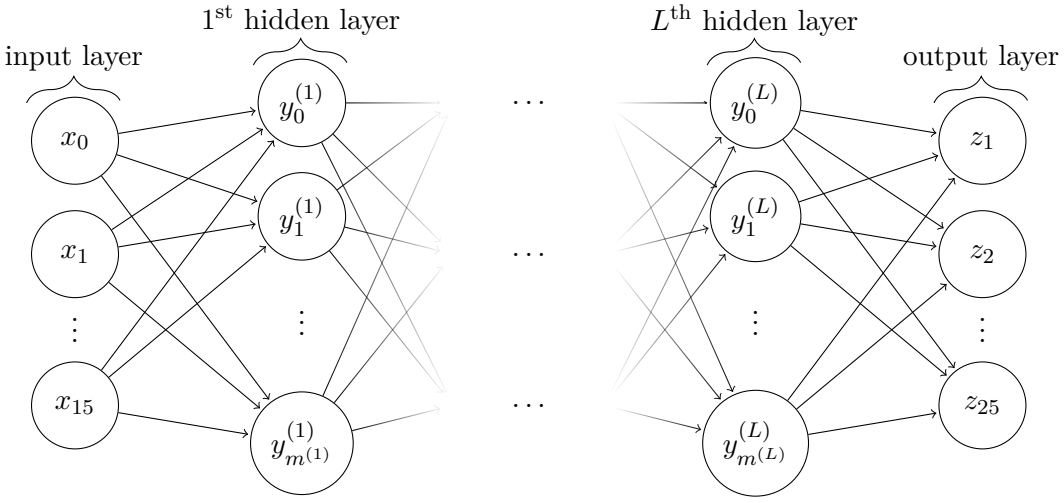

Figure 3.1: Network graph of a $(L+1)$ layer perceptron with 16 input units (parameters of our letters) and 26 output units (number of letters in Spanish alphabet). The l^{th} hidden layer contains $m^{(l)}$ hidden units.

From the results, we see that the Neural Networks are at the level of the best algorithms of the previous sections. With pretty similar results for the training-testing ratios 75%-25% and 95%-5%.

The results contains NN with "bottle necks" or without, with bigger number of neurons or fewer in each layer, with more or less epochs of training, and even with more or few layers, from which we conclude, that the accuracy is pretty similar for all the cases, given a minimum complexity (2 hidden layers with 50-100 neurons). What in general actually becomes important for this accuracy, is the number of epochs we train the NN, at least until we arrive to 100 epoch where it starts oscillating at a constant value around 95%:

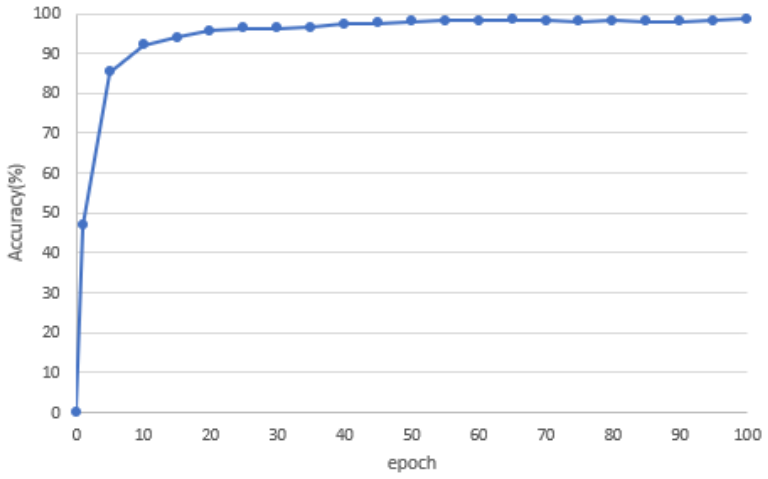

Figure 3.2: Accuracy(%) obtained for NN(16-1024-512-256-128-64-52-26)[100] depending of the epochs of training.

where we see that for 1 epoch we are almost at 50% accuracy (already better than some algorithm from the previous section), and for 5 epoch we are at orders of 85% of accuracy (competing with almost all the algorithms).

Now let's compare with the previous section algorithms, with:

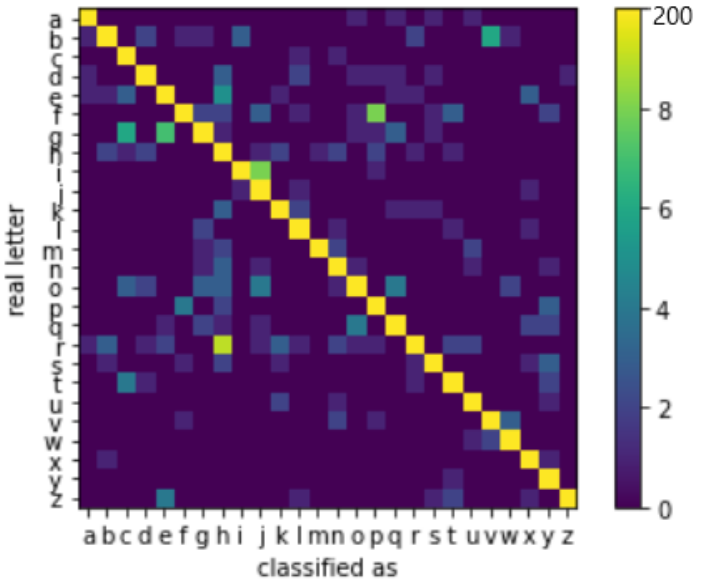

Figure 3.3: Experimental results for the classification of the letters with NN(16-1024-512-256-128-64-52-26)[200] for 75%-25% rate of training-testing.

which is not at the same scale than the previous ones (visually only works for a qualitative comparison, not quantitative). If we want to compare them more precisely, we have to take into account that there is 1/4 of the total number of testing in this case, so the super reds 20's-30's of the previous graphics would correspond to slightly greens of the order of 5's-8's in this one.

The results then approximately shows us that the NN, doesn't confuse so much the H-K as the algorithms did, but instead confuses H-R more, or G-E. Also we see that there is a kind of symmetry but not totally, as we discussed before. Another interesting things is that the NN also confuses V-B which was already weird with the algorithms.

Finally show the approximate topology of the NN that gave the best accuracy overall, which is NN(16-1024-512-256-128-64-52-26)[100]:

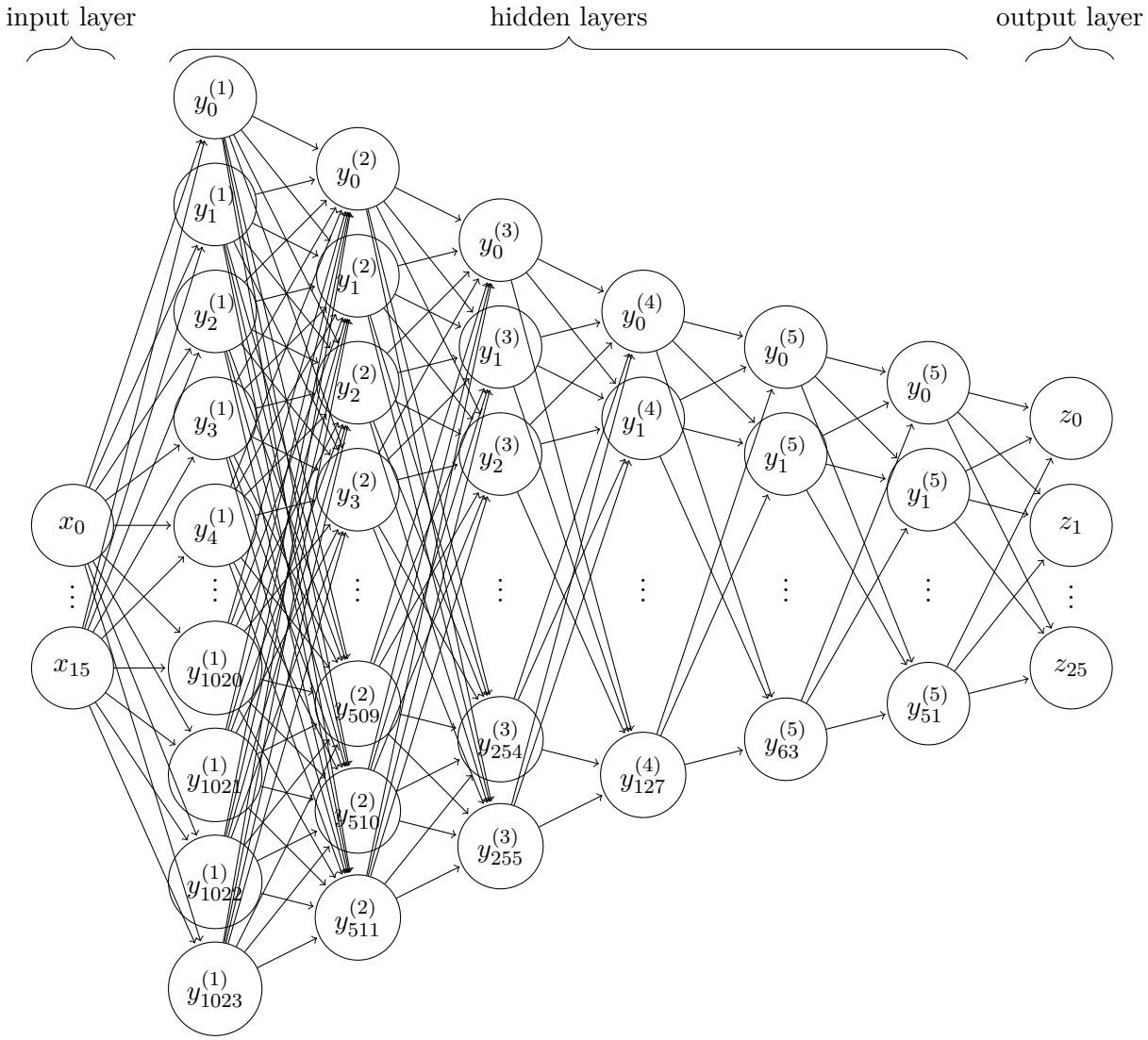

Figure 3.4: Network graph of a 5 layer perceptron with 16 input units (parameters of our letters) and 26 output units (number of letters in Spanish alphabet). The l^{th} hidden layer contains $m^{(l)}$ hidden units.

4 Conclusions

Concluding, we have seen that with both, the algorithm and the NN we can obtain very satisfactory results, of the order of 95% of Accuracy.

But there is a big difference between them. In the case of the algorithms some gave very bad results, when in the case of NN for a minimum intuitively complexity, all gave excellent results. So if I had to chose I would go with NN's to save me from problems of choosing a bad Weka classifier.

Also, mention, that we used supervised learning, which needed the "class" of some of the letters to train the NN, as did the algorithms of Weka to buil it's models. But if we did not had the "class" of the letters, we could still have used unsupervised learning to find patterns in the data and classify them in groups, and with a quick posterior human analysis, find which letters where each group. Which wouldn't be possible with the Weka classifiers.

From the assignment I think it's clear why, machine learning, will dominate the future, given it's adaptability and great performance in the majority of cases without the need to be specific of what it should do.

References

[1] P. W. Frey, D. J. Slate, and T. Dietterich, "Letter recognition using holland-style adaptive classifiers," in *Machine Learning*, 1991.

Used codes

Copy of the code of the Neural Network [python]:

(full Jupiter Notebook attached as main.ipynb)

```python
#IMPORT ALL THE PACKAGES...
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models
from datetime import datetime
from scipy.io import arff
import pandas as pd

#IMPORT THE DATA SET TO USE WITH TENSORFLOW
data = arff.loadarff('letter-recognition.arff')
df = pd.DataFrame(data[0])

df['class'] = df['class'].apply(lambda x : ord(x)-65)

Ntraining=15000 #Total is 20k

training_images=df.drop(columns='class')[:Ntraining]
testing_images=df.drop(columns='class')[Ntraining:]

training_labels=df['class'][:Ntraining]
testing_labels=df['class'][Ntraining:]

#CREATE THE NN (WITH THE DIFFERENT LAYERS)
model=models.Sequential()
model.add(layers.Dense(16,activation='relu',input_shape = (16,)))
model.add(layers.Dense(1024,activation= 'relu'))
model.add(layers.Dense(512,activation= 'relu'))
model.add(layers.Dense(256,activation= 'relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(64,activation= 'relu'))
model.add(layers.Dense(52,activation= 'relu'))
#Finally let's get probabilities
model.add(layers.Dense(26,activation='softmax'))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

#We say how many times the NN will see the training data (epochs)
model.fit(training_images, training_labels, epochs=200, validation_data=(
    testing_images, testing_labels))

loss, accuracy = model.evaluate(testing_images, testing_labels)
print(f"Loss: {loss}")
print(f"Accuracy: {accuracy}")

model.save('letter_classifier4.model')


#USE AN ALREADY TRAINED NN
Itry=Ntraining
counter= 0

#Give names to the different types 1, type 2 ... type 26
class_names = ['A', 'B', 'C', 'D', 'E', 'D', 'F', 'G', 'H', 'I', 'J', 'K',
    'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'V', 'W', 'X', 'Y', 'Z']

model = models.load_model('letter_classifier4.model')
```

```

CM = np.zeros((26, 26))
#Try the NN for the testing values
for i in range(20000-Ntraining):
    prediction= model.predict(df.drop(columns='class').iloc[[Itry+i]])
    index=np.argmax(prediction)

    if (index==int(df['class'].iloc[[Itry+i]])):
        counter=counter + 1
    print(counter)

    CM[index][int(df['class'].iloc[[Itry+i]])]= CM[index][int(df['class'].iloc[[Itry+i]])]+1
    plt.xticks
    ([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25],
     ["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"])
    plt.yticks
    ([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25],
     ["a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"])
    plt.xlabel('classified as')
    plt.ylabel('real letter')
    plt.imshow(CM)
    plt.clim(0, 10)
    plt.colorbar()
    plt.show()

#Get the accuracy over the testing sample
print(counter/5000, 'which is equal to Accuracy given by the model.evaluate')
```

Code used for the matrix plotting of the experimental results [gnuplot]:

```
set xrange[-0.5:25.5]
set yrange[-0.5:25.5]
set palette defined (0 'white',0.02 'red', 1 'green')

set ylabel 'real letter'
set xlabel 'classified as'

set xtics ("a"0,"b"1,"c"2,"d"3,"e"4,"f"5,"g"6,"h"7,"i"8,"j"9,"k"10,"l"11,"m"12,"n"13,"o"14,"p"15,"q"16,"r"17,"s"18,"t"19,"u"20,"v"21,"w"22,"x"23,"y"24,"z"25)
set ytics ("a"0,"b"1,"c"2,"d"3,"e"4,"f"5,"g"6,"h"7,"i"8,"j"9,"k"10,"l"11,"m"12,"n"13,"o"14,"p"15,"q"16,"r"17,"s"18,"t"19,"u"20,"v"21,"w"22,"x"23,"y"24,"z"25)

plot "CM_ibk.txt" matrix with image notitle,'CM_ibk.txt' matrix using 1:2:(sprintf('%.0f', $3)) with labels font ',7' notitle
```

Code for the visualization of the Neural Networks topology [Latex]:

```
\begin{figure}[H]
  \centering
  \begin{tikzpicture}[shorten >=1pt]
    \tikzstyle{unit}=[draw,shape=circle,minimum size=1.15cm]
    %\tikzstyle{hidden}=[draw,shape=circle,fill=black!25,minimum size=1.15cm]
    \tikzstyle{hidden}=[draw,shape=circle,minimum size=1.15cm]

```

```
    \node[unit](x0) at (0,3.5){$x_0$};
    \node[unit](x1) at (0,2){$x_1$};
    \node at (0,1.1){\vdots};
    \node[unit](xd) at (0,0){$x_{16}$};

    \node[hidden](h10) at (3,4){$y_0^{(1)}$};
    \node[hidden](h11) at (3,2.5){$y_1^{(1)}$};
    \node at (3,1.2){\vdots};
    \node[hidden](h1m) at (3,-0.5){$y_{m^{(1)}}^{(1)}$};

    \node(h22) at (5,0){};
    \node(h21) at (5,2){};
    \node(h20) at (5,4){};

    \node(d3) at (6,0){$\ldots$};
    \node(d2) at (6,2){$\ldots$};
    \node(d1) at (6,4){$\ldots$};

    \node(hL12) at (7,0){};
    \node(hL11) at (7,2){};
    \node(hL10) at (7,4){};

    \node[hidden](hL0) at (9,4){$y_0^{(L)}$};
    \node[hidden](hL1) at (9,2.5){$y_1^{(L)}$};
    \node at (9,1.2){\vdots};
    \node[hidden](hLm) at (9,-0.5){$y_{m^{(L)}}^{(L)}$};

    \node[unit](y1) at (12,3.5){$z_1$};
    \node[unit](y2) at (12,2){$z_2$};
    \node at (12,1.1){\vdots};
    \node[unit](yc) at (12,0){$z_{26}$};

        \draw[->] (x0) -- (h10);
    \draw[->] (x0) -- (h11);
    \draw[->] (x0) -- (h1m);

        \draw[->] (x1) -- (h10);
    \draw[->] (x1) -- (h11);
    \draw[->] (x1) -- (h1m);

    \draw[->] (xd) -- (h10);
    \draw[->] (xd) -- (h11);
    \draw[->] (xd) -- (h1m);

    \draw[->] (hL0) -- (y1);
    \draw[->] (hL0) -- (yc);
    \draw[->] (hL0) -- (y2);

    \draw[->] (hL1) -- (y1);
    \draw[->] (hL1) -- (yc);
    \draw[->] (hL1) -- (y2);

    \draw[->] (hLm) -- (y1);
    \draw[->] (hLm) -- (y2);
    \draw[->] (hLm) -- (yc);

    \draw[->,path fading=east] (h10) -- (h20);
    \draw[->,path fading=east] (h10) -- (h21);
    \draw[->,path fading=east] (h10) -- (h22);

    \draw[->,path fading=east] (h11) -- (h20);
    \draw[->,path fading=east] (h11) -- (h21);
    \draw[->,path fading=east] (h11) -- (h22);
```

```
71
72      \draw[->,path fading=east] (h1m) -- (h20);
73      \draw[->,path fading=east] (h1m) -- (h21);
74      \draw[->,path fading=east] (h1m) -- (h22);
75
76      \draw[->,path fading=west] (hL10) -- (hL0);
77      \draw[->,path fading=west] (hL11) -- (hL0);
78      \draw[->,path fading=west] (hL12) -- (hL0);
79
80      \draw[->,path fading=west] (hL10) -- (hL1);
81      \draw[->,path fading=west] (hL11) -- (hL1);
82      \draw[->,path fading=west] (hL12) -- (hL1);
83
84      \draw[->,path fading=west] (hL10) -- (hLm);
85      \draw[->,path fading=west] (hL11) -- (hLm);
86      \draw[->,path fading=west] (hL12) -- (hLm);
87
88      \draw [decorate,decoration={brace,amplitude=10pt},xshift=-4pt,yshift=0
        pt] (-0.5,4) -- (0.75,4) node [black,midway,yshift=+0.6cm]{input layer
        };
89      \draw [decorate,decoration={brace,amplitude=10pt},xshift=-4pt,yshift=0
        pt] (2.5,4.5) -- (3.75,4.5) node [black,midway,yshift=+0.6cm]{$1^{\text
        {st}}$ hidden layer};
90      \draw [decorate,decoration={brace,amplitude=10pt},xshift=-4pt,yshift=0
        pt] (8.5,4.5) -- (9.75,4.5) node [black,midway,yshift=+0.6cm]{$L^{\text
        {th}}$ hidden layer};
91      \draw [decorate,decoration={brace,amplitude=10pt},xshift=-4pt,yshift=0
        pt] (11.5,4) -- (12.75,4) node [black,midway,yshift=+0.6cm]{output
        layer};
92    \end{tikzpicture}
93    \caption[Network graph for a $(L+1)$ layer perceptron.]{Network graph of
        a $(L+1)$ layer perceptron with $16$ input units (parameters of our
        letters) and $26$ output units (number of letters in Spanish alphabet).
         The $l^{\text{th}}$ hidden layer contains $m^{(l)}$ hidden units.}
94    \label{fig:multilayer-perceptron}
95  \end{figure}
```